

UNIVERSIDAD DE ALCALÁ
DEPARTAMENTO DE AUTOMÁTICA

WHERE DO CAPTCHAS FAIL:
A STUDY IN COMMON PITFALLS IN
CAPTCHA DESIGN
AND HOW TO AVOID THEM

Dissertation written by
Carlos Javier Hernández Castro

Under the supervision of
María Dolores Rodríguez Moreno, PhD
David Fernández Barrero, PhD

Dissertation submitted to the Polytechnic Superior School of the
University of Alcalá, in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy
October 2017



Universidad
de Alcalá

DEPARTAMENTO DE AUTOMÁTICA
Edificio Politécnico
28805 Alcalá de Henares (Madrid)
Teléfono: 91 885 65 94
Fax: 91 885 69 23
secre.aut@uah.es

Dra. D^a. María Dolores Rodríguez Moreno, Titular de Universidad del Área de Arquitectura y Tecnología de Computadores de la Universidad de Alcalá y Dr. David Fernández Barrero, Titular de Universidad Interino del Área de Arquitectura y Tecnología de Computadores

INFORMAN: Que la Tesis Doctoral titulada **“Where do CAPTCHAs fail: A study in common pitfalls in CAPTCHA design and how to avoid them”**, presentada por D. Carlos Javier Hernández Castro y realizada en el Departamento de Automática bajo nuestra dirección, reúne méritos suficientes para optar al grado de Doctor, por lo que puede procederse a su depósito y lectura.

Alcalá de Henares, 6 de octubre de 2017



Fdo: Dra. D^a. María Dolores Rodríguez Moreno Fdo: Dr. D. David Fernández Barrero



Universidad
de Alcalá

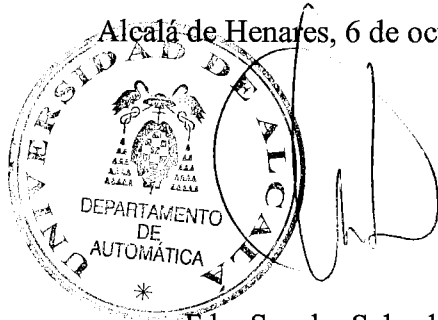
DEPARTAMENTO DE AUTOMÁTICA
Edificio Politécnico
28871 Alcalá de Henares (Madrid)
Teléfono: 91 885 65 94
Fax: 91 885 69 23
secre@aut.uah.es

Dr. D. Sancho Salcedo Sanz, Profesor Titular de la Universidad de Alcalá y Coordinador del Programa de Doctorado "Tecnologías de la Información y las Comunicaciones"

INFORMA:

Que la Tesis Doctoral titulada **"Where do CAPTCHAs fail: A study in common pitfalls in CAPTCHA design and how to avoid them"**, presentada por D. Carlos Javier Hernández Castro y dirigida por la Doctora D^a. María Dolores Rodríguez Moreno y el Doctor D. David Fernández Barrero, cumple con todos los requisitos científicos y metodológicos para ser defendida ante un tribunal.

Alcalá de Henares, 6 de octubre de 2017



Fdo. Sancho Salcedo Sanz

Acknowledgements

This thesis and the associated research has implied a sustained effort, mostly during off-work hours. It would have not been possible without the help and encouragement of several key people, to whom I am especially grateful. Some of them have contributed directly, while others indirectly, but all their contributions have led me to the completion of this work.

Mi madre, María del Carmen, me ha enseñado tanto. Sobre todo, me enseñó el interés por aprender. Ella puso la semilla de mi aprendizaje, y aún me enseña.

My brother Julio, who sparked my curiosity and showed to me the thrill of the discovery. He is my example on what a researcher should be.

My soul-mate Женя, who has encouraged me in every step of the way. She has supported me through the process with incredible strength, patience and wisdom, and very few водка shots. More importantly, she shows to me a better world worth of every effort.

Mom fis Declan, qui même s'il n'est pas lié à ce doctorat, il m'aide à avancer.

Last but importantly, to my tutors David and Malola. They have guided me carefully, with a perfect balance of freedom and guidance. They have encouraged me in the difficult moments, helped me to enhance every article

and this thesis. It has been a pleasure to work under their supervision.

This thesis would not have been possible without all of them.

I also want to thank:

Вова, кто, к сожалению, здесь не для того, чтобы увидеть конец этой работы, но кто я знаю, будет счастлив.

My father Julio, who encouraged me to be the best I could, and sacrificed for our education.

Abstract

Today, much of the interaction between clients and providers has moved to the Internet. Some tricksters, con-artists and charlatans have also learned to benefit from this new situation. New improved cons, tricks and deceptions can be found on-line. Many of these deceptions are only profitable if they are done at a large scale. In order to achieve these large numbers of interactions, these attacks require automation.

CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) or HIPs (Human Interaction Proofs) are a relatively new security mechanism against automated attacks. They try to detect when the other end of the interaction is a human or a computer program (a *bot*). Since their origins, most of the proposals have been based on the seminal idea of using problems thought to be hard for AI/ML but easy for humans. As of today, all the studied CAPTCHA schemes have failed.

CAPTCHA design is still in its initial conception. The stream of successful attacks on them are a hint that CAPTCHA are now as weak as the first cyphers. Yet cyphers were improved after successive successful cryptanalysis. We consider that similarly new security studies in novel, original CAPTCHAs will advance the corpus of knowledge in the field as well as the awareness about CAPTCHA security.

This dissertation focuses on the design of CAPTCHAs. Its first goal is to understand whether there are currently CAPTCHAs that can be considered secure. To do so, it analyses new, original CAPTCHA proposals. The second goal of this dissertation is to find a way in which to assess a basic level of security for new CAPTCHA designs. To do so, it studies the results of previous security analysis trying to find common weaknesses. Based on them, it proposes a guideline or framework that specifies mechanisms to avoid some of these design pitfalls. This can be the starting point for a high-level methodology for the design of new CAPTCHAs. Ultimately, the goal of this research is to build a semi-automatic framework for the analysis of the security of new CAPTCHAs.

Resumen Ampliado

El uso de Internet es creciente tanto en número de usuarios como de servicios proporcionados. Existe también un uso social y lúdico. Cada vez, más aspectos de nuestra vida son totalmente *en línea* (en Internet) o tienen una parte en línea. Esto representa un gran potencial no sólo para las empresas que gestionan estos servicios y datos, sino también para quien puede encontrar una forma de aprovecharse de ellos. Hasta ahora, una forma típica consiste en aprovecharse de servicios gratuitos o información disponible libremente. Un ejemplo sería una votación en línea. Realizar un voto no tiene mayor trascendencia. Pero controlar el resultado de la votación puede ser interesante, sobre todo si hay un premio en juego, o la votación tiene repercusiones en términos de reputación o influencia. Existen muchos otros ejemplos, incluyendo la infiltración en redes sociales, abuso de cuentas de *web-mail*, abuso de servicios en la nube, reservas en línea, etc.

El abuso manual, a pequeña escala, no es viable económicamente. Para que sea eficaz es necesario poder realizar una gran cantidad de interacciones, y normalmente esto sólo es rentable si dichas interacciones son automáticas: no son realizadas por humanos, sino por programas de ordenador (*bots*).

Los llamados CAPTCHAs (Test de Turin Público y Automático para Diferenciar Computadores de Humanos, o *Completely Automated Public*

Turing test to tell Computers and Humans Apart) o HIPs (Tests de Interacción con Humanos, o *Human Interaction Proofs*) son una medida de seguridad esencial contra ataques automáticos en Internet. Fueron propuestos por primera vez por Mori Naor en 1996 e implementados por primera vez por Andrei Broder en el buscador Altavista en 1997.

Inicialmente los CAPTCHAs estuvieron vinculados a lo que se percibían como las limitaciones del Aprendizaje Automático (ML) de la época. Sin embargo, esta idea no ha tenido gran éxito: desde sus orígenes hasta ahora, todos los CAPTCHAs que han sido analizados han sido atacados con éxito, ya haya sido mediante ataques *de canal lateral* como mediante ataques directos basados en algoritmos específicos o en mejoras en ML. Ningún CAPTCHA ha resistido, en el mismo formato, más de alguna decena de meses.

En nuestra opinión, el diseño de CAPTCHAs está en su fase inicial, de forma similar a cuando se diseñaron los primeros sistemas de cifrado hace miles de años. Estos sistemas de cifrado fueron mejorando tras cada criptoanálisis. Esperamos que de forma similar, el análisis de la seguridad de los CAPTCHAs actuales ayude a incrementar la seguridad de los venideros.

El principal objetivo de esta tesis se centra en el diseño de CAPTCHAs seguros. Intenta responder a la pregunta de si actualmente existen formas de crear CAPTCHAs que sean seguros. Para ello, analizaremos la seguridad de nuevos CAPTCHAs que sean originales e interesantes desde el punto de vista de su diseño, seguridad o usabilidad. La razón principal por la que elegiremos estos CAPTCHAs es porque los ataques a otros CAPTCHAs anteriores no son, en principio, extrapolables a ellos, ya sea porque los nuevos diseños se crean de manera sean resistentes a las técnicas usadas en los ataques conocidos,

o porque son diseños tan originales que caen fuera del ámbito de dichos ataques. Por ello se requieren nuevos análisis de seguridad. Analizaremos estos CAPTCHAs buscando vulnerabilidades, es decir, formas en las que estos CAPTCHAs filtran información que permita un ataque. De esta forma, esperamos contribuir al conjunto de conocimiento en el campo del diseño de CAPTCHAs.

El segundo objetivo de esta tesis es encontrar formas de comprobar cierto nivel de seguridad para diseños de CAPTCHAs que sean totalmente nuevos. Para ello, analizaremos los resultados de nuestros análisis de seguridad y de otros ataques en la literatura buscando elementos comunes en los fallos de seguridad. Buscaremos formas de detectar estas vulnerabilidades de forma automática o semi-automática. De encontrarlas, éstas podrían ser el inicio de una metodología que permita comprobar si un nuevo CAPTCHA ofrece al menos un nivel mínimo de seguridad. Consideramos que una metodología que permita certificar un nivel de seguridad mínimo para los CAPTCHAs puede contribuir a diseños más robustos que ofrezcan mayor seguridad.

Contents

List of Figures	ix
------------------------	-----------

List of Tables	xiii
-----------------------	-------------

1 Introduction	1
1.1 Automatic abuse	1
1.2 CAPTCHA design	2
1.3 Motivation	3
1.4 Outline of contributions	4
1.5 Structure and contents	5
1.6 Publications	6
2 Background and related work	9
2.1 Introduction	9
2.1.1 Classical CAPTCHA formalisation	10
2.1.2 Criticism to the classical CAPTCHA formalisation	11
2.1.3 Alternative formalisation	13
2.2 Aspects of CAPTCHA design	14
2.2.1 Threat model	15
2.2.2 CAPTCHA design constraints	16
2.2.3 Applications of CAPTCHAs	19
2.3 Alternatives to CAPTCHAs	26
2.4 CAPTCHA design variants	28
2.4.1 Text images / OCR CAPTCHAs	28
2.4.2 Language/semantic based CAPTCHAs	35
2.4.3 Image based CAPTCHAs	36
2.4.4 Game-based CAPTCHAs	42
2.4.5 CAPTCHAs based on the understanding of video	43
2.4.6 Audio CAPTCHAs	43
2.4.7 Alternative problems for CAPTCHA designs	44
2.4.8 So-called “behavioural” CAPTCHAs	47

2.5	Attacks against CAPTCHAs	51
2.5.1	Attacks to text recognition (OCR) CAPTCHAs	51
2.5.2	Attacks to language/semantic CAPTCHAs	59
2.5.3	Attacks to image classification CAPTCHAs	60
2.5.4	Attacks to game-like CAPTCHAs	64
2.5.5	Attacks to audio CAPTCHAs	65
2.5.6	Attacks to “behavioural” CAPTCHAs	66
2.6	General attacks against CAPTCHAs	71
2.6.1	DL and game, audio and image-based CAPTCHAs	72
2.6.2	Oracle attacks	76
2.6.3	Relay attacks	76
2.7	New proposed CAPTCHA types	77
2.7.1	CAPTCHAs based on empathy	77
2.7.2	Enhanced image-classification CAPTCHAs	77
2.7.3	Puzzle CAPTCHAs	78
2.8	Summary	79
3	Case Study: Capy and other puzzle CAPTCHAs	81
3.1	Capy CAPTCHA description	82
3.2	Capy CAPTCHA analysis	83
3.3	Capy CAPTCHA design flaws	84
3.4	Foundations of the side-channel attack	85
3.5	Side-channel attack	87
3.6	Experimental results	88
3.6.1	Basic attack results	88
3.6.2	Modal attack results	89
3.6.3	Results analysis	90
3.7	Other CAPTCHAs affected	94
3.7.1	KeyCAPTCHA	94
3.7.2	Garb CAPTCHA	98
3.8	Possible improvements	99
3.8.1	Broader solution space	100
3.8.2	Challenge pre-filtering	102
3.8.3	Bigger image library	102
3.8.4	Client interaction analysis	103
3.8.5	Several puzzle pieces	103
3.9	Discussion	104

4	Case Study: The Civil Rights CAPTCHA	107
4.1	Civil Rights CAPTCHA description	108
4.2	Civil Rights CAPTCHA analysis	109
4.3	Civil Rights CAPTCHA design flaws	112
4.4	Foundations of the Machine Learning attack	115
4.4.1	Reading the answers	115
4.4.2	Classifying the challenge text empathic emotions . . .	117
4.5	Machine Learning attack to the Civil Rights CAPTCHA . . .	122
4.6	Experimental results	124
4.7	Possible improvements	128
4.8	Discussion	129
5	Case Study: FunCAPTCHA	131
5.1	FunCAPTCHA description	132
5.2	FunCAPTCHA analysis	134
5.2.1	FunCAPTCHA initial analysis	134
5.2.2	FunCAPTCHA image repository	136
5.2.3	FunCAPTCHA protocol analysis	136
5.3	FunCAPTCHA design flaws	138
5.3.1	ML analysis of the flaws and strength	139
5.3.2	Results of the ML analysis	140
5.3.3	Machine Learning attack parameters	143
5.4	Machine Learning attack to the FunCAPTCHA	146
5.5	Experimental results	149
5.6	Possible improvements	153
5.7	Discussion	154
6	BASECASS: Basic SEcurity CAPTCHA ASSESSment	157
6.1	Framework objective	158
6.2	Introduction to BASECASS	159
6.3	Detailed Description of BASECASS	170
6.4	Revisiting the CAPTCHA definition	173
6.5	Step 1.- Black-Box basic security analysis	174
6.5.1	Phase I: Automatic interaction	175
6.5.2	Phase II: Analysis of the challenge space	176
6.5.3	Phase III : Analysis of the answer space	179
6.5.4	Summary	181
6.6	Step 2.- Black-box S/ML analysis	182
6.6.1	Phase I: De-noising	183
6.6.2	Phase II: Pre-processing & transformations	184
6.6.3	Phase III: Metrics	186

6.6.4	Phase IV: Statistical and ML analysis	207
6.7	Step 3.- Parameter-based S/ML Analysis	212
6.8	BASECASS summary table	214
6.9	Examples of application of BASECASS	219
6.9.1	BASECASS analysis of puzzle CAPTCHAs	220
6.9.2	BASECASS analysis of the Civil Rights CAPTCHA	236
6.9.3	BASECASS analysis of FunCAPTCHA	246
6.9.4	BASECASS partial analysis of Math CAPTCHA	253
6.9.5	BASECASS partial analysis of HumanAuth CAPTCHA	256
6.9.6	BASECASS analysis of CaptchaStar	261
6.10	Summary of BASECASS	276
7	Conclusions and future work	277
7.1	Conclusions	277
7.2	Future work	279
A	Alternatives to CAPTCHAs	281
A.1	Threat prevention	281
A.1.1	Cost increase	282
A.1.2	Spam bombarding	282
A.1.3	Money blockade	283
A.2	Attack prevention	283
A.2.1	Alternate-channel validation	284
A.2.2	Third-party identification	284
A.3	Attack detection	290
A.3.1	Form honey-pots	290
A.3.2	Statistical and ML analysis of content	291
A.4	Attack mitigation	295
A.4.1	Blacklists	295
A.4.2	Client detection & filtering	297
B	BASECASS template	299
	Bibliography	303

List of Figures

2.1	Example of a HIP test from AltaVista	29
2.2	Examples from PessimPrint	30
2.3	Examples of Gimpy	31
2.4	Examples of BaffleText and reCAPTCHA	31
2.5	Example from Captchaservice.org	32
2.6	Example from Megaupload	32
2.7	Example image of the Teabag 3D CAPTCHA	33
2.8	Some examples from HelloCAPTCHA	34
2.9	RapidShare OCR/text with cats & dogs CAPTCHA.	34
2.10	Example from Egglue CAPTCHA	36
2.11	Example of a challenge from the first phase of theIMAGINA- TION CAPTCHA.	39
2.12	Some CAPTCHA examples from What's up CAPTCHA	40
2.13	Example of FunCAPTCHA orientation CAPTCHA	41
2.14	Example from the Facebook Social Authentication CAPTCHA	41
2.15	Examples from the PlayThru CAPTCHA.	43
2.16	Webcam-CAPTCHA design	45
2.17	Movement CAPTCHA	46
2.18	Physical CAPTCHA, or CAPPCHA	46
2.19	Example challenge from the proposed Amazon CAPTCHA . .	47
2.20	Example of Mori & Mali attack to the Gimpy CAPTCHA, algorithm A	52
2.21	Example of Mori & Mali attack to the Gimpy CAPTCHA, algorithm B	52
2.22	Example from the Microsoft CAPTCHA in 2008	54
2.23	Example of segmentation of a challenge from the Microsoft CAPTCHA in 2008	54
2.24	Example of restoration of a challenge from the Megaupload CAPTCHA	55
2.25	Example of segmentation of character components using Log- Gabor filters	56

2.26	Steps to break the Teabag 3D CAPTCHA	58
2.27	Steps to break Hello CAPTCHA	59
2.28	Verbs and success rates for the Egglue CAPTCHA	60
2.29	Example of 5x5-pixel textures used as features for the SVM	61
2.30	Example of edge detection for a first challenge of the IMAGINATION CAPTCHA	62
2.31	Success rate of the attack against the Facebook Social Authentication CAPTCHA	63
2.32	Background detection for a drag & drop game CAPTCHA	64
2.33	Target detection for a drag & drop game CAPTCHA	64
2.34	Wavefront recognition of digits in Google Audio CAPTCHA	65
2.35	Recognition of digits in Google Audio CAPTCHA	65
2.36	VAE-GAN learning high-level facial features	74
3.1	The two different challenge types offered by Capy	83
3.2	Plane-waves of each DCT coefficient.	86
3.3	Success rate by JPEG compression quality for 200-series experiments.	89
3.4	Computing time per JPEG compression quality for 200-series experiments.	90
3.5	Correctly solved challenges.	91
3.6	Wrongly solved challenges.	92
3.7	Success rate per image type and JPEG quality setting	94
3.8	Different versions of KeyCAPTCHA.	96
3.9	Wrong, partially and completely solved challenges for KeyCAPTCHA.	97
3.10	Several computed solutions for the Garb CAPTCHA.	99
3.11	JPEG size proportions at different distances from the correct solution.	101
4.1	Civil Rights CAPTCHA main web-page.	109
4.2	Example of challenges created with Securimage.	110
4.3	CRC answers created with Securimage	110
4.4	HTML body from the <i>CRC</i> API	111
4.5	Number of appearances of each of the 133 answers	113
4.6	Example metrics of some CRC answers.	116
4.7	Flow chart of the CRC basic attack.	124
5.1	Different FunCAPTCHA gender recognition iterations.	134
5.2	kNN performance degradation with smaller training sets	143
5.3	Flow chart of the attack to FunCAPTCHA.	148

5.4	Success rate by classifier and challenge type.	152
6.1	Generic flow chart for downloading the data needed for the Step 1 of BASECASS. This flow chart encompasses phase I. The data gathered will be analysed in phases II and III.	162
6.2	BASECASS generic flow chart.	172
6.3	Example of a challenge produced with Securimage.	174
6.4	Example mapping between subsets of H and P	176
6.5	Distribution of correct answers of the QRBGS CAPTCHA by challenge subtype.	181
6.6	Example of a Captcha2 challenge.	184
6.7	Example transformation into Log-Gabor components.	186
6.8	Steps to automatically solve a challenge from Captcha2	194
6.9	Example of a challenge produced by CaptchaStar.	262
6.10	Renders of the same CaptchaStar challenge for different (x, y) cursor positions.	263
6.11	Solutions accepted for a CaptchaStar challenge.	265
6.12	Distribution of correct answers for CaptchaStar and for an uniform distribution.	266
A.1	Logging-in with the possibility of using third-parties, or alter- natively registering using a CAPTCHA	287
A.2	Sequence of a third-party requesting access to a Twitter account.	288
A.3	Initial authorization to a third-party.	289
A.4	OpenID login example.	289

List of Tables

2.1	Some of the main attacks on well-known CAPTCHAs.	68
3.1	Success rate per image type and JPEG quality setting, data corresponding to Figure 3.7	93
4.1	Best classifiers for OCR of Securimage in the CRC.	118
4.2	Best Empathy classifiers, by algorithm and data.	120
4.3	Best parameter results in 10-CV, by algorithm and data. . . .	120
4.4	Best parameter results for the <i>CRC</i> questions, by algorithm and data.	121
4.5	Program answers for the basic attack.	126
4.6	Program answers for the improved attack.	127
4.7	% of successfully solved <i>CRC</i> challenges.	127
5.1	Some FunCAPTCHA wrongly and correctly classified faces, and their statistics.	141
5.2	Classification success rates for different <i>k</i> NN parameters . . .	142
5.3	Best and worst classifiers for off-line gender recognition with FunCAPTCHA.	145
5.4	FunCAPTCHA success rates by classifier.	150
6.1	Comparison of <i>H</i> and <i>P</i> for FunCAPTCHA.	178
6.2	Comparison of <i>H</i> and <i>P</i> for the CRC-OCR.	179
6.3	QRBGS challenge subtypes.	180
6.4	Index of Coincidence for some languages.	189
6.5	Base-case metrics depending on challenge media and type. . .	199
6.6	Some FunCAPTCHA faces and their histogram values.	200
6.7	Best classifiers for off-line gender recognition with FunCAPTCHA and OCR-recognition with CRC.	209
6.8	BASECASS summary table.	215
6.9	Summary table of the application of BASECASS to Capy. . .	224
6.10	BASECASS analysis for Garb CAPTCHA.	228

6.11	BASECASS analysis of KeyCAPTCHA.	233
6.12	CRC-OCR BASECASS Analysis.	239
6.13	CRC-Empathy BASECASS Analysis.	243
6.14	FunCAPTCHA BASECASS Analysis.	249
6.15	QRBGS challenge subtypes and space.	253
6.16	BASECASS Analysis for the QRBGS CAPTCHA.	255
6.17	BASECASS Analysis for the HumanAuth CAPTCHA.	258
6.18	Results of different ML algorithms on the simple CaptchaStar dataset.	270
6.19	Results of different ML algorithms on the detailed CaptchaStar dataset.	271
6.20	Attack results for CaptchaStar.	272
6.21	CaptchaStar BASECASS Analysis.	273
B.1	BASECASS template.	299

Chapter 1

Introduction

This chapter presents an overview, the goals and the motivation of the dissertation. It starts by introducing the problem of automatic abuse from an IT Security point of view. Next, it presents the problem being tackled in this dissertation, the design of CAPTCHAs. Afterwards, the motivation of this work is stated. Then, the main contributions of this dissertation are explained. Finally, the structure of this essay is described.

1.1 Automatic abuse

The Internet has spread to every realms of life. New generations spend more time on-line both socializing and working. People are getting used to the advantages of being constantly connected. Today not just computers are connected to the Internet: mobile phones, tablets, cars and many home appliances, as well as the smallest new devices, are also connected (IoT). This creates a huge playing field for crackers and tricksters to run their attacks. Using this ample base of both services and people, attackers have found ways to run exploits that provide an infinitesimal reward, but can generate substantial revenue by increasing the number of times they are run. The fundamental way of protection from these attacks has been to try to detect if at the other end of the communication there is a human person or a computer program.

There are many proposals for ways of remotely detecting humans.

Most of them fall into the category of asking the *human* to perform a task that is considered hard for computers (or *Artificial Intelligence (AI)-hard*), but not too demanding for humans. These tests are known as HIPs or CAPTCHAs.

1.2 CAPTCHA design

Since the first CAPTCHA used in Altavista in 1997, there have been numerous, very varied CAPTCHA designs proposed, implemented, and cracked. Even though it might look like an easy problem to the inexperienced, CAPTCHA design is not a straightforward problem to solve. Summarising, we can identify the following difficulties related to the design of CAPTCHAs:

- CAPTCHAs are typically used to protect resources that for the customer are not of a very high value (for instance, adding comments to a story in the news), or to which there are other alternatives for the customer (for services like web-mail). This competition means the CAPTCHA needs not to be felt as a burden by the user. This typically implies that it has to be easy enough, or playfully enough, otherwise it might affect the conversion rate of the services being protected.
- For the same reason as above, a CAPTCHA must not require a big commitment for its completion, even if the experience is very playful and positive for the user. Completing a CAPTCHA is never the reason, but a means to an end.
- CAPTCHAs should present alternatives for impaired users that offer the same level of security. This is not straightforward, as typically a CAPTCHA will use some human ability that is linked to a sense of perception (visual, auditory, etc.) thus not being valid for users with disabilities in that sense.
- The number of attacks per second against a CAPTCHA can be augmented automatically: it is just a matter of resources. Thus, a very small success rate can imply that for practical purposes, a CAPTCHA is broken. This is the case as soon as the Return of Investment (ROI) for the attacker is positive. Thus, in order to protect the most interesting resources, we need *AI-hard* challenges with extremely constant hardness throughout their domain.

- For some attackers, it might be profitable to hire low-wage human workers (what is typically called a *farm*) to solve a particular CAPTCHA challenge and then proceed to do whatever they wish. This would constitute a semi-automated attack. These *human CAPTCHA solving* services are offered today on the Internet and accessed through an API. It is good if a CAPTCHA has some way of preventing this from happening. Some CAPTCHA designers consider this requirement, yet the rest do not try to counter it (Athanasopoulos and Antonatos, 2006, Mohamed et al., 2014).

For those CAPTCHAs that are based on the original idea of using an *AI-hard* problem, there is the additional question of what really constitutes an AI-hard problem. An example was ASIRRA (Elson et al., 2007), an image-classification CAPTCHA based on a task that was thought to be hard for AI. It was broken months later using slightly different Machine Learning (ML) techniques (Golle, 2009).

We lack a consistent definition of what is *AI-hard*, nor a theoretical proof to show if a problem is *AI-hard* or not. This implies that we cannot know if such *AI-hard* problem would be in fact hard for a computer to solve *in all cases*, or there is a straightforward mechanism to evaluate in which cases it would.

Even if we find a genuinely *AI-hard* problem, how should we translate its hardness to the difficulty (for bots) of a CAPTCHA on which it is *based* on? By definition, the CAPTCHA will be automatically created and marked. This implies that its challenges will be a subset of the whole *AI-hard* problem. We also lack a method to know if this sub-set will keep the same AI-hardness as the original problem.

1.3 Motivation

There is an important body of research on the security of typical CAPTCHA schemes, which has found them to be insecure or too hard even for humans. Nowadays some new CAPTCHA proposals appear to which these known security analyses do not apply. We want to learn if these new proposals offer increased security, as claimed by their authors. To that extent, we have chosen the newest, original CAPTCHA schemes as case studies and analysed

their security. We expect that the analysis presented in this thesis contributes to the general knowledge of the design of CAPTCHAs.

To date, all proposals for CAPTCHAs that have been analysed have been found not secure, typically within a short span of a few months from their proposal time or from when they were put into production. This has happened for every type of proposal: commercial CAPTCHAs, academic proposals -both from researchers in ML and in Security-, alternative proposals from programmers or from amateurs. Many CAPTCHA start-ups had to close shortly after their CAPTCHA was found insecure. Many big companies have to constantly update their CAPTCHA in a race-like effort to make them resistant to the latest attacks. More worrying, security researchers that have successfully broken other CAPTCHAs and learned from those failures, have proposed their schemes just to see them also broken.

Most of the attacks found against CAPTCHAs can be considered to be side-channel attacks. These attacks do not try to solve the underlying problem on which the CAPTCHA designer has created her system, nor they try to advance the state-of-the-art in ML. Instead, they find weaknesses in the particular design of the CAPTCHA and ways to use them to gather enough information as to bypass the challenge a sufficient number of times. The frequency with which this type of attack is successful conveys the message that it is quite difficult to translate an *AI-hard* problem into a secure CAPTCHA.

There have been a few proposals for design guidelines for CAPTCHAs. They have been typically the result of a security analysis of one or more CAPTCHAs, and thus with limited scope and usability (Yan and Ahmad, 2007, Hindle et al., 2008, Bursztein et al., 2011, Nguyen, 2014). Nowadays, is not unusual that a new CAPTCHA design is put into production without performing a sound security assessment nor conducting external IT Security tests. These CAPTCHAs are implementations just based on an idea though to be hard enough by its designers. We want to know **whether there are some basic tests that we can run as to ascertain a basic level of security for a new CAPTCHA design, and that possibly can be automatic or semi-automatic**. In the long term, our goal is to increase the security of CAPTCHA designs.

1.4 Outline of contributions

The contributions of this thesis can be summarised as follows:

1. The main contribution of this thesis is to **test the security of new, original CAPTCHAs**, to which previous knowledge cannot be applied. To this extent, we have selected some case studies and analysed their security. Case studies are typical in IT Security and accepted as a way to contribute to the main corpus of knowledge in the field. We also perform our security analyses in novel ways, checking the challenge and answer domains, and using ML not to attack the base problem but to check for side-channel leaks of information.
2. The second contribution is a **meta-analysis of the results** of these previous security analyses. In this meta-analysis, we look for a common way to characterise the security problems found. This has the potential to show common patterns in failures in CAPTCHA design.
3. Building on the previous points, the third contribution is to propose a **framework to test for a basic level of CAPTCHA security**. This framework is based on the previous findings and can be applied to other CAPTCHA designs with minor modifications. It also goes beyond what other authors have proposed as CAPTCHA design guidelines. Testing for a basic level of security is important, as in Security a fundamental variable is the cost of an attack.

1.5 Structure and contents

This dissertation is divided into seven chapters. Chapter 1 presents the motivation and goals of this dissertation. Chapter 2 gives an overview of the state-of-the-art in CAPTCHA design. The following three chapters (3, 4 and 5) present the different case studies performed in new, original CAPTCHA proposals. Based on these results, Chapter 6 introduces BASECASS, a framework for Basic SEcurity CAPTCHA ASSEssment. Finally, Chapter 7 concludes the dissertation. Here we describe these contents in greater detail:

- Chapter 1 presents the motivation, contributions, and structure of this dissertation.
- Chapter 2 describes the different aspects that affect the design of CAPTCHAs. It also gives an overview of the state-of-the-art in CAPTCHA design and security analysis. It briefly mentions other alternatives to

CAPTCHAs, described into further detail in Annex A. Finally, it comments on the current trends in CAPTCHA design, presenting a brief analysis on them.

- Chapter 3 analyses the security of three puzzle CAPTCHAs. These are Capy, Garb and KeyCAPTCHA. These CAPTCHAs require the user to reconstruct the original image. This is a new type of image-based CAPTCHA.
- Chapter 4 analyses the security of the Civil Rights CAPTCHA, which is based on both empathy and OCR. The novelty of this scheme is that it uses empathy to increase the security of an OCR CAPTCHA. Empathy has not been analysed in ML before, although other writer emotions have.
- Chapter 5 presents the security analyses of the FunCAPTCHA gender recognition CAPTCHA. There are several proposals to use faces for CAPTCHAs, FunCAPTCHA being the first implementation of one.

Each Case Study ends with comments on how to possibly improve the designs and lessons learned.

- Chapter 6 introduces BASECASS. The ideas behind BASECASS are based on the results from these case studies and previous work. This framework is explained in detail, including summaries of its application in different cases.
- Chapter 7 presents the conclusions and comments on future research directions.

1.6 Publications

Some of the work presented in this dissertation has been previously published in the following articles:

1. Carlos Javier Hernandez-Castro, David F. Barrero, María D. R-Moreno. *A Machine Learning Attack Against the Civil Rights CAPTCHA*. In Proceedings of the 8th International Symposium on Intelligent Distributed Computing (IDC), 2014, Madrid, Spain.

2. Carlos Javier Hernandez-Castro, María D. R-Moreno, David F. Barrero. *Side-channel attack against the Copy HIP*. In Proceedings of the 2014 IEEE Fifth International Conference on Emerging Security Technologies (EST), 2014, Alcalá de Henares, Spain. *Best paper award*.
3. Carlos Javier Hernandez-Castro, María D. R-Moreno, David F. Barrero. *Using JPEG to Measure Image Continuity and Break Copy and Other Puzzle CAPTCHAs*. IEEE Internet Computing, Volume 19, Issue 6, Nov.-Dec. 2015.
4. Carlos Javier Hernandez-Castro, David F. Barrero, María D. R-Moreno. *Machine Learning and Empathy: The Civil Rights CAPTCHA*. Concurrency and Computation: Practice & Experience, Volume 28, Issue 4, March 2016.
5. Carlos Javier Hernandez-Castro, María D. R-Moreno, David F. Barrero, Stuart Gibson. *Using Machine Learning to identify common flaws in CAPTCHA design: FunCAPTCHA case analysis*. Computers & Security, Volume 70, September 2017.

Chapter 2

Background and related work

This chapter presents CAPTCHAs, including the different factors influencing their design, and commenting on the security of these designs. It starts by presenting and discussing the classical formalisation of CAPTCHAs that imposes some constraints on their design (section 2.1). After, it introduces the various aspects that influence the design of CAPTCHAs. In particular, section 2.2.1 defines their threat model, that is, the main threats that a CAPTCHA-protected service faces. It then discusses their primary use cases which also affect their design by the type of interaction (time, difficulty) that is considered appropriate for each use. CAPTCHAs are not the only security measure of protection for these scenarios. Some of these use cases accept the use of different alternatives. We will briefly present these alternatives in section 2.3 and discuss their benefits and drawbacks. Then, section 2.4 presents the different CAPTCHA designs, giving a brief historical introduction to the evolution of the major design paradigms. To better understand the forces driving the evolution of CAPTCHA design, section 2.5 comments some of the most relevant attacks to CAPTCHAs. This chapter finishes by presenting new proposed alternatives.

2.1 Introduction

IT Security has a history comprising several decades. During it, several prevention, protection and mitigation measures and mechanisms have been conceived. CAPTCHAs fall in a category of their own. No other security

mechanism has the task of remotely identifying the human species against an agent trying to mimic it.

Even though we use the name CAPTCHAs for these protection mechanisms, the name is misleading because CAPTCHAs, as they were defined by Naor (1996) and Ahn et al. (2003), are just a specific version of this protection mechanisms: as we will see in section 2.1.1, for a Human Interaction Proof (HIP) to be a CAPTCHA, it has to meet certain requirements, including being based on a *AI-hard* problem, using a public algorithm, etc.

Other mechanisms have been proposed, and more might be created, that do not follow these requirements, but still try to solve this security problem. For this reason, the more general but less used term HIP is better suited to describe these security mechanisms. As the term CAPTCHA is more widespread, we will use them indistinctly in this dissertation to refer to HIPs unless otherwise stated.

In the following sections we present the classical formalisation of CAPTCHAs as well as a discussion on it and a simpler alternative.

2.1.1 Classical CAPTCHA formalisation

Ahn et al. (2003) presented a somewhat restricted formalisation of HIPs that they defined as CAPTCHAs. This formalisation followed the seminal idea proposed by Naor (1996) that CAPTCHAs could be based on AI problems. In their formalisation, Ahn et al. (2003) link -by definition- the test to the AI problem it relies upon. Their definition can be summarised as follows:

Definition 1. A test V is (α, β) -human executable if at least a proportion of α humans can pass V with a success rate β or higher.

Definition 2. An AI problem is a triple $P = (S, D, f)$ where S is a set of problem instances, D is a probability distribution over S and $f : S \mapsto \{0, 1\}^*$ answers the problem instances. Let $\delta \in (0, 1]$. For a fraction $\gamma > 0$ of the humans H , it is required that $Pr_{x \leftarrow D}[H(x) = f(x)] > \gamma$.

Definition 3. An AI problem P is (ϕ, τ) –solved if there exists a program A running in time τ or less on any input from S such that:

$$Pr_{x \leftarrow -D, r}[A_r(x) = f(x)] \geq \phi \quad (2.1)$$

It is possible to prove that a particular program is able to solve a problem P in time τ or less on inputs from S . It is typically much harder to prove the opposite, that is, that for a problem P and any input from a broad set S , such a program does not exist.

Definition 4. An (α, β, μ) –CAPTCHA is a test V that is (α, β) –human executable and if there exists B that has success probability greater than μ over V to solve a (ϕ, τ) –hard AI problem P , then B is a (ϕ, τ) solution to P .

This definition links a CAPTCHA to the underlying *AI-hard* problem. It also links the strength of the CAPTCHA to the hardness of the *AI-hard* problem. This is done *by definition*, but in practical terms, there is no way to prove it.

Definition 5. An (α, β, μ) –CAPTCHA is secure if there exists no program B such that:

$$Pr_{x \leftarrow -D, r}[B_r(x) = f(x)] \geq \mu \quad (2.2)$$

for the underlying AI problem P . Note that in general, it is impossible to prove such a case.

2.1.2 Criticism to the classical CAPTCHA formalisation

CAPTCHAs as defined by Ahn et al. (2003) do pose an unnecessary constraint on what a HIP needs to be. They force CAPTCHAs to be related to an AI problem. The rationale for this requirement is that if a CAPTCHAs

has to detect something that is particularly human, that a machine cannot fake, then *it can be assumed* that it has to be something that even the most sophisticated programs -which we can think as of AI or ML algorithms- cannot fake. Some believe that the human characteristic is its ability to perform the most abstract or *elevated* types of thinking. To some, these types of endeavours have been the final aim of AI, so some conclude that such a human characteristically thing has also to be a challenge for AI.

In the late XX century, there was research in *AI* focused on this type of abstract, symbolic reasoning. This lead to the creation of symbolic languages like LISP or Prolog, the creation of Expert Systems and the evolution of formal theories of knowledge. Other human abilities that were considered related to our intelligence, like strategic board games such as Chess or Go, remained too hard for machines at that time. More so, other human abilities that were never related to our intelligence, like vision or audition, were considered easy at that time (Papert, 1966, Hankins, 2004). Ironically, for many decades these abilities remained among the most difficult to properly mimic by computers.

In 1997, advances in parallel programming and hardware allowed machies to beat Garry Kasparov, the Chess world champion. In the early XXI century, advances in ML and parallel hardware (GPGPUs), and the massive amounts of data created by the Internet, lead to machines beating Lee Sedol, the Go world champion, using DL (Deep Learning) and reinforcement learning. Nowadays DNNs (Deep Neural Networks) are able to modify paintings in the style of a painter and are starting to produce results at music composition or text writing in the style of an author. We do not know how long it will take machines to be as good as humans even at the tasks that today are considered highly intellectual. Thus, linking HIPs to *AI-hard* problems might not be a good idea.

Ahn et al. (2003) also define CAPTCHAs as being as strong as the hardness of the related AI problem. Note that they do so by definition. They do not offer a way to test if a CAPTCHA meets this definition, which in general is impossible. Thus, their formalisation is useless.

Defining a CAPTCHA to be as hard as the AI-problem it is based on does not offer a significant real-world value, as Ahn et al. (2003) do not provide a way to easily check that a CAPTCHA proposal actually meets their definition criteria. As we will see in our section about attacks (section 2.5), most CAPTCHAs have failed to attacks that did not improve the state-

of-the-art in ML. This is evidence of the extreme difficulty in translating a possible *AI-hardness* into the robustness of a particular CAPTCHA design. A slightly different formalisation, that allows to focus on the properties of the base problem and the CAPTCHA problem, as well as does not impose additional constraints, might be helpful to measure this *transfer* of robustness.

2.1.3 Alternative formalisation

Here, we present an alternative formalisation for HIPs/CAPTCHAs that does not impose any more constraints than the key ones, yet allows for a common way to refer to their different aspects. The aim of this formalisation is not to make claims of the strength of a HIP/CAPTCHA, but to present the essential elements of their design in a way in which we can later refer to them.

Definition 1. First let's define a generic problem, that will be the base from which a CAPTCHA test might be based. A problem P is a set of pairs $P = (pr, sol) \in E \times S$, being E the set of problem elements, and S the set of possible solutions.

Definition 2. A HIP/CAPTCHA H can be seen as a function f that returns a test and has up to two input parameters: a random seed, and optionally, a level of difficulty, $f(R, diff) \rightarrow t$. Only the first parameter is needed, as we can say that $f(R, diff) = f_{diff}(R)$.

Definition 3. We will say that H is based on P if and only if $\forall (R, c, corr_c) \in H, (c, corr_c) \in P$. This means that all valid examples of H will create a valid element $c \in E$ and return a valid validation function $corr_c \in VF_E$, plus the two will be linked in P . This is to say that every challenge and validation of solutions in H are correct examples and related solutions in P .

Note that this definition is not just theoretical but can be checked on a case-by-case basis. For example, in the case of the gender recognition challenges of FunCAPTCHA, it is equivalent to state that every female picture is regarded as a female by most humans and vice-versa.

Note that if H is based on P , this only implies that H can be seen as a subset of P , but does not imply that the strength (or difficulty) of H is

the same as that of P .

Definition 4. We will define the *human difficulty* of P as per equation 2.3:

$$P_{hs} = P(h(c) =_x corr_c, \forall (c, corr_c) \in P) \quad (2.3)$$

Where x is the *degree of similarity* that we will require to characterize two answers as identical (or almost, to the point that they are both correct). Similarly, we will define the *computer program difficulty* of P as $P_{cps} = P(cp(c) =_x corr_c, \forall (c, corr_c) \in P)$, where cp is a computer program that maximizes this function.

Definition 5. We will say that for an H based on a P , H retains the difficulty of P if and only if $H_{cps} = P_{cps}$. This is a theoretical definition and in general cannot be proved.

This definition of a HIP allows us to later further detail the important aspects of a CAPTCHA for our work in section 6.5.

Now that we have introduced what a HIP/CAPTCHA is, we will focus on the different aspects that influence its design. In particular, we will look into the different constraints affecting its design, both related to its security and to other aspects. We will also present an overview of different CAPTCHA designs, providing a brief historical background of the most widespread ones.

2.2 Aspects of CAPTCHA design

The design of any CAPTCHA is affected by its context and a set of practical constraints. This view is quite important to understand the difficulties and potential pitfalls in CAPTCHA design. First, we present the threat model as a way to introduce the different threats that both the service being protected and the CAPTCHA/HIP protecting it. Then, we give the main design constraints that mainly affect CAPTCHA design. Finally, we give an overview of the main current uses of CAPCHAs/HIPs.

2.2.1 Threat model

Here we present the potential threats for a service protected by a CAPTCHA. Theoretically, a CAPTCHA should be able to protect the service from most, if not all of them.

Threat 1. Automated abuse. Automated abuse happens when someone creates an algorithm that correctly solves the HIP/CAPTCHA “bypassing” or “cracking” it. Depending on the methods used by the attacker, we can distinguish two ways to “break” it: following the intended path of attack or side-channel attacks. The intended path is when the attacker creates an algorithm that solves the problem on which the HIP/CAPTCHA is based. A side-channel attack is the one that solves the CAPTCHA/HIP challenges, but not the underlying problem. Notice that in order to break a CAPTCHA, we do not need a high success rate in order for it to be effective, as the attack can be repeated and scaled up as long as there is a ROI.

Threat 2. HIP/CAPTCHA compromise. Different services can be protected by a central HIP server. This creates a single point of failure. If the HIP server is compromised, an attacker could gain automated access to all the services protected. This attack is more relevant regarding the major services providing CAPTCHA challenges.

Threat 3. DoS (Denial of Service) against the HIP/CAPTCHA server. Similarly to the previous threat, if a HIP service is disrupted either by internal issues or by a DoS attack, it is possible that authentic users will lose their ability to access the services protected by it. Because of this reason and the previous one, services protected by a HIP should have an alternative in case of HIP failure or compromise.

Threat 4. Compromise of communications. There is a potential risk if the communications between the client and the HIP/CAPTCHA server are compromised. This can happen through a MITM (Man-In-The-Middle) attack. This can allow an attacker to impersonate the HIP server and thus gain automated access to the service protected. Currently communications over the Internet can be secured by TLS (Transport Layer Security), but this protocol can also present vulnerabilities (Sheffer, 2015).

Threat 5. Semi-automatic abuse. If an attacker wants to bypass a HIP protection in order to access a service that offers substantial revenue, there is a low-cost alternative to finding an algorithm that bypasses the CAPTCHA. The alternative is to hire third-party “CAPTCHA solvers”, also known as *solving farms*. These are low-wage workers that solve the HIP challenges remotely. The service is provided through an API, so the rest of the access can be automated. Another option for an attacker is to syphon the CAPTCHA challenges to other human users that will solve them in order to get some service or revenue. An example of this are some malware and trojan horses (Cluley, 2007), phishing attacks (Kang and Xiang, 2010) or some Bitcoin *faucets*, which are a way to obtain cheap human labour solving CAPTCHAs.

Threat 6. Oracle attacks. HIPs typically do not produce their challenges completely at random but instead use some internal database that can include words, images, etc. depending on the CAPTCHA type. Typical CAPTCHAs allow the attacker to learn instantly if the challenges have been passed or not. Thus, it is possible to use most of them as oracles to learn if a proposed solution is or not correct, and thus launch learning attacks against them. Some work has been done to try to prevent oracle attacks for image-classification CAPTCHAs (Kwon and Cha, 2016), but this work has significant flaws (Hernández-Castro et al., 2017).

Threat 7. Service compromise. Ultimately if an attacker is able to gain access to the servers on which the service is provided or to find an alternative route to reach the service, then she will be able to bypass the HIP/CAPTCHA protection.

The design of CAPTCHA/HIPs is not just affected by this threat model, but also by a series of constraints related to the way they are administered, the human interaction and additional optional constraints. We present them in the following section.

2.2.2 CAPTCHA design constraints

CAPTCHA design constraints are of two different types: those that are fundamental and affect the design of any HIP, and those that are a much-wanted characteristic of HIPs but that can be considered optional by some

clients or in some scenarios.

The following constraints are fundamental to any HIP design:

- **CAPTCHAs have to be administered in a non-controlled environment:** a fundamental aspect of HIPs is that they are conducted remotely, at a site controlled by the client - not the HIP provider and through an unreliable network. This, for example, rules out the usage of biometric tests, as it is well known that biometric tests are only secure if administered in a controlled environment.
- **High usability:** in IT Security, it is well-known that typically a security measure will hinder the usability of the system being protected. This is a paramount concern in the case of CAPTCHAs: they are sometimes used to protect a resource that has a minimal value to the client (for instance, participate in an on-line poll) or to which the client has alternatives (for instance, web-mail account creation). Thus they have to be not too intrusive in the process they protect, or else they will affect the conversion rate. This is a severe restriction that affects both the difficulty of the HIP and the time to complete it. Designers have created some solutions to make HIPs more user-friendly, like drag & drop interfaces or in general the gamification of HIPs. Other CAPTCHA designers have decided to make them more appealing, for example marketing their HIPs as producing a benefit to humankind (for instance, helping OCR-scan old books). The user-friendliness of a CAPTCHA can be indirectly measured using the conversion rate metric.
- **CAPTCHAs need to offer alternatives for impaired people:** HIPs typically rely on one or more human perception abilities, like vision or auditory. There is a substantial number of people with difficulties in the use of a particular sense. Thus, ideally a HIP design has to present alternatives, so it is accessible for most or all of the population. The problem with this is that whenever an alternative HIP is present, the access is as strongly protected as the weaker of the HIPs used to protect it.

The constraints that are important, but can be considered optional are:

- **Privacy.:** there are many on-line services in which a certain level of privacy is necessary. This is more the case when opinions are encouraged.

An example can be a blog entry or news page that allows a comment section. If all comments can be traced back to the person that wrote them, we are restricting the sense of privacy and freedom, indeed curbing the willing of the public to express some of their opinions.

- **Reliance on a public algorithm:** the P in the acronym *CAPTCHA* means that the algorithm to create and grade the tests has to be public. In IT Security, there is a famous line of thought that states that security should never be based on secrecy, and thus the only secret should be the keys. The opposite, relying on the secrecy of algorithms and data further from the keys, is known as *Security by Obscurity* and has a long tradition of failure (Anderson, 2002, Hoepman and Jacobs, 2007, Swire, 2004). The history of IT Security related to Cryptography, Digital Watermarking, Steganography, etc. shows us that this type of foundation of security is not typically time-proof. This is usually regarded as a case of Kerckhoffs' principle, that can be stated as "(the system) should not require secrecy and it should not be a problem if it falls into enemy hands" (Kerckhoffs, 1883). Some standardization organisations as NIST advocate that "system security should not depend on the secrecy of the implementation or its components" (Scarfone et al., 2008). The problem with *Security through Obscurity* is that it would take an attacker a certain amount of effort to analyse it (or a leak), but once it's done it will allow her for ample abuse. This can be the case even if some of these measures are adaptive (through the use of ML). This is not the case with systems designed following *Security by Design* principles.
- **Reliance on a public dataset:** the same reason commented above strongly suggests that if the HIP uses some dataset, this should be public instead of private. As the dataset has to be exposed at least partially, even if modified/protected, it is difficult to protect HIPs from oracle attacks. Also if the dataset has a public source, it can be used to gain partial access to it or even poison it. Thus, the strength of a HIP should ideally not rely on its dataset being private. This is more relevant when the size of the dataset is small.

Next, to fully understand the context of CAPTCHA/HIP design, we will describe their main applications. This will allow the reader to have a complete overview of the ecosystem in which they work, and thus better understand the different designs and design criteria.

2.2.3 Applications of CAPTCHAs

In this section we introduce the most typical applications of CAPTCHAs that refer not only to the different kind of services that can be protected by CAPTCHAs, but also to other uses that CAPTCHAs might have by themselves. It is difficult to give a full list of applications, so here we focus on the most well-known cases.

2.2.3.1 DoS mitigation

A DoS (Denial-of-Service) attack intends to render a service unavailable. To do so, it tries to exhaust the service capabilities. There are various ways to do so, depending on the service. As an example, if the service is an on-line shop, one can try to perform costly operations like searches or modifications of the shopping cart. Typically this is done automatically and using several attacking machines, in what is called a Distributed Denial-of-Service attack (DDoS). There are several IT Security mechanisms to prevent them. One that we can implement at the application layer is a CAPTCHA: if a user is performing expensive operations, or many activities, and in general consuming a significant portion of resources, we can present a HIP/CAPTCHA to her to check that she is, in fact, a human.

2.2.3.2 Web scraping

Web scraping is the different techniques that allow for a third party to gain/copy information navigating a web-site automatically. An example could be a third party navigating the web-sites of several air travel companies and then offering the flights on their own web-site.

There exists a convention to disallow this to happen through the file *robots.txt* that, if present, can disallow any *bot* from navigating some of the subdirectories of the web-site. It is up to each particular *bot* to follow or not this convention.

Another way of protecting parts of a web-site from scraping is to present HIPs/CAPTCHAs to the users requiring information from those parts. An example of this is used by Google, that presents a regular OCR/text CAPTCHA when it receives some petitions from the same IP address, a

measure that sometimes leads to unexpected results (Cheng, 2016).

Another example of web-scraping is collecting e-mail addresses. CAPTCHAs/HIPs can be used to protect e-mail addresses from web-scraping by requiring the person wanting to access the e-mail address to solve a CAPTCHA before.

2.2.3.3 On-line polls

The first example of bot abuse on an on-line poll is the well-known case of the poll of slashdot asking which was the best graduate school in Computer Science, and that resulted in a voting competition of bots from Carnegie Mellon University (CMU) and Massachusetts Institute of Technology (MIT).

Today, there are on-line pools with very different purposes. The most typical is to gather the opinion, but there are others, like to select new products to produce, select best pictures, etc. Some of these polls have prizes, and others publish their result and can influence people. Being able to control the output of a survey can be very lucrative. The way to protect them with HIPs/CAPTCHAs is to request the user to solve a challenge before casting a vote. This alone does not prevent a user from casting multiple votes, but it can be combined with other measures, and will in the worse case allow for several votes per minute from the same user, instead of several thousands of votes per minute from a bot.

2.2.3.4 On-line sales and reselling

The best example of this is the case against *Wiseguys Tickets*, a ticket-scalping agency. They automatically purchased thousands of tickets from Ticketmaster and other vendors to resell them. They used a network of bots to bypass CAPTCHAs and grab more than 1 million tickets for concerts and sporting events, making over USD 23 million selling them. They were able to impersonate thousands of individual ticket buyers. They used credit card numbers and account holder names from ticket brokers. They also had a bank of about 1000 phone numbers, which their bot submitted as customer contact numbers (Zetter, 2010). Spite regulations and different countermeasures, these bots are still in use today (Hogan, 2016).

2.2.3.5 Preventing account gathering

Web-mail is the gateway to many other on-line services that require registration, including social media accounts. It can also be used to launch phishing campaigns and bypass some phishing filters thanks to the reputation of the web-mail service. Thus, the ability to create and use a large number of web-mail accounts is interesting for attackers.

Nowadays some web-mail providers advise their users to provide a back-up contact method that also helps these providers to rule out the possibility of an automatic creation of accounts. An example is letting/requiring the user to provide their phone number. This though is typically not a requisite, as web-mail providers do not want to narrow their possible market and lose clients to other more open providers. Also, there are on-line providers of temporary telephone numbers, so it would not be impossible to fake them.

Similarly to protecting web-mail registration, some sites do not require for much personal info nor e-mail accounts to register with them. Typically, after registering, the new user gains access to new information and services from the site. HIPs/CAPTCHAs can be used to protect the registration process and thus protect these sites from bots crawling into them, interacting with regular users, or using additional resource-intensive processes on the site (searching, basket manipulation, etc.)

Social networks are also an example of a service in which automatically gathering and managing accounts can imply an economic reward. Social networks are gaining widespread use and people are using them an increasing amount of time, becoming ubiquitous. They no longer serve just as social contact pages, but now are used to spread news, opinions, to contact with brands, to give feedback and reputation, to play on-line games, etc. Many trends, news and rumours spread *virally* at least partially through social networks. Attackers seek the ability to create thousands or millions of social network profiles and use them in different campaigns to influence or disrupt on-line discourse with spam hashtags, astroturfing, or fake users for persuading, smearing, or deceiving (Ferrara et al., 2014, Abokhodair et al., 2016, Echeverría and Zhou, 2017). Another use of social bots is to help the spread of disinformation disguised as real or *fake news*. In an study, it has been seen that “bots are particularly active in amplifying fake news in the very early spreading moments, before a claim goes viral”, “bots target influential users through replies and mentions” and “bots may disguise their geographic

locations. People are vulnerable to these kinds of manipulation, retweeting bots who post false news just as much as other humans” (Shao et al., 2017).

Social network bots are so common that there are tutorials on-line to create them, libraries in several programming languages, or even complete turn-key solutions (Bilton, 2014). Facebook reportedly has around 170 million fake users, or possibly more (Parsons, 2015).

There is much research that tries to flag, detect and nullify bots in social media. This is difficult as long as there is an economic incentive to it. A large-scale social bot infiltration on Facebook showed that over 20% of legitimate users accept friendship requests indiscriminately, and over 60% accept requests from accounts with at least one contact in common (Boshmaf et al., 2013). This further simplifies the infiltration in networks using social bots.

Social media networks have tried different methods to prevent bot accounts. Facebook studied incorporating a new CAPTCHA based on the identification of untagged faces of friends in pictures, but it was shown not to be resilient enough to current ML methods (Polakis et al., 2012).

A HIP/CAPTCHA that is more resistant than the current proposals can be used to protect social networks from bots. This can be done not only requiring the user to pass a challenge while registering, but also while performing certain actions on the site, or if she flagged by other algorithms as a user with a suspicious behaviour, pertains to a flagged network, etc.

2.2.3.6 Protection against dictionary attacks

When a user logs-in into a web page for which she has registered before, there are several authentication mechanisms. The most widely used requires the user to input her user-name (or e-mail address) and password. It is well-known that a significant fraction of the users choose passwords that offer little security, even if the systems try to prevent it. An attacker can automatically try thousands or millions of passwords for certain accounts, in what is called a *dictionary attack*, trying passwords that are combinations of words, numbers, sentences, etc., and typically gaining access to a substantial number of accounts. If these accounts contain links to real data from the users, they can be used to gather private information. They can also be used for altering reputation, phishing, spreading malware, etc.

2.2.3.7 Prevention of game cheating

Many on-line games allow for the creation of in-game economies. Players can get rewards either from work or from defeating other players. Players can exchange these rewards for enhancements that make the game easier and more enjoyable. Optionally, players can buy those rewards from the game publisher. This creates a market, either inside the game or outside in forums visited by the game users. In these markets, these rewards and prices are bought and sold for real currency.

It is known that some people in low-wage areas resort to playing games on-line as a mean to have some minimum income. An example of this is the “gold farming” performed in the on-line game Word-of-Warcraft, in which “virtual in-game currency and items are obtained by Chinese MMO players and sold for real-world currency to western gamers” (Hartley, 2009). Reportedly, this even happened forcefully (Tassi, 2011, Vincent, 2011). HIPs/CAPTCHAs cannot prevent this, but they can prevent attackers from using bots to perform these tasks.

Other more recent on-line games have also suffered from abuse from bots. One well-known case was Pokemon-Go, which had to install a CAPTCHA/HIP in-game in order to prevent bots from picking-up all pokemons that appeared on the map (Smith, 2016).

2.2.3.8 Prevention of fake feedback and reputation

Several sites and social networks allow their users to provide feedback for other users, sellers, services or products. Among the most well-known are the market site Ebay, the travelling site TripAdvisor, the social networks Google+ or Facebook, etc. In on-line selling, feedback and reputation are paramount. Being able to influence the reputation of a seller or service provider can significantly increase their sales. Similarly, providing bad feedback can severely affect sales - which can be used in blackmail or to hurt the competition. To do any of these, the attacker has to create some credible profiles in the sites and use them to provide bogus feedback (Fenton, 2015).

2.2.3.9 Prevention of comment spam

A well-known SEO technique consists on altering the content of high-ranking websites (news-sites, popular blogs, etc.). This is typically done adding comments to news, blog posts, etc. This type of comment is called *comment spam*. Because of the abuse of these, several techniques have been created in order to try to filter comment spam, including filters for the comment content and statistical learning to flag possible comment spam. These techniques have their limits (Ramilli and Prandini, 2009). Others have created their own techniques, like requiring to answer questions related to the post before submitting a comment (Lichterman, 2017), with users taking it as a fun quizz game (Schmidt, 2017), but this technique has limitations when used as a security measure (Hernandez-Castro and Ribagorda, 2009a). Some of these sites have started requiring user registration and/or linking to other web-mail or social media accounts. This strategy just passes the problem to another service provider. Another option for this is to require the user to pass a CAPTCHA/HIP challenge each time a user wants to post a comment.

2.2.3.10 Advertising

Apart than as a security mechanism, CAPTCHAs/HIPs can also be used as an advertising platform. Some CAPTCHA proposals have endorsed this idea with different targets. The Civil Rights CAPTCHA presents the users with news regarding Human Rights around the World as a way to increase awareness. Other proposals like *Captch Me*¹, *SolveMedia*² or *CAdCAPTCHA*³ require the user to interact with an advertisement (that can look as a game) or answer a question relative to it.

This should not be the main intent of a CAPTCHA/HIP, because focusing on the advertising can affect the security of the proposed CAPTCHA. It offers a way for web-site owners to monetize their content, as they can explain that they are using a CAPTCHA/HIP for protection - not just for advertising.

¹Located at <http://www.captchme.com/en/>, retrieved in March 2017.

²Located at <http://solvemedia.com/advertisers/>, retrieved in March 2017.

³Located at <http://captchaad.com/solutions/>, retrieved in March 2017.

2.2.3.11 Collaborative work

Some CAPTCHA proposals have used to some extent the idea of using the challenges to solve a real problem. Thus, leveraging the human CAPTCHA solvers, the CAPTCHA provider is also able to offer a solution to some related problem.

The original *reCAPTCHA* is an example of using this collaborative idea in a CAPTCHA. It had the double intent of being a security mechanism and helping digitise parts of books and papers that were hard for machines (Von Ahn et al., 2008). *reCAPTCHA* presented two words, being one artificially distorted (a word that the machine knew) and another one a word the OCR could not read (so that the machine initially did not know). If the user replied well to the first one, it was considered human. Her answer to the second was then recorded. After enough similar answers, the OCR word would be considered as read. At this point, the OCR word could be used to tell humans from computers. This model gained widespread usage.

Notice that this model is not especially secure. True, if OCRs fail on a word, it is a good candidate to discriminate humans and programs. But OCRs typically offer suggestions of what the word might be, and even if different OCRs do not agree, one of them might be providing the correct answer, or an answer that is close enough and that can be corrected with a dictionary. Thus, even if an ensemble of OCRs cannot determine the word, that does not mean that a machine cannot guess the word a number of times good enough for an automated attack. The attacker does not need to be correct 100% of the time.

reCAPTCHA was successfully attacked and suffered many evolutions. It evolved by departing more and more from its original model in order to try to become more robust against previous attacks.

Summary As already explained, CAPTCHAs/HIPs are not the only security mechanism that can help distinguish bots from humans. In some user cases, there are other possibilities. It is important to be aware of them to understand the benefits that CAPTCHAs offer against the rest of alternatives.

In the following section 2.3, we give an overview of these alternatives. This overview will also allow the reader to understand the broader context in which CAPTCHAs/HIPs operate.

2.3 Alternatives to CAPTCHAs

We have presented the most typical use cases for CAPTCHAs/HIPs. While doing so, we mentioned that there are other security mechanisms to protect the services in many of these use cases. Here, we give an overview of the most prominent mechanisms alternatives to CAPTCHAs/HIPs for some of these cases and discuss their limitations. In Appendix A we provide a more detailed introduction to them as well as a more comprehensive discussion of their limitations. A full list of these other security mechanisms is out of the scope of this dissertation.

The different alternatives to CAPTCHAs can typically be applied to a subset of the problems that CAPTCHAs try to prevent. They also work in different parts of the threat model: threat prevention, attack prevention, attack detection and countermeasures.

One alternative to protect forms or other interaction services (as on-line polls) is to add *form honeypots*: form fields that are not visually present to a human, thus will not be filled by a human. This is not difficult to spot for a not-too-basic form filler. Also, to protect comments from *comment spam*, we can use *Statistical Analysis* tools in order to classify comment spam, as well as it is done with e-mail spam. This is prone to *evasion attacks* in which the attackers find ways for some text not to be analysed, or add “good” words in order to influence the analysis. Some central services offer this kind of analysis. Among the most well-known ones is Akismet, given its user base in WordPress blogs. There are critics that complain about its rate of false positives⁴ and research able to bypass it (Ramilli and Prandini, 2009).

Another option to the use of CAPTCHAs/HIPs for user registration is to require *alternate-channel validation*, for instance, requiring a phone number to which a confirmation code is sent, or an additional e-mail address. The main drawbacks with this approach are both the lack of anonymity and the price, as for example, Ringcaptcha, that calls users to check if they are humans, charges US\$49 per month if you are calling US numbers and more for overseas.

Some service providers might prefer to leave the identification to

⁴Criticism with complaints like “Akismet has a reputation for flagging good comments as spam” can be found in blogs and forums. This one, in particular, is from “Why We Don’t Use Akismet” post at <http://www.web-development-blog.com/archives/why-we-dont-use-akismet/>

a third party that they consider technologically strong enough as to filter bot users. This can be done using third-party authentication protocols like *OAuth/2* and *OpenID Connect*. This poses important drawbacks concerning user privacy, as the identification sites can follow a user's steps through the internet. Furthermore, there is a huge number of applications and services that are clients of OAuth, and this prevents from properly testing them. As an example, just for Twitter, a new app is registered every 1,5 seconds. These also represent a single point of failure, as if the authentication mechanisms or servers are compromised, an attacker will be able to impersonate another user or users in several services.

Another option to HIPs/CAPTCHAs is the use of whitelists/blacklists. Blacklists are lists of well-known attackers. Their identification can be done through different possible mechanisms, as using their IP address, characteristics of the request strings, techniques for client & browser fingerprinting, the new HTML5 APIs, etc. Their detection is typically done through abuse detection. A well-known example of this is used by Cloudflare, that aims to protect, speed up, and improve availability for websites and mobile applications. They do this by imposing an intermediate server layer thanks to changes in the DNS entries. These mechanisms have their own drawbacks. For example, if a node in a private network that is behind a proxy is abusing a site, all nodes in that network will lose access it. Services that run these blacklists and filtering mechanisms also provide a *single point of failure*.

To be able to create a black list, it is necessary to first distinguish among different clients and detect who is running an attack. Several techniques can be used for it, as client & browser fingerprinting, source IP detection, cookies and many others, even more so with the new HTML5 APIs, but each one has its limits. Because most of them are created at the client side, with enough motivation or dedication they can be faked.

A somehow related idea is *client detection & filtering*: classify without doubt those clients that are clearly bogus, or attackers. The most common idea behind this mechanism is that many attackers do not use a regular browser, but some other SW that does not replicate the full functionality of a browser. As an example, many attackers do not run the Java Script code of a web-page, run it partially, or do not have full JS & DOM support. This is just an arms-race.

2.4 CAPTCHA design variants

In this section, we introduce the different CAPTCHA design variants and show how they have coped with the different design constraints. It also gives a short historical introduction for the most popular variants, like OCR CAPTCHAs (subsection 2.4.1). This section finishes presenting CAPTCHAs that are based on alternative, non AI-based base problems (subsection 2.4.7), and the so-called “behavioural” CAPTCHAs (subsection 2.4.8), that constitute the main trend today, thanks to this paradigm being used by the main CAPTCHA provider.

Through the beginning and evolution of CAPTCHAs/HIPs, there have been and currently are many different proposals. Section 2.5 shows the attacks to many of the design variants presented here. Initially, we can classify CAPTCHAs as either based on some problem perceived as *AI-hard* or based on some alternative problem that is not related to AI.

First we will present the CAPTCHAs based on the idea of an *AI-hard* problem, following the initial idea of Naor (1996) and later Ahn et al. (2003). These are by far the most popular. We can further divide them into different design categories both based on their transport media (text, text images, audio, images, video ...) and in the particular problem they are based on (OCR, classification, understanding, ...).

2.4.1 Text images / OCR CAPTCHAs

Text-based CAPTCHAs pertain to two main categories: those based on the problem of text recognition from an image (OCR), and those using text as a means to ask a question. Next, we will explain both in detail.

2.4.1.1 Text OCR CAPTCHAs

The first category has been the most popular CAPTCHA class from 2000 to around 2014, when image-based CAPTCHAs started being increasingly popular, yet OCR CAPTCHAs are still popular.

One of the earliest examples of abusing on-line services started in 1997 when some people started using automatically the “add-URL” service



Figure 2.1: Example of a HIP test from AltaVista (Baird, 2006).

provided by Alta-Vista, the most popular Internet search engine at the time, for Search Engine Optimization (SEO) purposes. They were automatically submitting large numbers of URLs in an effort to manipulate the importance ranking algorithms of AltaVista.

Andrei Broder and his colleagues at DEC Systems Research Center were collaborating with Alta Vista at that time. Possibly following on the ideas of Naor (1996), his team developed an algorithm that randomly generated an image of printed text with some distortions so that OCR programs could not read it, requesting the *human* user to input such text (see figure 2.1). The distortions included random typefaces, rotation and scaling, as well as the optional addition of background noise. Characters were chosen at random, not from a dictionary. It is important to note that by that time or shortly after, there were known algorithms able to recognise patterns even after being rotated and scaled (Shen et al., 1999, Leung et al., 1998). In January 2002, Broder stated that the system had been in use for “over a year” and had reduced the number of “spam add-URL” by “over 95%”, even though there was no additional information on the remaining 5% (Baird, 2006). Thus this security measure reached *some* level of efficacy. A U.S. patent was issued in April 2001 with these ideas (Lillibridge et al., 2001).

Udi Manber of Yahoo! encountered a similar problem when *bots* started joining on-line chat rooms and pointing the users to advertising sites. He described this “chat room problem” to researchers at CMU.

Professors Blum, Von Ahn and Langford articulated desirable properties for any such test to remotely tell humans and computers apart (Baird, 2006):

- The challenges should be automatically generated and then graded by a computer using a public algorithm.
- The challenges should be easy and fast to complete for *virtually all* humans, independently of abilities, cultural background, etc.
- The test should be able to reject *virtually all* machines.



Figure 2.2: Examples from PessimalPrint (Baird et al., 2003).

- The test will be able to resist automatic attacks for *many years*, even as technology advances and even if the test’s algorithms are known.

Thus they coined the term CAPTCHA, for *Completely Automated Public Turing test to tell Computers and Humans Apart*. Note that since the creation of these broad guidelines, some have been dropped by practitioners and new ones have been proposed. For example, nowadays it is considered that the limit for a production HIP for the ratio of computer-solved challenges should not exceed 0,01% (Chellapilla et al., 2005*b*) or 0,6% (Zhu et al., 2010*a*). Also, many practitioners are not making public the grading algorithm internals of their CAPTCHA proposals (Hernandez-Castro et al., 2011, Shet, 2014*a*, NuCaptcha, 2016, Inc., 2016). Additional requirements have also appeared, most notably the protection against third-party human solvers.

Henry S. Baird, an expert on computer vision and document image analysis at Xerox PARC, organised the first International Workshop on HIPs in January 2002. He was also part of the team that created PessimalPrint, an OCR/text CAPTCHA that uses ten typical image degradations, including spatial sampling rate and error, affine spatial deformations, jitter, speckle, blurring, thresholding, and typeface size (see Figure 2.2). They published their proposal (Baird et al., 2003) in which was the first peer-reviewed proposal for a CAPTCHA.

The professors at CMU worked on their own proposal, that they called Gimpy. It rendered random words as images of printed text, applying to them some shape deformations and image occlusions. Particularly interesting was that the word images often overlapped (shown in Figure 2.3). The user had to write down three of the 10 words shown to pass the test. An in-depth recount of this early historical phase can be found at Baird (2006) and also at “Human or Computer? Take This Test”⁵.

⁵Located at <http://www.nytimes.com/2002/12/10/science/human-or-computer-take-this-test.html>, retrieved on November 2016.

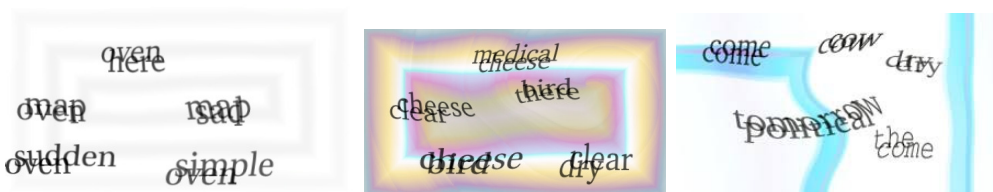


Figure 2.3: Examples of Gimpy challenges (Mori and Malik, 2003).



Figure 2.4: Examples of (a) BaffleText and (b) reCAPTCHA challenges (Chew and Baird, 2003). It can be seen that the first word from the reCAPTCHA challenges uses the ideas presented by Chew.

Yahoo! decided to implement a much watered-down version of Gimpy that used just one word. At the early 2000s the Internet was still spreading fast as new users and services were being added. Text-based CAPTCHAs became extremely popular. They were easy to understand and implement and apparently secure enough, so it caught attention and spread quite rapidly. With as few as 5 characters (case-insensitive letters and digits) there are $36^5 \approx 60$ million possible answer combinations.

By this time, the first CAPTCHA breaks were being published. Regarding the design of OCR CAPTCHAs, the most important consequence of these attacks is that they identified segmentation as the most challenging tasks for the attacking algorithms. This realisation affected the following OCR CAPTCHAs, as they tried to make stronger use of known and new anti-segmentation techniques.

Through the 2000s, active research was done in OCR/text CAPTCHAs. They were by far the most deployed CAPTCHA type (Hernandez-Castro and Ribagorda, 2009a). It would be impossible to discuss all the different variations and alternatives that appeared. We will mention some of the most notorious ones to give the reader a general overview of some of the typical designs and problems with OCR/text CAPTCHAs.

In 2003, Monica Chew and Henry S. Baird proposed BaffleText, a novel OCR/text CAPTCHA that used non-English but “pronounceable” words along with some masking techniques motivated by the Gestalt psychology (Chew and Baird, 2003). The ideas presented in this work were later put to use by reCAPTCHA starting in 2010 and at least until 2014 (Figure 2.4 shows an example).

In 2005, Chellapilla et al. (2005a) were able to attack several OCR/text proposals already deployed. They concluded that new OCR/text CAPTCHAs schemes should be based on hard-segmentation problems (Chellapilla and Simard, 2005, Chellapilla et al., 2005b).

In 2007, Captchaservice.org appeared. It described itself as “the first web service designed for the sole purpose of generating CAPTCHA challenges” (Converse, 2005, Yan and Ahmad, 2007). Their different OCR challenges were marketed as successfully tested against OCR SW (an example challenge is shown in Figure 2.5).



Figure 2.5: Example from Captchaservice.org⁶.

Even though by then most OCR CAPTCHAs had been defeated one or several times, the main trend for companies was to update them trying to evade the current attacks. This was typically done by making segmentation harder.



Figure 2.6: Example of a challenge from the Megaupload CAPTCHA.

In 2010, a popular place for file sharing (Megaupload.com) designed a CAPTCHA that showed strong anti-segmentation techniques. This CAPTCHA prevented segmentation by overlapping large portions of characters amongst themselves.

⁶Example taken from the copy of their webpage on 07/15/2006, located at <https://web.archive.org/web/20060715020359/http://captchaservice.org/>.

2.4.1.2 3D OCR/text CAPTCHAs

Some of the OCR/text CAPTCHAs proposals were created using very different ideas that set them apart from the typical OCR/text CAPTCHAs. One of these proposals is the Teabag 3D CAPTCHA, an example of a 3D text-based CAPTCHA (see Figure 2.7). It is not the only example, but possibly the best implementation of the idea, done by a very knowledgeable group of CAPTCHA security analysts and programmers called *OCR Research Team* (Kolupaev and Ogijenko, 2013).

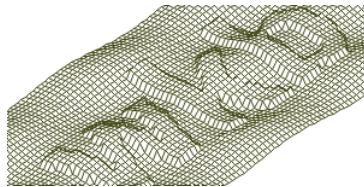


Figure 2.7: Example image of the Teabag 3D CAPTCHA v1.0.1 (Kolupaev and Ogijenko, 2013).

There are other proposals, including a 3D moving CAPTCHA (Kund, 2011) as well as a 3D CAPTCHA that uses the human ability of stereoscopic vision (Susilo et al., 2010), but as they have not been implemented publicly, so their security remains to be checked.

2.4.1.3 Animation of OCR/text CAPTCHAs

Another variation for OCR/text CAPTCHAs comes from adding a time component to them through animation. The idea behind this is to distribute the information required to solve the CAPTCHA so that no single frame contains all the information needed, thus *in principle* rendering useless previous attacks to typical OCR/text CAPTCHAs. This idea was called the “zero knowledge per frame” principle (Cui et al., 2010).

There have been several proposals based on this idea. One of the first proposals presented the text rendered on an animated surface (Fischer and Herfet, 2006). Naumann et al. (2009) proposed another animated OCR/text CAPTCHA based on entities that move together over a noisy background, so they *become visible* based on their movement. A similar proposal was presented by Cui et al. (2010).

HelloCAPTCHA (Group, 2016) is another example of this idea, even

though it does not strictly follow the “zero knowledge per frame” principle (shown in Figure 2.8). NuCaptcha (NuCaptcha, 2016) is a commercial proposal that implemented animation in its OCR/text CAPTCHA as well as anti-segmentation by overlapping.



Figure 2.8: Some examples from HelloCAPTCHA, showing two frames for each. Some characters change position and orientation, and in some challenges, not all the characters are visible at the same time. This measures try to prevent a typical OCR attack over a single frame.

2.4.1.4 Alternative OCR/text CAPTCHAs Ideas

A popular file-exchange service called *Rapidshare* developed several OCR CAPTCHAs that were apparently broken. Trying to raise their security, they developed a CAPTCHA that mixed OCR and image classification. To do so, they showed distorted images of cats and dogs next to characters. The user was asked to write down only those characters next to a cat (see Figure 2.9). This CAPTCHAs gained a lot of criticism, as it was considered too difficult even for humans. After just a few months, CAPTCHA breaking tools bypassed it (Martin, 2008), so it was finally retired from use.



Figure 2.9: RapidShare OCR/text with cats & dogs CAPTCHA.

The Quantum Random Bit Generator Service⁷ (QRBGs) is a free service providing “truly random” bits on demand, hosted by the Ruder Boskovic Institute of Zagreb. As their creation bandwidth is limited, they require registration in order to access it. In order to prevent automatic

⁷Located at random.irb.hr, retrieved March 2017.

registration, they developed a CAPTCHA that required the user to provide the numerical result of a mathematical expression. The idea is that this expression is rendered in low dpi, so OCR programs have trouble detecting its different parts and reading them. Also, correctly relation the different elements in the expression is not a straightforward problem.

Another such OCR/text CAPTCHA was the commercial proposal Captcha2, in which the user had to click on the correct character, distorted and rotated, and protected with clutter. Nguyen et al. (Nguyen et al., 2014a) present another proposal that relates characters to their locations. They have not implemented it publicly so their security remains unknown.

2.4.2 Language/semantic based CAPTCHAs

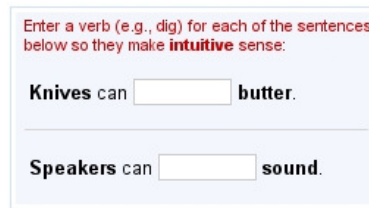
There are other proposals that use text to present a message, but their aim is not for the user to read correctly the text but to understand it, and do some action related to it. Among these, there are many simple CAPTCHAs that ask users very simple questions, as to detect which word is different from a list of words, or solve a very simple arithmetic problem.

In general, the idea of semantic CAPTCHAs is to use the human abilities of language processing and semantic analysis. An example of this is TextCAPTCHA, that is able to create thousands of textual questions from different categories⁸. Another example is Egglue Semantic CAPTCHA, that uses a web service to create “over 10,000 knowledge-based, accessible CAPTCHA challenges” in which the user has to choose the verbs that make sense to complete two sentences⁹ (Figure 2.10).

A different idea consists on asking humans to create labelled data to later use in your CAPTCHA, a bit similar to the idea behind the ESP game (Von Ahn and Dabbish, 2004), which is described in the next section. Another similar idea is to use one particular aspect of “humanity”, as could be humour, or interpreting emotions from paintings, and trying to detect humans from computers depending on the preferences on it, as depicted by Chew and Tygar (2005). This is a possibly interesting idea, that also has potential to be further developed. This proposal has never been implemented,

⁸Available at <http://textcaptcha.com>, retrieved on November 2016.

⁹Located at https://www.drupal.org/project/egglue_captcha, retrieved on November 2016.



Enter a verb (e.g., dig) for each of the sentences below so they make **intuitive** sense:

Knives can butter.

Speakers can sound.

Figure 2.10: Example of a challenge from Egglue CAPTCHA. Here, it asks the user to complete the sentences “Knives can ***** butter” and “Speakers can ***** sound” using a verb in each one.

though, so we are not sure about its potential capabilities and limits.

2.4.3 Image based CAPTCHAs

The decade of the 2000s marked the start on research on OCR/text CAPTCHAs and their security. After several successful OCR CAPTCHA proposals had been fundamentally broken, some people started to believe that OCR/text CAPTCHAs were fundamentally limited. Thus, some people looked for different alternatives that could lead to stronger CAPTCHAs that still had high usability.

Many of these researchers focused on the more general problem of Computer Vision (CV). This was a natural election, given that CV was an AI field that had several unsolved problems at that time. The human vision system is good at recognising objects in pictures, and there is a great variety of possible objects to recognise - orders of magnitude bigger than different characters. Additionally, these objects can have almost unlimited different versions.

Next, we will present the different CAPTCHAs based on image problems considered *AI-hard*. First we will introduce those based on image classification, that is, the ones that classify an image into a single class that describes the main content of the image (main object or scene it is representing). These are based on the long-studied CV problems of image classification and object recognition. Then, we will present other CAPTCHAs that also use some image-related CV problem as their base. Finally, we will discuss those CAPTCHAs that are based on face identification, recognition, or extracting information from faces.

2.4.3.1 Image classification CAPTCHAs

The first CV problem to be used as a base for a HIP was image classification, that is, presenting images that depict a single object (or a very clearly defined primary object) and asking the user to select its class.

Chew and Tygar were the first to use labelled images to produce CAPTCHA challenges. They used the label associated with images according to Google Image Search, which then, was extracted by the image title and alternative text, as well as the surrounding text. This technique is not very well suited for CAPTCHAs, as sometimes the text was not a good representation of the image contents. For an example of this, using the query “river” might refer, among others, to a flow of water, or to the Club Atlético River Plate, an Argentinian soccer club.

With the intention to associate interesting labels to images, Von Ahn and Dabbish created the “ESP game” to feed the PIX CAPTCHA database (Von Ahn and Dabbish, 2004). It encouraged the players to assign *easy* labels to the different images, as straightforward labels would lead to easier agreements with unknown partners.

In 2006 appeared the first production CAPTCHA that used a large, labelled database of images. It was *HotCaptcha.com* (Marshall and Lin, 2006). It used the human-labelled database of *HotOrNot.com* and its public API. It was taken down in less than two years, possibly given problems with the providers of the images (the *HotOrNot.com* web-site).

Oli Warner had a similar idea, in this case using photos of kittens (Warner, 2009). Its biggest problem is that its database of pictures was small (< 100). Similarly, the HumanAuth CAPTCHA requested to distinguish between images depicting either a natural object (a tree) or an artificial one (a watch). It was an Open Source project that was shipped with 69 pictures, very lightly obfuscated using a watermark.

In 2007, Jeremy Elson et al. from Microsoft presented the ASIRRA CAPTCHA (Animal Species Image Recognition for Restricting Access) (Elson et al., 2007). It asked users to distinguish images depicting cats from images depicting dogs, which according to some CV specialists asked by the authors, was a very difficult CV problem. Their main contribution is that, as did *HotCaptcha.com*, ASIRRA used a huge human-labelled database of pictures, in this case from the website *PetFinder.com*, who had an extensive database

“of more than 3 million photos”, growing in thousands each day. In exchange for access to it, ASIRRA displays an “adopt me” link next to each picture, that links to the PetFinder web-page about the pet - note that this was a potential security risk, even if it was not used by any attack.

ASIRRA authors performed a previous security analysis to try to assess the level to which current ML was able to break their CAPTCHA, and concluded that it was not possible. Better, they provided a training set for anyone willing to try their ML algorithms on it. This deserves a high praise to the ASIRRA authors, that were considering the security of their CAPTCHA seriously.

There have been other CAPTCHA proposals based on image classification, though they have lost interest with the advance of CV techniques, particularly using DNNs. In section 2.6.1, we discuss the implications of DL for CAPTCHAs in greater detail. As a summary, today it is quite difficult to affirm that an image classification/object recognition problem is hard enough for ML, given that enough computation power and data is available.

2.4.3.2 Alternative image-based ideas for CAPTCHAs

Apart from image classification and object recognition, other CV problems and alternative ideas were also used as a base problems for CAPTCHA designs. In some cases, the designers of new CAPTCHAs completely avoided the need for a big, growing database of human-labelled images. To do so, they altered dynamically a library of images creating many times more derived ones.

One example of dynamic image creation is the IMAGINATION CAPTCHA (Datta et al., 2005). Their CAPTCHA proposal has two phases. The second is a typical image labelling CAPTCHA. The first, most interestingly, asks the user to click on or around the centre of any of the images that form a mosaic. This mosaic of images is created in a way that makes it difficult for known CV techniques to segment it (see Figure 2.11).

Another example of an image CAPTCHA no based on image classification was the proposal by Gossweiler et al. (2009) from Google Research. Their CAPTCHA presents rotated images to the user. The user has to rotate them back to their original orientation, or alternatively, to select the image that is vertical from a set of images (shown in Figure 2.12). They use a database of images that they filter, trying to remove images that offer clues

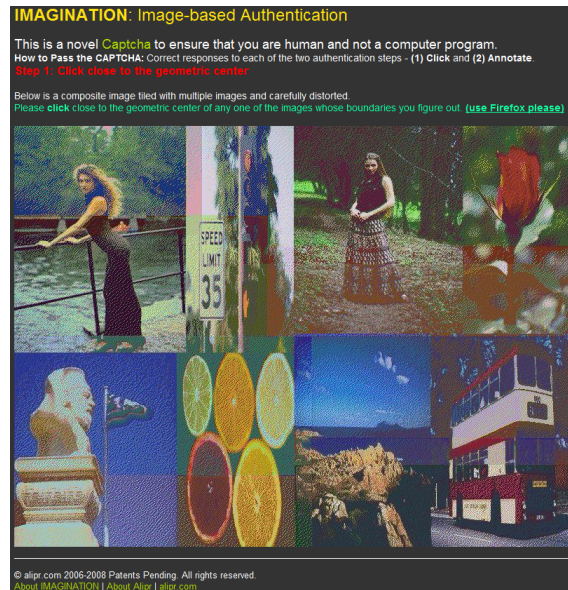


Figure 2.11: Example of a challenge from the first phase of theIMAGINATION CAPTCHA.

so that verticality is easy to detect for a program: images that contain faces, skies, grass, sand or other objects easy to locate for CV algorithms, or easy to detect lightning conditions. They also filter out images that might be difficult for humans. To do both, they use set of pre-trained classifiers. They also filter out images that appear to be difficult for humans to orientate, depending on the on-line results obtained.

Other similar proposals have appeared that do not seem to add significant novelties (Kim et al., 2010, Mehrnejad et al., 2011, Gross, 2015). Sketcha is a CAPTCHA proposal based on line drawings of 3D Models (Ross et al., 2010) that have to be rotated to their original position. FunCAPTCHA has put into production a variant of this idea in which they render several intertwined 3D models for what they call their “high-security mode” (see Figure 2.13) (Gosschalk and Ford, 2016).

Cortcha (Zhu et al., 2010a) also pertains to this category, although it is based on a more advanced idea. It uses a database of images to build a database of objects, using the JSEG method from Deng and Manjunath (2001) to segment images into objects. Small objects are merged with their neighbours. Then, they assign each object a “perceptual significance” value using an algorithm from Liu et al. (2011). They apply heuristics to classify objects as useful or not. These heuristics measure how easy it is to recognise

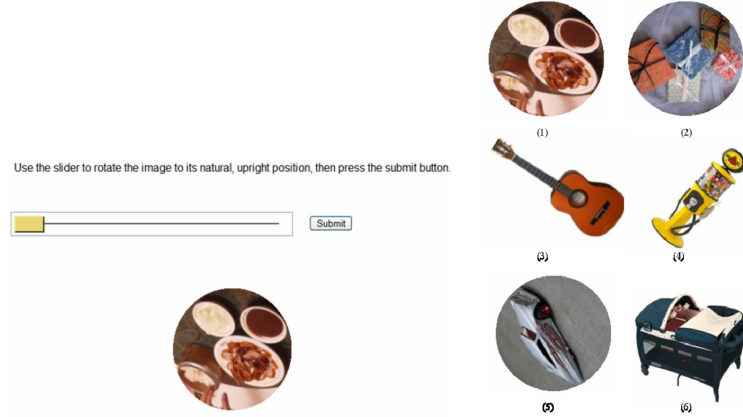


Figure 2.12: Some CAPTCHA examples from What’s up CAPTCHA (Gossweiler et al., 2009).

the objects if they appear cropped out of the image, and how “meaningful” they are. To generate a challenge, an image from the database and an object from that image are randomly picked-up. The object and a buffer region around it are cropped from the image, and in-painted using a variant of the algorithm proposed by Sun et al. (2005). Then, another n similar objects are selected from the database. The user has to select the correct object and drag & drop it to its correct location in the image.

Another of such interesting proposals is based on video. It uses a property of human perception that they called *emergence* (Mitra et al., 2009a,b): “*Emergence* refers to the uniquely human ability to aggregate information from seemingly meaningless pieces”. In their CAPTCHA, a video in which an object is moving is processed so the background is converted into seemingly-random noise, and so is the moving object, although in a way that the human perception can easily follow it. The authors claim that there is not enough information on a single frame to detect the moving object. This might be so, but it remains to be analysed whether information cannot be easily extracted from successive images. Unfortunately, this proposal did not reach production phase, so it could not be tested.

2.4.3.3 Face classification and identification CAPTCHAs

Face identification is a very well known CV problem that has been studied for long. Several CV algorithms can locate faces in pictures and extract informa-

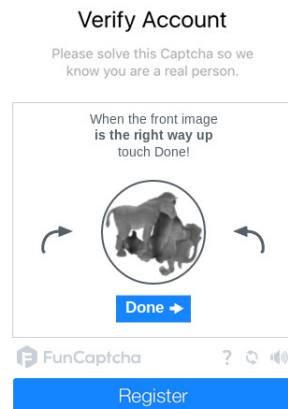


Figure 2.13: Example of FunCAPTCHA orientation CAPTCHA in “high-security mode”.

tion from them, including the location of the different facial features. Several face-related problems have been proposed as a base for several CAPTCHA designs. An early CAPTCHA that rely in face identification (Goswami et al., 2014*b,a*) was broken a year later (Gao et al., 2015).

Facebook, a popular social network in western countries, has heavily invested in face recognition and moved from a feature-based approach (Facebook, 2011) to a DL approach (Taigman et al., 2014). This change resulted in a huge improvement in their error rates (97% on the Labelled Faces in the Wild (LFW) dataset). They use their new SW to automatically suggest tags for people appearing in the pictures uploaded to their site.

Facebook is at a prominent position given its picture database. To

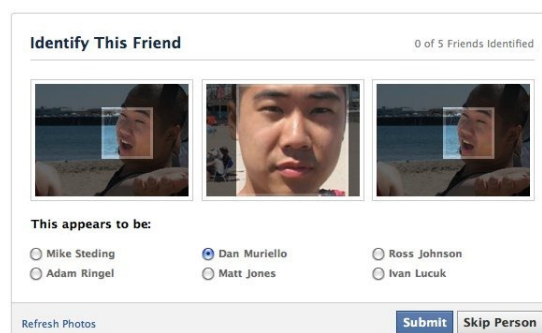


Figure 2.14: Example from the Facebook Social Authentication CAPTCHA.

use it, Facebook studied the use of their face identification data as the base for a CAPTCHA. The idea for their *social* CAPTCHA is to present a picture of one of your Facebook friends: you will need to identify the person to authenticate yourself (as shown in Figure 2.14). This protection can be used when you are trying to retrieve a lost password or when Facebook detects suspicious login or posting activity from your account. If your answers are wrong, your account is locked down and you can try again after a period (apparently 1 hour).

2.4.4 Game-based CAPTCHAs

In the earlier part of the decade of the 2010s, several proposals appeared that tried to increase the usability of CAPTCHAs by making them appear as small, simple games (Paxton and Tatoris, 2012, Gosschalk and Ford, 2016). This technique was termed “gamification”. It consists on making the challenges appear as a small, simple game. The User Interface (UI) also improved to include techniques like drag & drop, more user friendly, especially when using mobile (tactile) devices. The underlying mechanisms for the CAPTCHAs did not change abruptly, but the interaction with the user was improved.

The idea of benefiting from games is not new to CAPTCHAs. Some other researchers have used game-like strategies to create data for their CAPTCHAs (Von Ahn and Dabbish, 2004). But now, the game (or game-like interaction) takes the central part of the CAPTCHA UI.

One of the first production CAPTCHA to use these techniques is the one created by *Are You a Human*. They named it the *PlayThru* CAPTCHA, that they also use for advertisement purposes. It is composed of small drag & drop games (see Figure 2.15).

Another related example, already presented, is the FunCAPTCHA (Gosschalk and Ford, 2016), that we analyse as a case-study in Chapter 5. It uses a “click” interface for their rotation CAPTCHA and a drag & drop interface for their genre recognition CAPTCHA.

Another subcategory are the puzzle CAPTCHAs: they require the user to drag & drop pieces of an image in order to reconstruct the original. Some examples are Capy, KeyCAPTCHA and Garb. We analyse them in Chapter 3.

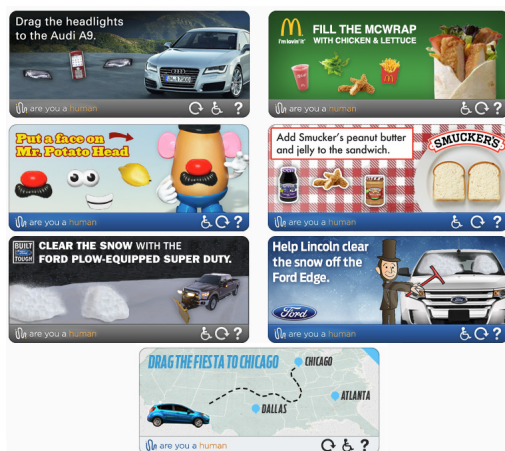


Figure 2.15: Examples from the PlayThru CAPTCHA.

2.4.5 CAPTCHAs based on the understanding of video

Some ideas have appeared that are purely based on video. These are different from the proposal based on *emergence* that we discussed (Mitra et al., 2009a) and other proposals based on adding animation to OCR/text CAPTCHAs (NuCaptcha, 2016). We will call them *pure-video* CAPTCHAs because they are based on extracting semantic information from the sequence of actions that the video portrays. Some of these proposals are the ones from Kluever (2008), Hernandez-Castro and Ribagorda (2009b) or the similar “motion and interaction based CAPTCHA” (Qvarfordt et al., 2013). They have not been implemented, so a proper security analysis is missing.

For any CAPTCHA based on video, there is the concern that the additional information that the video will provide will somehow make it easier to find clues in order to break these CAPTCHAs. No proper security analysis can be done until there is a public implementation.

2.4.6 Audio CAPTCHAs

Most CAPTCHAs proposals have been based at least partially on the visual capacities of ordinary humans, but there is a number of people who have vision problems. Some CAPTCHAs that have been put into production have had to provide an alternative for visually impaired users. This is the way that most audio CAPTCHAs proposals have appeared, as an alternative to

visual ones. Audio CAPTCHAs were not typically the *first thought* of their designers, and thus they were possibly designed with less emphasis on their security.

One of the most popular audio CAPTCHA was the Google audio CAPTCHA, presented in 2008¹⁰. Each challenge consisted on a series of digits being spoken with background noise. It was clearly not enough secure, as back in 2008, it could be broken using very basic methods (Santamarta, 2008, Tam et al., 2008).

This has led to an increase in their difficulty, adding noise and choosing audio cues more that are difficult to understand. Current audio-based CAPTCHAs are extremely difficult for humans (Bigham and Cavender, 2009). Even after improvements, the audio version of reCaptcha by Google was broken again using simple speech recognition (Sano et al., 2013) and later using the speech recognition API of Google (Sidorov, 2017). It remains to be seen if a strong yet usable audio CAPTCHA could be created.

2.4.7 Alternative problems for CAPTCHA designs

We have introduced the broader category of CAPTCHAs: those based on *AI-hard* problems (or thought to be). Now we present the much less common CAPTCHAs based on alternative problems: those problems not tackled by AI nor considered to be *AI-hard*. The fact that they are not considered *AI-hard* does not imply that these problems could not be solved or bypassed using ML techniques; it only means that the problems are not traditional problems previously studied in AI nor ML. We present these proposals in the following paragraphs.

Capturing your face This idea relies on having a camera (a web-cam on a PC, or a frontal camera on a phone) to pass a CAPTCHA that requires the user to produce certain actions (Greenblatt and Lagares-Greenblatt, 2012) or gestures (De Marsico et al., 2016) (as shown in Figure 2.16). The idea might seem too intrusive to certain users though, and given that the video is taken at the clients' location, it is susceptible to be tampered with or faked.

¹⁰According to <https://devopedia.org/captcha>, visited on Aug. 2017.



Figure 2.16: Webcam-CAPTCHA design (Greenblatt and Lagares-Greenblatt, 2012). The user is requested to perform some gestures in front of the camera.

Detection of movement Smart-phones, tablets, etc. incorporate a panoply of sensors that are not found on the typical desktop or laptop computer. Among those, there is typically a motion detector. There are different proposals to use it in order to pass a CAPTCHA.

In one of them, the CAPTCHA asks the user to use gestures to operate specific objects on the screen so as to complete a CAPTCHA challenge (Jiang and Tian, 2013, Jiang and Dogan, 2015) (shown in Figure 2.18).

In another proposal, the CAPTCHA asks the user to perform gestures “from everyday life” such as hammering using the smart-phone as if it was a hammer, while the user has to hit a nail five times (Hupperich et al., 2016).

In a similar way, other researchers propose to use these movement tests to enhance the protection of typical security measures, like inputting a PIN number for authentication. This approach is called CAPPCHA, or *Completely Automated Public Physical test to tell Computers and Humans Apart* (Guerara et al., 2017). In a typical CAPPCHA, the user has to tilt the mobile terminal in different degrees as required (see Figure 2.18).

These CAPTCHAs variants or alternatives might be interesting, but their security remains unknown until there is a public implementation that can be studied. The fact that the movement sensor resides in the client

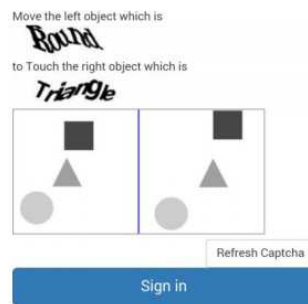


Figure 2.17: Movement CAPTCHA design (Jiang and Dogan, 2015). The user is asked to move her mobile device in order to move the corresponding object to the desired location.



Figure 2.18: Physical CAPTCHA, or CAPPCHA design (Guerara et al., 2017). The user is asked to rotate her mobile device to a certain degree prior to access some other security measure.

machine, and this, might be simulated by software, and also that the whole mobile platform can be simulated by software, shadows some concerns about these alternatives.

Human mistakes If there was a simple, automated way to detect humans by the known mistakes that they make, typically related to well-known perception effects, we could use this idea to create a CAPTCHA that would create challenges and allow only to pass of the answers include these perception known biases. This idea is for example used in a patent from the web seller Amazon (McInerney et al., 2017) (see figure 2.19). There are problems to solve, though. If there is an automated way to grade these challenges, there is an automated way to solve them. Additionally, with enough data, it might be possible to train a ML algorithm, particularly a DNN, to mimick the errors made by the human perception.

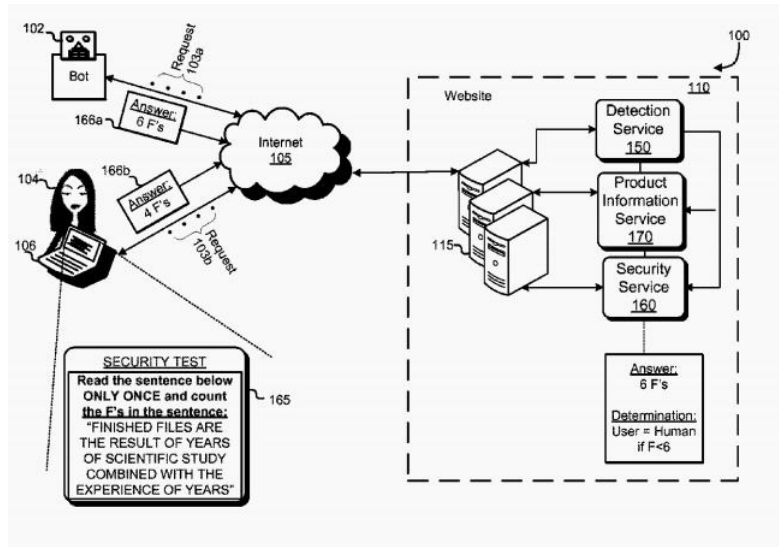


Figure 2.19: Example challenge from the proposed Amazon CAPTCHA asking the user to read just once a sentence while counting the appearances of a particular character (McInerny et al., 2017).

2.4.8 So-called “behavioural” CAPTCHAs

Even though many bot detection mechanisms are marketed as “no-CAPTCHA” alternatives, they are a mix of CAPTCHAs and algorithms to decide whether to display them or not as well as with what level of difficulty. These different proposals typically resort to some conventional CAPTCHA when they determine that there is not enough information. They constitute one of the main current trends in the CAPTCHA world. They sometimes call themselves *behavioural analysis*, which is a fancy term to refer to more or less typical mechanisms to automatically create blacklists of potential attackers and/or white-lists of low-risk users. These mechanisms associate a level of potential danger to each *different*¹¹ client.

The current trend is to decide to show or not a CAPTCHA challenge depending on the associated risk level. Also, the difficulty of the challenge presented to the user might be affected by this risk level. This can mean that a user is faced with successive hard CAPTCHA challenges for no obvious reason apparent to her.

This idea is strongly related to blacklists/white-lists, as seen in

¹¹Note that the word *different* here means different as regarded per the server. This does not necessarily mean a real different client, depending on the possible scenarios.

section 2.3. Depending on its particular implementation, it can be considered a *white-list*, allowing some users to bypass the CAPTCHA, or a *blacklist*, increasing the difficulty of the access to some users through CAPTCHAs created using their hardest security settings. It can also be implemented as a combination of both.

This idea is an example of *Security through Obscurity*, as the mechanisms used to assess the risk of the different clients are not public, and their strength relies precisely on these mechanisms not being known.

2.4.8.1 reCAPTCHA

This trend has been accepted by the current main actor in the CAPTCHA scenario, which is Google. Already in 2014, they commented that “we have significantly reduced our dependence on text distortions as the main differentiator between human and machine (...) and instead perform advanced risk analysis”¹².

Going in more detail, Google reCAPTCHA Product Manager Vinay Shet described that “Google has begun actively considering the user’s entire engagement with the CAPTCHA: before, during and after they interact with it. That means that today the distorted characters serve less as a test of humanity and more as a medium of engagement to elicit a broad range of cues that characterize humans and bots”¹³. This is not new, as the general idea of using a client’s interaction with a web-page or web-site to measure her chances of being human are already present in Baird and Bentley (2005).

Further in this direction, in December 2014 Google introduced something that they called “No CAPTCHA reCAPTCHA”. This was an initial public relations success, as the words “No CAPTCHA” were understood as that Google got rid of the need of using CAPTCHAs altogether thanks to their technology. When it was better explained, it was understood that it was just another step in the same direction, in which they would white-list some users of which Google had *enough data* as to associate to them a very

¹²From The Atlantic article “CAPTCHAs Are Becoming Security Theatre”, located at <http://www.theatlantic.com/technology/archive/2014/04/captchas-are-becoming-security-theater/360786/>, retrieved in 2016.

¹³From the Google Security Blog post “reCAPTCHA just got easier (but only if you’re human)”, located at <https://security.googleblog.com/2013/10/recaptcha-just-got-easier-but-only-if.html>, retrieved on 2016.

low level of risk (Shet, 2014a).

2.4.8.2 Other “behavioural” proposals

Google has not been the first to follow this path. Another example is NuCaptcha. They proposed an improved OCR/text CAPTCHA that incorporated moving characters. Their system uses a combination of what they call a “behaviour analysis system to monitor all interactions on the platform” and modify the difficulty of the CAPTCHA challenge¹⁴.

There are more examples of this trend. One of them is Mollom, which uses the same ideas, and finally falls back to their own CAPTCHA if a user’s risk assessment is high or unknown¹⁵. Another example is Capy¹⁶. They introduced a puzzle CAPTCHA that was broken (Hernández-Castro et al., 2014). After they were contacted with the attack info, they introduced “Capy Risk-Based Authentication”, an “authentication system which takes into account the profile of each user requesting access to the system to determine the (login) history”¹⁷. An additional example is *Are You a Human*, that introduced the *PlayThru* CAPTCHA, and after it was broken by Mohamed et al. (2013), started offering their *Real Time Human Detection* and *Verified Human Whitelist* solutions¹⁸.

2.4.8.3 Discussion of “Behavioural” Analysis

The use of *behavioural analysis* is prone to errors that can miss-classify a legit user for an abuser, leaving her with the need to pass a CAPTCHA challenge for every petition.

The bottom point is that the so-called *behavioural analysis* introduces a benefit and some possible drawbacks:

¹⁴From <http://www.nucaptcha.com/security-features>, retrieved on November 2016.

¹⁵From <https://www.mollom.com/how-mollom-works>, retrieved on November 2016.

¹⁶Located at <https://www.capy.me/>, retrieved in November 2016.

¹⁷From https://www.capy.me/products/risk_based_authentication/, retrieved in November 2016.

¹⁸Both approaches are described at <https://areyouahuman.com/solutions>, retrieved on November 2016.

- Typically, legit users that have a long-enough (or *apparently-secure-enough*) track will be able to bypass the CAPTCHA while their track is still *apparently-secure-enough*, but ...
- An abuser that hijacks the profiles of these users (for instance, through a botnet) might use them to further abuse, lowering their security assessment, and forcing them to solve CAPTCHA challenges for their real queries that, in turn, will allow further attacks.
- If there is not enough data from a user, if the user is somehow related to a case of abuse (same sub-network, etc.), or if the user wants to keep her privacy through the use of semi-anonymous networks as Tor or web browsers with high privacy settings, then it will have to turn to its base-case scenario and present CAPTCHA challenges to these legit users, maybe using a *hard* version of them. This, that was the common and accepted behaviour before, is now seen as a discrimination compared to other users.

The recent controversy between Cloudflare¹⁹ and Tor developers and users has lead is a good example of the latter case. This controversy has led to a war of declarations between both²⁰ and a privacy threat (Anonymous, 2016).

In 2016, after a number of complaints, and looking to achieve an increase in usability, Cloudflare started developing a plug-in that sits on the client's browser and allows to limit the number of CAPTCHAs presented to the client under certain circumstances²¹.

Cloudflare still requires an initial CAPTCHA solution, but in certain scenarios, future challenges to the user can be avoided, as they are controlled *centrally* by the browser plug-in. This idea is not really an alternative to CAPTCHAs, as the solution itself incorporates a CAPTCHA in order to work. What it does is centralise the CAPTCHA information in the browser and share it with different servers through cryptographic protocols.

¹⁹Cloudflare provides security services, cache/proxy services and DNS services, placing their servers between the web-site client and the server (their clients).

²⁰March 2016 Cloudflare blog article "The Trouble with Tor" by Matthew Prince, CEO & Co-Founder of Cloudflare, located at <https://blog.cloudflare.com/the-trouble-with-tor/>.

²¹More information at Cloudflare development folder at GitHub at <https://github.com/cloudflare/challenge-bypass-specification/blob/master/captcha-bypass-formal-spec.txt>.

In summary, the so-called “behavioural” mechanisms are heavily based in *Security through Obscurity*, that as we have mentioned in section 2.2.2, is not a time-proof way of designing security measures.

As an example of this, some of these CAPTCHA proposals have been broken, as we will explain in section 2.5.6.

A major influence on the evolution of CAPTCHA design has been the different successful attacks against them. In the next section, we present the most relevant attacks against CAPTCHAs.

2.5 Attacks against CAPTCHAs

One key element affecting CAPTCHA design remains to be introduced, and that is the different attacks that have been successfully able to break CAPTCHAs/HIPs. These attacks have strongly guided the evolution of CAPTCHA/HIP design. Not all attacks have been public, and is not unusual to see that a CAPTCHA design evolves without presenting a reason why. In general, the evolution of CAPTCHAs design has followed a path to avoid known weaknesses, so it is reasonable to assume that the main attacks, those more fundamental to their design, have seen the light.

Much as Cryptography and Cryptanalysis evolve in tandem, so do CAPTCHA design and CAPTCHA breaking. In IT Security it is fundamental to assess the security of a proposal. If not, a false sense of security might arise, and translate into very low levels of real protection.

In this section we present the most significant attacks to the different types of CAPTCHA designs mentioned in section 2.4.

2.5.1 Attacks to text recognition (OCR) CAPTCHAs

In the early 2000s, two CV researchers used their already developed framework for object detection to successfully break both the EZ-Gimpy CAPTCHA in use at Yahoo! and the *hard* Gimpy CAPTCHA (Mori and Malik, 2003). Their work is notable because of two aspects. It was the first research paper that was peer-reviewed and published that focused on finding weaknesses on a CAPTCHA and breaking it. The second important aspect is that it

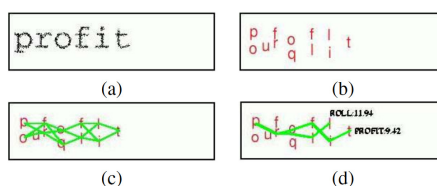


Figure 2.20: Example of Mori & Mali attack to the Gimpy CAPTCHA using their first algorithm: (a) is the original Gimpy challenge (b) edge detection output (c) hypothesized bigrams (d) pixels remaining after guessing the word “round” and removing its pixels (Mori and Malik, 2003).

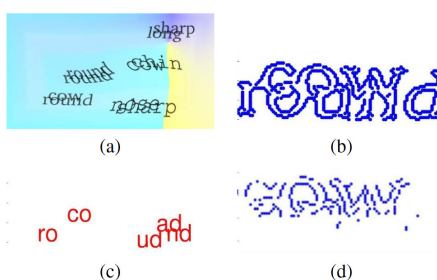


Figure 2.21: Example of Mori & Mali attack to the Gimpy CAPTCHA using their second algorithm: (a) is the original Gimpy image (b) Locations of hypothesized characters (c) direct acyclic graph of possible strings (d) scores of top matching words and their graphs after pruning and dictionary check (Mori and Malik, 2003).

showed some weaknesses of the *hard* Gimpy CAPTCHA, but contrary to the assumptions of Ahn et al. (2003), these weaknesses and their exploit were not clearly applicable to any OCR/text CAPTCHA, thus not improving the state-of-the-art of OCR. Contrary, these weaknesses were related only to the way in which these particular CAPTCHAs obfuscated the characters. Figure 2.20 shows their first attack, in which they locate possible characters through edge detection and hypothesize different bigrams based on their likelihood. Figure 2.21 shows their second attack, in which they also find possible locations for characters based on edge detection and prune the possible strings using a dictionary.

This started the *arms race* between CAPTCHA developers and breakers - note that many times, the same researchers have tried to work in both bands. An example is professor Mori, who advised on the design of NuCAPTCHA (NuCaptcha, 2016), an animated OCR/text CAPTCHA, later broken almost simultaneously by Bursztein (2012) and Xu et al. (2012). Another such example is the work of Professors Chellapilla and Simard from

Microsoft, in which they break different OCR/text CAPTCHAs and identify segmentation as the most challenging tasks for the attacking algorithms. Then, they designed an OCR/text CAPTCHA that bases its strength in the segmentation problem and deployed it at Microsoft (MSN Passport) (Chellapilla and Simard, 2005, Chellapilla et al., 2005*b,a*).

This *arms race* is very much similar to the one in which Cryptography & Cryptanalysis have been involved for hundreds of years. That race has created stronger cryptographic algorithms, and also allowed us to better understand the foundations of the security of cryptosystems. With the implicit intuition that a similar race would help evolve the security of CAPTCHAs, many researchers and developers engaged in this race in the following years. Other researchers explicitly mention this hope, as Gao et al. (2016) and Yan and Ahmad (2008).

In 2005, Chellapilla and Simard were able to attack several OCR/text CAPTCHAs in use. Their work led to the proposal that new OCR CAPTCHAs schemes should be based on hard-segmentation problems (Chellapilla et al., 2005*a,b*). Several approaches have focussed on making this division harder, sometimes at the expense of making it also harder for the human user.

In 2007, Yan and Ahmad (2007) were able to successfully attack Captchaservice.org using quite unsophisticated but effective algorithms. In order to do so, they used pixel-counting of contiguous regions for character detection as well as vertical pixel counting for segmentation, attaining a 36% success rate. Adding a dictionary look-up assisted by a total pixel sum matching, as well as a dictionary pruning for characters with similar pixel count, as well as some other simple heuristics, they attained a 94% success rate, again increased up to 99% with additional heuristics.

Even though by 2007 most OCR CAPTCHAs had been defeated one or several times, the main trend for companies was to update these CAPTCHAs to try to evade the current attacks, typically making segmentation harder.

In 2008, Yan and El Ahmad publish again an attack on the CAPTCHA deployed by Microsoft in services like Hotmail, MSN and Windows Live. This CAPTCHA had been designed specifically to be segmentation-resistant (for an example of this, see Figure 2.22), and was partially based on the works of well-known experts in CAPTCHA analysis (Chellapilla and Simard, 2005,



Figure 2.22: Example from the Microsoft CAPTCHA in 2008 (Yan and Ahmad, 2008). Note the lines drawn at random to try to prevent segmentation.

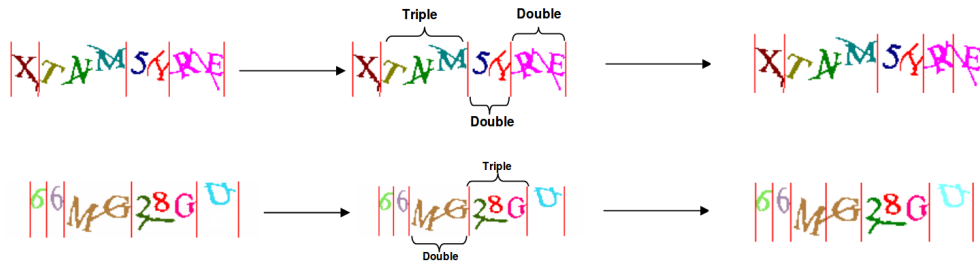


Figure 2.23: Example of the segmentation of a challenge from the Microsoft CAPTCHA (Yan and Ahmad, 2008). In this image we can see their segmentation phase.

Chellapilla et al., 2005*b,a*), working with an “interdisciplinary team of diverse expertise in Microsoft including document processing and understanding, ML, HCI and security” (Yan and Ahmad, 2008).

In order to segment these characters, Yan and El Ahmad use a similar idea to that they had already used before (Yan and Ahmad, 2007) based on counting pixels per columns. This time they also detect continuity of groups of pixels by flood-filling. This idea helps with those chunks of characters not correctly segmented by using the vertical pixel histogram that contains more than one character. Figure 2.23 shows an example of their segmentation phase: using a vertical pixel histogram, their algorithm is able to identify the different segments $|X|TNM|5Y|RE|$ and $|6|6|MG|28G|U|$. Using flood-fill colouring, it is able to identify TNM and $28G$ as triple characters, and $5Y$, RE and MG as doubles. Double characters are segmented by averaging their width. Yan and El Ahmad enhance their algorithm with some heuristics for the removal of arcs. With these simple algorithms, they are able to break the Microsoft CAPTCHA on 92% of the occasions (Yan and Ahmad, 2008).

In 2010, El Ahmad and Yan are able to break the CAPTCHA of

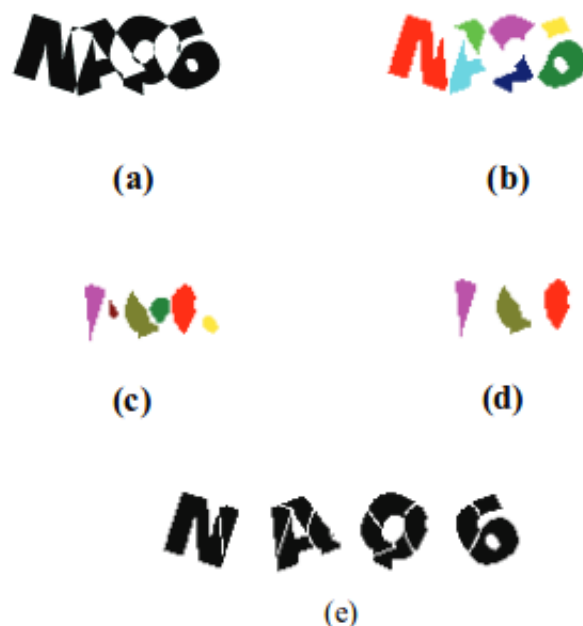


Figure 2.24: Example of restoration of a challenge from the Megaupload CAPTCHA (El Ahmad et al., 2010).

a popular file sharing (Megaupload.com) that used substantial overlapping to avoid segmentation. They did so identifying and merging character components (El Ahmad et al., 2010). Figure 2.24 shows an example of their attack, working to reconstruct the characters *NAQ6*. In this figure, (a) shows the original Megaupload challenge, where the characters heavily overlap in order to avoid segmentation attacks; (b) is the result of the extraction of black components, and (c) of white components (excluding the background). Subfigure (d) shows the extraction of shared components, and (e) the merging of them along with the ones from (b) into the original characters.

An important break work was published by Bursztein et al. (2011, 2014). It uses ML to attack both character segmentation and recognition simultaneously, scoring hundreds of different possible segmentation decisions. They use well-known ML algorithms like kNN, a voting mechanism based on ensemble learning and resembling Random Forests, reinforced learning for validating the segmentation segments, etc. Their approach is able to break all the most used OCR CAPTCHAs of the time, like the ones used by Baidu, eBay, reCAPTCHA, Yahoo! or the Wikipedia.

In 2016, Gao et al. (2016) published another generic attack for OCR

CAPTCHAs. Gao's attack is based on Log-Gabor filters. A 2D Gabor filter is a Gaussian kernel function that is being modulated by a sinusoidal plane function. Over 2D Fourier or DCT transforms, 2D Gabor filters have the advantage of being able to localise the origin of the frequency. These filters are thought to be able to model the response of the neocortical neurons, and thus be similar to how the first steps of the perception works in the human visual system. Gao et al. (2016) use them to break the characters into their different components.

Figure 2.25 shows an example, where subfigure (a) shows the character components of a CAPTCHA challenge from QQ (each component is painted in different colours), and subfigure (b) shows the same segmentation of components for a challenge from the CAPTCHA at Microsoft. Gao et al. (2016) use kNN to choose the most probable word from the graph of character components. When they try their attack on CAPTCHAs deployed by the top 20 most popular websites according to Alexa ranking, they found that their attack successfully breaks all of them, with success rates varying from 5% for Yahoo! up to 77.2% for reCAPTCHA. They reach these rates with no pre-processing, even for hollow characters.

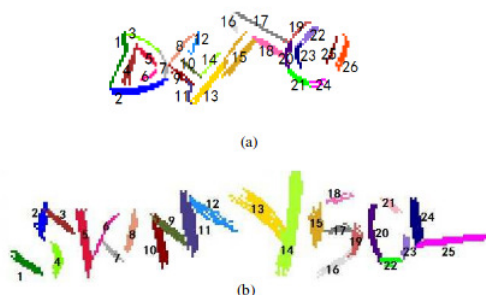


Figure 2.25: Example of segmentation of character components using Log-Gabor filters (Gao et al., 2016).

The attacks of Bursztein et al. (2014) and especially Gao et al. (2016) that affect several OCR/text CAPTCHAs do pose a question mark on the possibility of continuing to use OCR/text CAPTCHAs as a security mechanism, at least in the ways they are used now.

Yet as bad as they are, they might not be the most devastating attacks to OCR/text CAPTCHAs. Starting in 2012, results attained with NNs (Neural Networks) started to increase in accuracy, thanks the availability of more examples, new network architectures, units and training algorithms and new ways to use parallel hardware (GCGPUs, or General Computing

on Graphical Processing Units). This led to a significant increase in the accuracy of NNs. This has a major repercussion on CAPTCHAs. For further discussion on this, we refer the reader to section 2.6.1.

During the evolution of the research in OCR CAPTCHAs, it was seen that distortions to characters have their limits, especially when computers are better at recognising single characters, and segmentation can be solved using NNs and other methods. Thus, some researchers looked into how to still use characters, but using a different representation that could be made harder for machines. This is how the ideas of 3D OCR CAPTCHAs and animated character CAPTCHAs started.

3D OCR/text CAPTCHAs The Teabag 3D CAPTCHA is a 3D OCR CAPTCHA designed by a very knowledgeable group of CAPTCHA security analysts and programmers called the *OCR Research Team* (Kolupaev and Ogijenko, 2013). This 3D OCR/text CAPTCHA was broken by Nguyen et al. (2011, 2014b) in what is the first breakage of a CAPTCHA of this type. Figure 2.26 illustrates how their attack works: first, it performs character extraction by distinguishing triangles of different sizes and shadows, following it performs the segmentation of these elements, next it does some post-processing in order to remove artifacts, and finally it recognizes the characters. Some ideas on how to attack it previously appeared in Hernandez-Castro and Ribagorda (2009a). Other 3D OCR/text CAPTCHAs proposals broken by the same authors include the one they call *3dcaptcha* and *Super CAPTCHA* (Wells, 2011).

Some other proposals, including a 3D moving CAPTCHA (Kund, 2011) as well as a 3D CAPTCHA that uses the human ability of stereoscopic vision (Susilo et al., 2010) have not been implemented yet, so it is not possible to properly assess their strength.

Animation of OCR/text CAPTCHAs The “zero knowledge per frame” principle (Cui et al., 2010) was used in several production CAPTCHAs. In 2012, Nguyen et al. successfully attacked several OCR/text animated CAPTCHAs (Nguyen et al., 2012a). HelloCAPTCHA, another example of this idea, was also broken by Nguyen et al. (Nguyen et al., 2012b) using different frames from the animation and detecting the different elements. Figure 2.27 shows an example of their processing pipeline breaking one challenge from HelloCAPTCHA. NuCaptcha (NuCaptcha, 2016), a similar OCR CAPTCHA

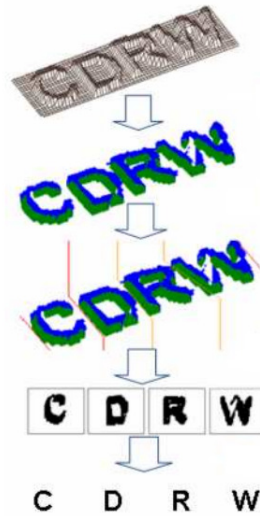


Figure 2.26: Steps to break the Teabag 3D CAPTCHA in successive order (Nguyen et al., 2011).

that added animation to its characters, was broken by Bursztein (2012) using different techniques like Scale-Invariant Feature Transform (SIFT) to find interesting *regions*, isolating the most “interesting” object in each frame, detecting the most interesting 50 frames, and then using their previously published techniques for segmentation and recognition (Bursztein et al., 2011, 2014).

Alternative OCR/text CAPTCHAs Ideas The Quantum Random Bit Generator Service CAPTCHA was based on a completely new idea, and gained widespread publicity, both positive and negative. Hernandez-Castro and Ribagorda (2010) found that this CAPTCHA has important design limitations that render it vulnerable. Possibly the most important one was the skewed distribution of correct answers and that the answers were all integers. For example, for the tests based on derivatives, 0 is by far the most common correct answer. Also, it was possible to use the CAPTCHA as an oracle.

Captcha2 was a strange commercial proposal, as since the work of Chellapilla et al. (2005a) it is understood that a proposal like this would very probably be able to be broken using CV/ML methods. Thus Captcha2 made special emphasis in character obfuscation techniques. Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda (2010) were able to break it using straight-forward methods for background removal and pixel counting of

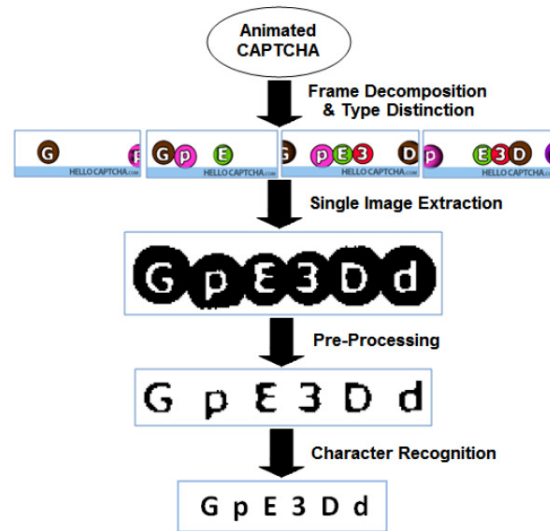


Figure 2.27: Steps to break Hello CAPTCHA (Nguyen et al., 2011). First, the frames are analysed to extract a single image. The rest of the steps are common with other OCR CAPTCHAs.

the different contiguous regions. This was possible because it had the major flaw of using bigger font sizes for the correct characters.

2.5.2 Attacks to language/semantic CAPTCHAs

TextCAPTCHA, a textual question generator, presents important design flaws that allow to easily reverse-engineer it. In particular, it is straightforward to detect which subtype of challenge it is using, and thus apply an ad-hoc solver to each case.

The Egglue CAPTCHA uses a proprietary algorithm, accessible through a web service, that creates two sentences that the user has to fill in with the correct verb. Its mechanisms remained as a black-box, with no information on how Egglue created and marked the challenges. After some research, it was seen that the algorithm it is using for marking a challenge was not strong. For example, it allowed using general verbs successfully even for sentences for which they did not make sense. Figure 2.28 shows that some verbs have a success rate over 90%, which is clearly not related to the distribution of appearances of verbs in English. This also implies that several sentences are considered correct with many different verbs. Both CAPTCHAs

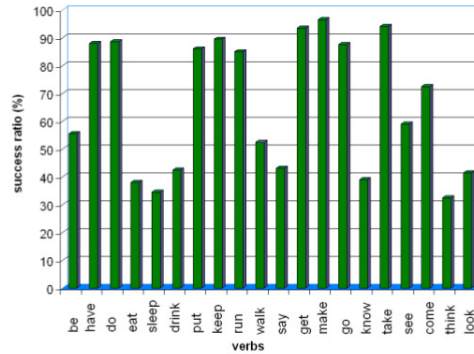


Figure 2.28: Verbs and success rates for the Egglue CAPTCHA (Hernandez-Castro et al., 2011).

were broken by Hernandez-Castro et al. (2011).

2.5.3 Attacks to image classification CAPTCHAs

The HumanAuth CAPTCHA requests to distinguish between pictures depicting either a natural object (a tree) or an artificial one (a watch). It was an Open Source project that was shipped with 69 pictures, very lightly obfuscated using a watermark. This obfuscation did not serve much, as each image had assigned a textual description for the visually impaired (Gigoit, 2006). It was easily broken using some simple metrics from each image measured using the ENT pseudorandom number sequence test program from the Fourmilab²² and training an ML classifier on these metrics. This was possible even when the CAPTCHA was using the watermark and the attack did not take advantage of the textual description (Hernandez-Castro, Ribagorda and Saez, 2010, Fritsch et al., 2010).

The ASIRRA CAPTCHA published an initial security assessment, and even more, their authors provided a training set for anyone willing to try their ML algorithms on it. Golle (2009) experimented with similar ML methods to the ones used by the creators of ASIRRA, using different features to train a SVM classifier. His SVM used a radial basis kernel. The most successful features were boolean colour presence (if a colour was or not present in a certain part of the image) and 5×5 -pixel texture features, selected at random and filtered to be different enough (an example is shown

²²Available at <http://www.fourmilab.ch/random/>.



Figure 2.29: Example of 5x5-pixel textures used as features for the SVM (Golle, 2009). They were extracted randomly, and then filtered in order not to use too similar ones.

in Figure 2.29). Golle was able to break ASIRRA with a success rate of 10.3% (82.7% accuracy for a single image).

Next, we will present attacks related to other CAPTCHAs also based on images, but not on a typical image-classification problem, yet in other problems that are related to CV.

Alternative Image-based Ideas The IMAGINATION CAPTCHA (Datta et al., 2005) has two phases. The first one, most interestingly, asks the user to click in or around the centre of any of the images that form a mosaic. This mosaic of images is created in a way that makes it difficult for known CV techniques to segment it. This phase presents some usable deviations from random that might render it vulnerable (Hernandez-Castro and Ribagorda, 2009a). This proposal was broken by Zhu et al. (2010a) using a clever algorithm to find candidates for image boundaries (shown in Figure 2.30).

Another such example was the proposal by Gossweiler et al. (2009) from Google Research. Their CAPTCHA presents rotated images to the user. The user has to rotate them back to their original orientation. The brute force attack success rate will depend on the tolerance of accepted answers. Taking into account the data given by Gossweiler, their CAPTCHA accepts an answer within a 16% margin. In this set-up, they report a brute-force success rate of .009% for a challenge with three images. The success rate for a single image seems to be $\sqrt[3]{0.00009} \approx 4.48\%$, too high for a production

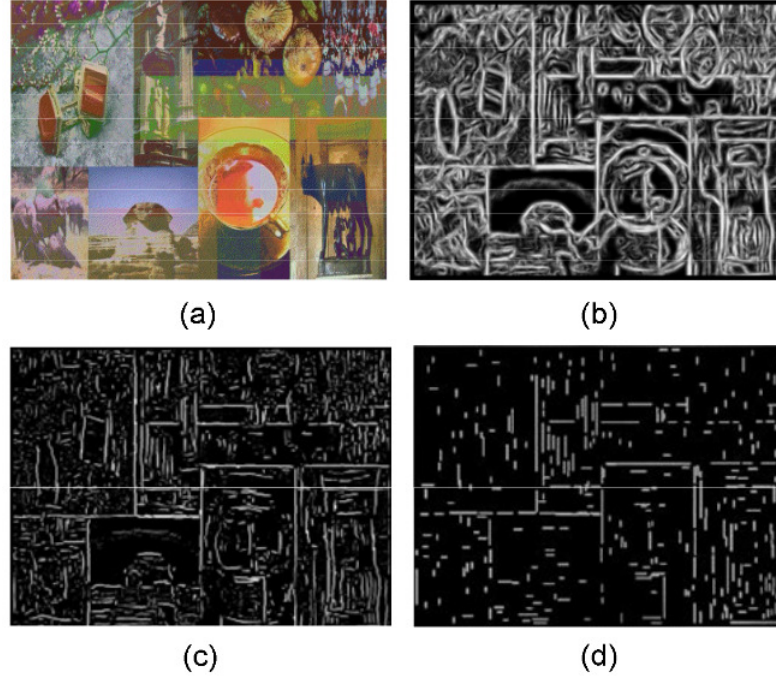


Figure 2.30: Example of edge detection for a first challenge of the IMAGINATION CAPTCHA (Zhu et al., 2010a). (a) is the original image (b) is the map of edge candidates (c) is the complete edge map (d) shows the horizontal and vertical line segments detected.

CAPTCHA. This proposal was never implemented at large scale by Google, so a proper analysis is pending.

Face Classification and Identification Several proposals have used the problem of gender recognition of face pictures (Kim et al., 2014, Sim et al., 2014, Gosschalk and Ford, 2016, Schryen et al., 2016). The only one that was put into production is FunCAPTCHA (Gosschalk and Ford, 2016), that we analyse in Chapter 5. FaceDCAPTCHA (Goswami et al., 2014a) and FR-CAPTCHA (Goswami et al., 2014b) are two CAPTCHAs based on human face recognition. FR-CAPTCHA asks the user to find matching pairs of human faces in an image. FaceDCAPTCHA presents images of both real and fake faces, distorted and partially occluded, and asks the user to select the images containing real faces. Both were broken by Gao et al. (2015). They employed edge detection and SVM classification to differentiate the images of real and fake human faces using as features color, texture, LBP, SIFT and Laws' Masks in order to break FaceDCAPTCHA. To break FR-CAPTCHA,

they extract four features and compare them amongst the images to find probable pairs.

Facebook studied the use of their face identification data as the base for a CAPTCHA. This CAPTCHA proposal was analysed by Kim et al. (2012) finding possible attacks. It was later broken using well known classifiers: they try both kNN and SVC with better results, but choose kNN as results are similar and it is computationally less expensive. They gather the training data from public data, obtaining a 22% success rate. Figure 2.31 shows their results: overall they solve correctly 28/127, identifying correctly a minimum of five of the seven friends presented. More so, in 71 additional tests (71/127 = 56%), their attack identifies correctly two to four friends, so the attacker can perform brute-force guessing attacks with $O(10^{-1})$ to $O(10^{-2})$ success rates. Optionally, they perform a *social engineering* attack to reach “sensitive” data. They do so using fake Facebook profiles to befriend friends of the target. With the data collected through *social engineering*, they reach a 100% success rate (Polakis et al., 2012).

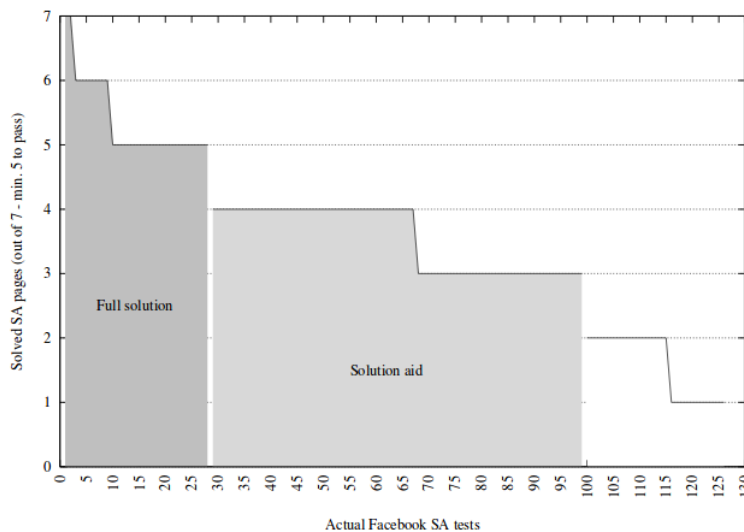


Figure 2.31: Success rate of the attack against the Facebook Social Authentication CAPTCHA (Polakis et al., 2012). Facebook requires a minimum of five correctly classified faces from *friends* in a set of seven challenges.

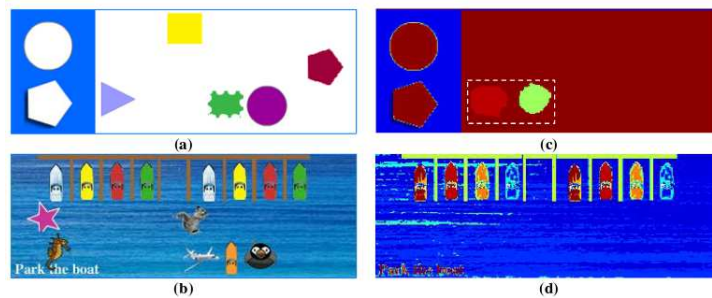


Figure 2.32: Background detection for a drag & drop game CAPTCHA (Mohamed et al., 2013). Each row shows the detection of non-moving background for a challenge.

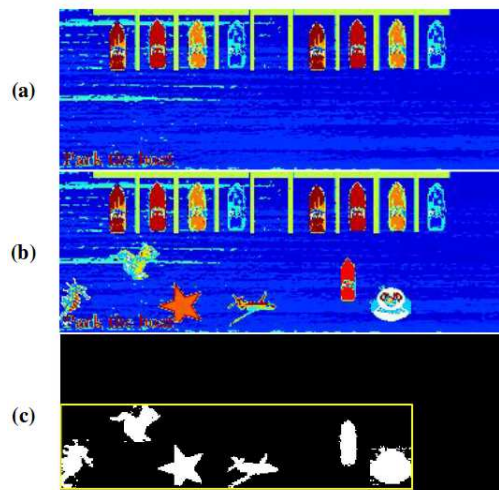


Figure 2.33: Target detection for a drag & drop game CAPTCHA (Mohamed et al., 2013). Here we detect the objects that are movable.

2.5.4 Attacks to game-like CAPTCHAs

One of the first production CAPTCHAs to use gamification techniques is the one created by *Are You a Human*, their *PlayThru* CAPTCHA. It is composed of small drag & drop games. They use simple heuristics to detect the background: Figure 2.32 shows the different steps of this detection. Similarly they detect the foreground objects (the detection steps are shown in Figure 2.33), as well as learning from the objects by remembering them, Mohamed et al. (2013) are able to easily break this CAPTCHA.

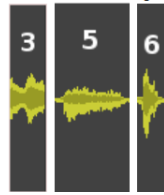


Figure 2.34: Wavefront recognition of digits in Google Audio CAPTCHA (Santamarta, 2008). Given that the digits are spoken with a volume higher than the background noise, their waveform remains similar each time.

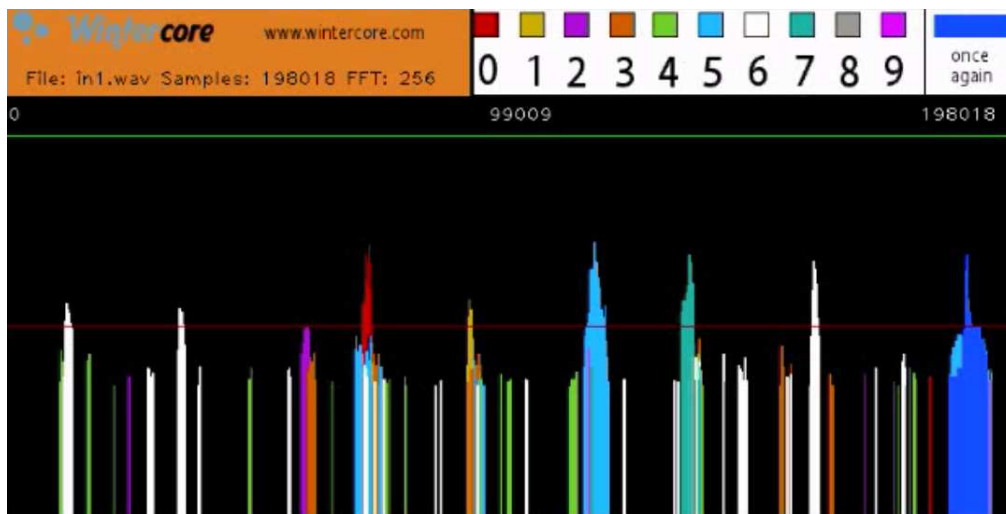


Figure 2.35: Recognition of digits in Google Audio CAPTCHA (Santamarta, 2008). In these case, the digits are 6 – 6 – 2 – 0 – 1 – 5 – 7 – 6.

2.5.5 Attacks to audio CAPTCHAs

Santamarta (2008) showed that the Google audio CAPTCHA could be broken using very basic methods. In particular, it was possible to detect the characteristic wave and FFT of each digit spoken (see Figure 2.34), and because they were played with higher volume than the background noise, it was possible to distinguish them (see Figure 2.35).

Another successful attack on it was based on well-known ML algorithms, in particular using AdaBoost, SVM, and kNN for both letter and digit recognition (Tam et al., 2008). They used a static window size, and train on well-known features for NLP, in particular, twelve MFCCs and twelfth-order spectral and cepstral coefficients from PLP and RASTA-PLP. The functions to do so are included in the Voicebox package. They are able to break Google

Audio CAPTCHA with a 67% success rate, Digg with a 71% success rate and reCAPTCHA with a 45% success rate. The approach they use is thus quite straightforward for speech recognition specialists, and it is a bit surprising that the CAPTCHAs of these important companies have not been tested for similar attacks.

With the recent advances in DL, there has been an important improvement in speech recognition. Thus the gap between humans and machines has got thinner (Hannun et al., 2014). The idea of using speech recognition as the base for an audio CAPTCHA might not be useful any-more. This leaves fewer alternatives for vision-impaired persons.

2.5.6 Attacks to “behavioural” CAPTCHAs

These currently constitute one of the main trends in the CAPTCHA world. That does not mean that they are more secure than the alternatives. Even though “No CAPTCHA reCAPTCHA” used extreme obfuscation code for their Java Script client code and client-server communications, within a week from its release it was broken²³. The information on this research was not available for some months as per request from Google²⁴.

This reverse-engineering allowed to understand the local metrics that Google’s reCAPTCHA was using. Among the metrics used were the list of plug-ins installed in the browser, the user-agent string, screen resolution, execution time, time-zone, number of user actions inside the CAPTCHA iframe, the behaviour of some CSS rules and functions that are typically browser-specific, whether the browser renders canvas elements, etc.²⁵

Other security flaws of “No CAPTCHA reCAPTCHA” were also soon pointed out (Homakov, 2014). Even though the “No CAPTCHA re-

²³The details can be found <https://github.com/neuroradiology/InsideReCaptcha>.

²⁴The author of the reverse-engineering posted that “I received an email from Google requesting the following: “The code you reversed is used to protect many sites’ registration process including Google and many others. We are concerned that having your code and analysis publicly available will make it easier to build registration automation tools [...] This is why we kindly ask you to temporarily remove it [...]” I removed the GitHub repository for now. [...] Google also proposed me to come visit them in their offices to discuss about my work.” Comment located at https://www.reddit.com/r/netsec/comments/2or9e3/reverseengineering_the_new_captchaless_recaptcha/cmqa04/).

²⁵Information taken from the GitHub “neuroradiology/InsideReCaptcha” repository at <https://github.com/neuroradiology/InsideReCaptcha>

CAPTCHA” increases usability in certain scenarios, it does not seem to increase security in any case, and indeed presents new potential flaws.

Later on, researchers published an easy to implement attack both on the “behaviour” client-side metrics and on the image CAPTCHA that Google sometimes presents to the users (Sivakorn et al., 2016a). This attack is based on design flaws of “No CAPTCHA reCAPTCHA” and in readily available image classification APIs and libraries that use DL. It breaks “No CAPTCHA reCAPTCHA” with a 70% success rate, and has a 83% success rate against the Facebook image CAPTCHA that is “shown to users when they send messages to other users that contain suspicious URLs” (Sivakorn et al., 2016b). The authors of the attack report that “Following our disclosure, reCaptcha altered the safeguards and the risk analysis process to mitigate our large-scale token harvesting attacks. They also removed the solution flexibility and sample image from the image CAPTCHA for reducing the attack’s accuracy”. (Sivakorn et al., 2016b). But it is understood from this that even though their particular attack might be *less* successful now, variations of it can still be able to bypass it.

NuCaptcha uses a combination of what they call a “behaviour analysis system to monitor all interactions on the platform” to modify the difficulty of the CAPTCHA challenge²⁶, relying in an improved OCR/text CAPTCHA that incorporated moving characters. Note that this did not prevent the attack by Bursztein (2012) that successfully breaks it.

Summary We have presented the most well-known and influencing attacks to several types of CAPTCHAs. This is by no means a complete list of attacks, in fact there are many more attacks published on particular schemes and subtypes, but in this list we include the attacks that influenced most the evolution of the different subtypes of CAPTCHAs.

As a summary, we present Table 2.1, in which we list each CAPTCHA and type mentioned, along with the mentioned attacks against them. The last column represents whether the attack solves the base *AI-hard* problem, or at least it is a step in that direction. As can be seen, most attacks cannot be classified as such, as they are side-channel attacks that decode enough information as to bypass the CAPTCHA.

²⁶From <http://www.nucaptcha.com/security-features>, retrieved on November 2016.

Table 2.1: Some of the main attacks on well-known CAPTCHAs.

Category	CAPTCHA	Attack method	Solves / improves base problem
OCR	Gimpy	Edge detection, likelihood of bi-grams or words, dictionary attack (Mori and Malik, 2003)	no
OCR	MSN/Hotmail, Register.com, Yahoo!, Ticketmaster, Google	Segmentation is trivial, NNs for character recognition (Chellapilla and Simard, 2005)	no
OCR	Captchaservice	Pixel counting, flood-filling, vertical pixel counting, dictionary look-up (Yan and Ahmad, 2007)	no
OCR	MSN	Flood-filling, vertical pixel counting, arc removal algorithm (Yan and Ahmad, 2008)	no
OCR	Megaupload	Identifying and merging character components (El Ahmad et al., 2010)	no
OCR	Baidu, eBay, reCAPTCHA, Yahoo!, Wikipedia	ML to do both character segmentation and recognition, scoring possible segmentation decisions (Bursztein et al., 2011, 2014)	possibly
OCR	reCAPTCHA, Yahoo!, Baidu, Wikipedia, QQ, Microsoft, Amazon, Taobao, Sina, Ebay	Log-Gabor filters for segmentation, kNN to choose most probable word (Gao et al., 2016)	possibly
OCR	reCAPTCHA	DNN (Goodfellow et al., 2013)	yes

OCR/ 3D	Teabag, 3dCAPTCHA, Super-CAPTCHA	Distinguishing triangles for segmentation and recognition (Nguyen et al., 2011, 2014b)	no
OCR/ animated	several ²⁷	Pixel delay map to detect regions not moving, catching line to detect characters displayed at a particular height (Nguyen et al., 2012a)	no
OCR/ animated	Hello CAPTCHA	Using different frames from the animation and detecting the different elements (Nguyen et al., 2012b)	no
OCR/ animated	NuCaptcha	Using SIFT to find interesting <i>regions</i> , isolating the most “interesting” objects and detecting the most interesting 50 frames (Bursztein, 2012)	no
OCR/ Math	QRBGS	Skewed answer distribution (Hernandez-Castro and Ribagorda, 2010)	no
OCR	Captcha2	Pixel counting (Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda, 2010)	no
Text/ Logic	Text CAPTCHA	Easy to parse (Hernandez-Castro et al., 2011)	no
Text/ Semantic	Egglue	Flawled grading routine (Hernandez-Castro et al., 2011)	no

²⁷SiteBlackBox, Animierte CAPTCHA, Sandbox, CharitelBilling, iCaptcha, Atlantis, AmourAngels, SnapPages, Bayu, BulletDrive, CAPTCHANIM, Dracon CAPTCHA, KillBot Professional.

Image/ Classifica- tion	HumanAuth	Small DDBB, watermark does not prevent recognition using random metrics and ML (Hernandez-Castro, Ribagorda and Saez, 2010, Fritsch et al., 2010)	no
Image/ Classifica- tion	ASIRRA	Improved colour and texture recognition (Golle, 2009)	yes ²⁸
Image/ Under- standing	IMAGINATION	Developed an algorithm to find candidates for image boundaries (Zhu et al., 2010a)	no
Image/ Under- standing	What's Up	Can be broken by brute-force & learning	no
Image/ Face/ Iden- tification	Facebook	Using ML, training data through social engineering (Polakis et al., 2012)	no
Game	Are You a Human	Simple heuristics to detect the background and the foreground objects, brute-force learning (Mohamed et al., 2013)	no
Audio	Google Audio	Detect the characteristic wave and FFT of each digit spoken (Santamarta, 2008)	no
Audio	Google Audio, Digg Audio, reCAPTCHA Audio	Using a static window size, and training AdaBoost, SVM, and kNN on well-known features for NLP (twelve MFCCs and twelfth-order spectral and cepstral coefficients from PLP and RASTA-PLP) (Tam et al., 2008)	possibly

²⁸Golle's approach is an advance in the state-of-the-art of the ability to automatic classify images containing cats vs. images containing dogs.

Image/ Face/ Iden- tification	FR- CAPTCHA	Extracts and matches four features per face (Gao et al., 2015)	no ²⁹
Image/ Face/ Fake vs. Real	FaceD CAPTCHA	Edge detection and SVM classification, using as fea- tures color, texture, LBP, SIFT and Laws' Masks	no
<i>Behavioural</i>	No- CAPTCHA reCAPTCHA	Reverse-engineering their obfuscation techniques ³⁰	no
<i>Behavioural</i>	No- CAPTCHA reCAPTCHA	Test of <i>behavioural</i> metrics, using DL for image classifica- tion (Sivakorn et al., 2016a)	no

2.6 General attacks against CAPTCHAs

Most of the attacks against CAPTCHAs that we have introduced so far are tailored to the specific type of challenge presented. This has been typically the case in attacks against OCR CAPTCHA, text CAPTCHA, the first attacks against audio CAPTCHA, and also many of the attacks against image-based CAPTCHA.

Here, we present two additional attacks that have the potential to be general, that is, can be applied to many types of CAPTCHAs. These attacks do not necessarily imply that CAPTCHAs based on the *AI-hard* paradigm are finished, nor that all CAPTCHAs proposals can be solved using them, but they are a common threat that many new CAPTCHA proposals need to tackle.

²⁹Because of the low-number of features extracted, this approach cannot be scaled to face identification with a large DDBB of faces.

³⁰The details can be found at GitHub “neuroradiology/InsideReCaptcha” repository at <https://github.com/neuroradiology/InsideReCaptcha>.

2.6.1 Deep Learning and game, audio and image-based CAPTCHAs

NNs had been used successfully for character recognition (LeCun et al., 1989). But the success of NNs for OCR and CV in general was limited, and not as good as some other CV/ML techniques. In the 2000s, some proposals were made to increase the recognition abilities using more advanced schemes. Some of them included full training of complex systems through Graph Transformer Networks (Lecun et al., 1998), and later, the improvement of these results using several NNs through Multi-Column DNNs (Ciregan et al., 2012).

But it was in 2012 when Alex Krizhevsky was able achieved a milestone benchmark using DCNNs (Deep Convolutional Neural Networks). He used new training procedures, including regularization through “dropout” and the *ReLU* activation function, and also used GPUs that allowed him for *inexpensive* parallel computation (Krizhevsky et al., 2012).

NNs were not new, but for the first time it was possible to efficiently train a NN with many layers. DL benefits from three main aspects: a) the large increase in the sizes of the training sets, both labelled and unlabelled, thanks to several efforts, crowd sourcing (like Amazon Turk), and the availability of the Internet; b) the increase in parallel computing power thanks to the evolution of the Graphic Processing Units (GPUs), initially created for 3D gaming, also able to perform other highly parallel computations, and c) the ability to leverage the two using improved architectures and improved training methods.

Krizhevsky et al. results’ in the ImageNet challenge meant that since then, most CV research has been done using DNNs, and the results achieved have been much better than by any previous method. Since then, the research in DL has experimented an explosion that has rendered great progress in locating objects, interpreting images, reading text, recognizing speech, and many other fields and applications.

In 2013, Google researchers used DCNNs to read street numbers from Google Street View. They also applied the same NN to the hardest version of their own reCAPTCHA. Their approach was able to break it with 99% success rate (Goodfellow et al., 2013).

Unfortunately, the consequence was that Shet, the product manager of reCAPTCHA, concluded from this that “This shows that the act of typing

in the answer to a distorted image should not be the only factor when it comes to determining a human versus a machine” (Shet, 2014b), redirecting their product towards the so-called “*behavioural*” analysis.

Currently, the same types of DCNNs that have been successful at the ImageNet competition have also been successful at breaking image classification CAPTCHAs like the one used by Google in their “No CAPTCHA reCAPTCHA” system as well as the Facebook image classification CAPTCHA (Sivakorn et al., 2016a).

When analysing a CAPTCHA, seldom we will be able to gather large amounts of labelled data. For that reason is more interesting that, in some other cases, we will be able to use a DNN to learn high-level *features* in an unsupervised way (Larsen et al., 2015). Figure 2.36 shows an example of a DNN that has learned facial features in a completely unsupervised way. In this case the DNN is composed of a Variational Auto Encoder (VAE) mixed with a Generative Adversarial Network (GAN). This architecture is able to learn high-level representations (features) unsupervised. From the third column onwards, each column represents the generation performed by the network when that attribute was added to the internal high-level representation of the VAE, from the original image (first column). Once a NN has such high-level representation, we can use it either with a DNN or with more typical ML algorithms. We do that by feeding the activation of these features to a NN layer or other ML algorithm for further classification. This opens exciting new possibilities for automatic extraction of CAPTCHA parameter creation attributes.

Game-like CAPTCHAs provide a different kind of interaction that tries to mimic simple games. Recently, there has been significant advances in the ability of DNNs to learn to play games in their own by reinforcement learning, as learning to beat a series of *Atari 2600* games just from pixels (Mnih et al., 2015). At a higher level, computers have been able to learn to master the ancient game of Go at top human level (Silver et al., 2016), something that was considered extremely difficult just a decade ago.

Given these results, and even though it remains to be determined to which level this is a possible compromise for future game-like CAPTCHAs, it seems clear that the future of these alternatives cannot have them rely on their game & control part for their security. The target of the game has to be itself hard for computers. And even in this case, the recent advances like those of Silver et al. (2016) present a difficult scenario for game-like CAPTCHA

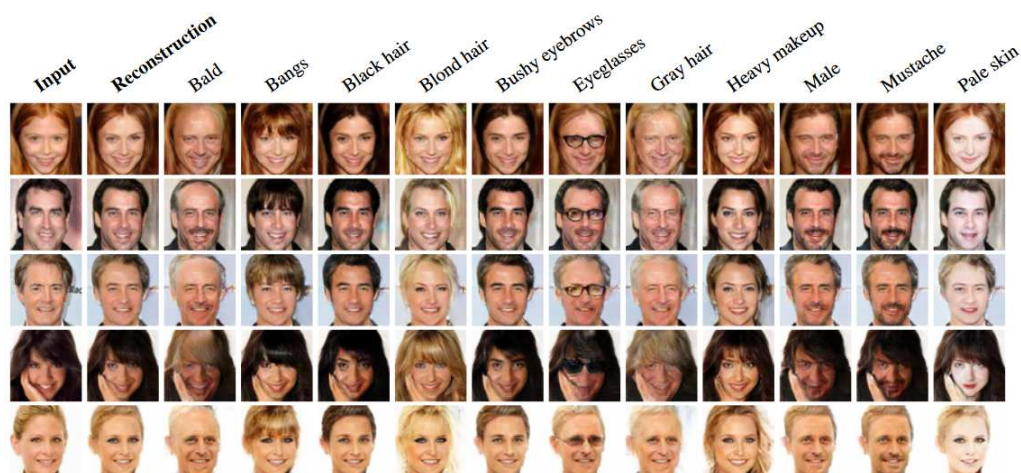


Figure 2.36: Example of a DNN that has learned abstract representations equivalent to facial features, using unsupervised training (Larsen et al., 2015). Unsupervised learning of features opens the door to learn using CAPTCHAs as sources of content, when not as oracles.

designers.

We can distinguish three issues related to the use of DNNs to learn to solve CAPTCHAs.

Adversarial Learning Generative Adversarial Networks (GANs) are NNs trained to generate data mimicking a given distribution, in an adversarial manner (Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville and Bengio, 2014). They can be trained along with discriminative NNs in an alternating fashion, as to create examples that provoke mistakes in the discriminative NNs.

It is also straightforward to alter images (Goodfellow, Shlens and Szegedy, 2014) and natural language (Jia and Liang, 2017) to make DCNNs miss-classify them, while they are still classifiable by a human. It has been shown that “adversarial examples generalise well to different [DNNs] architectures and initializations” (Goodfellow, Shlens and Szegedy, 2014, Szegedy et al., 2013), meaning that they are independent of the training data and the network topology. It is also possible to create adversarial examples that work under different transformations.

It is possible to train a NN considering one or several types of adversarial examples, so the NN is resilient to them. Unfortunately, this

does not protect the network from other kinds of adversarial examples, which makes NNs inherently vulnerable to this attack.

This has led to the creation of DeepCAPTCHA by Osadchy et al. (2016), a new CAPTCHA proposal based in adversarial learning, in which their authors introduce in their challenges adversarial perturbations that are resistant to removal attempts. This line of research based on limitations of DNNs when compared to their human counterparts might be promising for the development of new CAPTCHAs resistant to DL.

Suitability NNs are more successful than other ML approaches when the amount of examples (training data) is large, there is a structure on the data, and is not of categorical nature (not hundreds of possibly related or unrelated variables). A typical example in which NNs are typically better than other ML methods is image recognition. A typical example in which other ML methods might still achieve better results is classification tasks based on categorical data.

Training set size Another important aspect in order for DL to be successful is that there is a very large number of training examples available, or a way to create them realistically through transformations over current data. Not all data used for training has to be labelled, in fact, DNNs can typically take advantage of a first phase of unsupervised learning followed by a phase of gradient descent with supervised learning. In any case, the number of training examples has to be in proportion to the number of parameters of the networks, that can run in the hundreds of millions.

This is not always the case: sometimes, it is possible to retrain the last layers of a pre-trained DNN in order to adapt it to a slightly different use than its original one. This normally leads to good results, but not as good as if the network was fully trained with appropriate examples.

These three conditions mean that DL is best suited for recognition tasks as the ones in which game, audio and image-based CAPTCHAs are based (including OCR/text). To date, this covers most of the CAPTCHA proposals so far. But there are other proposals based on different methods that, at least in principle, might not be as well suited for DL.

2.6.2 Oracle attacks

CAPTCHAs also add a potential vulnerability in the sense that, even if a system is able to pass then with a certain low success rate $x\%$, maybe using some heuristics or just by chance, this would allow to gain additional training data, as every time a challenge is passed the *bot* learns the ground-truth value of the elements involved in that challenge. This can allow for a better trained system with a bigger success rate $x'\% > x\%$ (Stark et al., 2015). To prevent this, some authors propose to take some of the images out of the verification mechanism, and more so, to use them as *traps* if they would have rendered not correct a previous solution - that way, confusing the possible bot about the correct image classification (Kwon and Cha, 2016). This approach might be useful to prevent further training on the CAPTCHA data, but unfortunately has flaws that make it useless (Hernández-Castro et al., 2017).

2.6.3 Relay attacks

After the first years of evolution of CAPTCHAs, a new threat appeared: third-party CAPTCHA solving services. These were initially service providers that based their human work-force in low-wage countries (Danchev, 2008). They provide their services through an API, so the whole abuse process could be (semi-)automated. Nowadays they rely not only on workers in low-wage countries, for instance, Amazon Turk also has a lot of spam-related *job* offers (HITs). During an analysis in 2010, up to 40% of HITs in Amazon Turk were spam-related (Ipeirotis et al., 2010). Other ways are through on-line solvers that offer some revenue, trojan horses that display CAPTCHA challenges (Chuley, 2007), or phishing attacks (Kang and Xiang, 2010) (for a thread model on this, see section 2.2.1).

In certain scenarios, this can be the most economic way of attacking a CAPTCHA. Just a few CAPTCHA proposals try to address this new threat with very limited success (Baird and Bentley, 2005, Halprin, 2007, Mitra et al., 2009a, Longe, 2010, Onwudebelu and Ugwuoke, 2012, Mohamed et al., 2013).

2.7 New proposed CAPTCHA types

In this section we present some new and original CAPTCHA proposals. Some of them pertain to the design variants commented before, but use the already seen challenges in an original way or try to increase their security with some additions. These proposals distinguish themselves because of their novelty. This also means that known attacks against other CAPTCHAs do not affect these ones, as each one of them presents an original mechanism that has not been used before.

2.7.1 CAPTCHAs based on empathy

The authors of the Civil Rights CAPTCHA (*CRC* from now on) use the human ability to feel empathy to strengthen a typical OCR/text CAPTCHA. The *CRC* picks up a Civil Rights news from its database (DB) and then uses Securimage to create three images containing words depicting possible emotions related to the text. These images contain words describing feelings (for instance, "upset", "happy" and "furious"). The user has to write down the correct one based on the emotions originated from the news headline presented to her. Thus, the *CRC* is based on the human ability to show empathy after being presented with a news excerpt, typically containing some news about Human Rights and/or Civil Rights around the world.

The Civil Rights CAPTCHA uses a traditional OCR CAPTCHA, to which there are known attacks, but it is further secured by the detection of empathy. There is currently no ML algorithm that tries to simulate empathy. There are ML approaches to understanding the human languages (NLP, Natural Language Processing), but they focus on detecting the feelings and opinions of the writer through the use of adjectives and adverbs. They do not focus on the induced feeling on the reader. This proposal is further analysed in Chapter 4.

2.7.2 Enhanced image-classification CAPTCHAs

There are several CV algorithms able to locate faces in pictures and extract information from them. Face identification is a very well known CV problem that has been studied for long. These reasons allow ML practitioners to be

aware of the limits of these algorithms when confronted with difficult input. This includes partially occluded faces, facial expressions, faces looked not in front but with some angle, strange lightning, etc. There have been proposals to use these limitations in the creation of a CAPTCHA. By definition, if this is possible and the CAPTCHA is well defined, this CAPTCHA should be secure.

Several proposals have used the problem of gender recognition in pictures of faces (Kim et al., 2014, Sim et al., 2014, Gosschalk and Ford, 2016, Schryen et al., 2016). The only one that has been put into production is FunCAPTCHA (Gosschalk and Ford, 2016). Thus, it is interesting to check whether they attain the said security level. We analyse FunCAPTCHA in Chapter 5.

2.7.3 Puzzle CAPTCHAs

A recent game-like CAPTCHA proposal are puzzle CAPTCHAs. In them, an image is divided into parts, of which at least one is missing. The user has to place the missing parts correctly to solve it. Other variants have the parts shuffled and the user has to reorder them. In any case, the user has to reconstruct the original image.

These proposals are fundamentally different, as there is no need to recognise and interpret the different elements. Also, the puzzle pieces are not differentiable elements by themselves, that is, a puzzle piece is not recognised by our visual system as a ball, a lamp or a door; it is nothing more than a puzzle piece. Thus object detection does not serve a purpose here. These proposals are also different from image classification CAPTCHAs, as the only classification relevant here is if the image is correct (as the original) or one of the many incorrect possibilities, with the puzzle pieces wrongly placed.

There are many attacks on image classification CAPTCHAs and other image-based CAPTCHAs, but none on puzzle CAPTCHAs. As explained, these pose a fundamentally different problem, in which we are not interested in interpreting the images, but on restoring it to its original state. Some of these puzzle CAPTCHAs are Capy³¹, KeyCAPTCHA³² and Garb³³.

³¹It can be tested at <https://www.capy.me/account/signup/>, retrieved on November 2016.

³²It can be found at <https://www.keycaptcha.com/>.

³³It can be accessed at <https://ky.wordpress.org/plugins/captcha-garb/>

We analyse them in Chapter 3.

2.8 Summary

CAPTCHAs remain a generic security mechanism to prevent automated attacks in most scenarios. Unfortunately, to date no CAPTCHA proposal has been able to provide their stated security target - protecting from automatic abuse for a long period of time. It is not straightforward to design a CAPTCHA. In particular, the *hard-AI-paradigm* introduced by Naor (1996) and expanded by Ahn et al. (2003) might not be as promising as originally, because:

- We do not know what is a *hard-AI-problem*: “(...) in AI [we] would require a precise definition of a hard AI problem, and it isn’t clear how to create one. ‘We’ve decided not to follow that route’ Blum says. Instead, in designing their CAPTCHAs, researchers are using problems that AI researchers believe to be hard” (Robinson, 2001).
- Even if a problem is *hard for AI* and remains so for years, we are not sure about how to transfer such hypothetical *hardness* to the problem subset that a CAPTCHA produces. If we fail in doing so, our proposal might be susceptible to side-channel attacks. Note that most CAPTCHA attacks to date have been side-channel, while extremely few of them had advanced the state-of-the-art of AI, as originally intended by Ahn et al. (2003).
- It remains to be seen how to cope with the recent advances in ML thanks to DL. Proposals based on limitations of DL, like DeepCAPTCHA (Osadchy et al., 2016), might be promising.
- CAPTCHAs have very stringent design requirements both regarding their usability (user-friendliness, easy of use, time spent to solve them) and their security (automated success below 0.1%, resilient to oracle attacks, resistant to third-party human solvers, ...).

The previous attacks presented in section 2.5 are not directly applicable to some of the new CAPTCHAs types presented in section 2.7, as

installation/, retrieved on November 2016.

these are either based on fundamental problems that are new, or at least use well-known problems in a new way. For these reasons, we find these proposals to be both original and interesting as to check their security level. By analysing their security, we will advance the state of the art in CAPTCHA security, especially in these new domains of CAPTCHA design. We will try to find whether they fall or not for variations of problems common in other CAPTCHA designs.

While we present these three new security analysis, we will also check if there are general patterns or ideas that can be used for testing for a basic level of security of a new CAPTCHA design. If so, we might be able to create a procedure that uses these patterns in order to test for a basic level of security.

Chapter 3

Case Study: Capy and other puzzle CAPTCHAs

One of the aims of this dissertation is to analyse CAPTCHA proposals that have not been previously studied. Puzzle-like CAPTCHAs are a relatively recent novelty. Contrary to other image-based CAPTCHAs, they are not based on the problems of image classification nor object recognition. There are no previous security studies about puzzle CAPTCHAs, so their strength remains unknown. Two commercial solutions exist that use puzzle CAPTCHAs and have reached some levels of success.

There are several different puzzle CAPTCHAs. Among them, we have chosen three that are a representative subset of them. Two of them are commercial proposals that are based on the same idea of restoring an original image that has one or several pieces lacking. They do it using quite different implementation details that might affect their security, so both are worth to be analysed. The third one that we choose to study is based on the idea of shuffling image parts.

In this chapter we describe these three puzzle CAPTCHAs, analysing their security and focusing on their possible flaws. We also show an attack against them. First, we focus on the Capy CAPTCHA. This analysis shows potential flaws. Then, we present two other puzzle CAPTCHAs and explain the results of an attack on them. After, we show several potential mitigation measures. Section 3.9 finishes the chapter presenting a summary of the findings.

The methodology used during their security analysis as well as the results produced are used as input for the design of the BASECASS methodology, that is explained in chapter 6.

3.1 Cappy CAPTCHA description

The Cappy CAPTCHA was started in 2010 as an academic research project at Kyoto University, designed by a PhD in Computer Science, who turned it into a company in 2012. It has been well praised both through awards and in the press: awarded "Best Demonstration" at IEEE CCNC International Conference in Las Vegas 2010, first prize at MIT Entrepreneur and Innovation Pitch Competition 2012, "Top Startup" winner of the TiE 50 2013, first prize at the Infinity Ventures Summit Kyoto 2013, first prize at Technology & Business Plan Contest in Kyoto 2013. Among others, it has been featured in IEEE Spectrum Magazine 2011.

Cappy CAPTCHA has got the public attention, and has been considered quite good by several panels of experts that analysed it. According to press¹, Cappy has secured US\$1 million in investments, and is currently charging around US\$0.001 per challenge served².

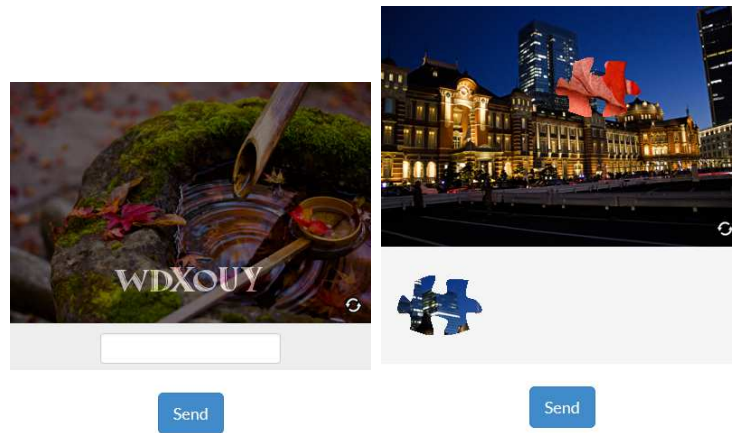
Cappy CAPTCHA offers several types of CAPTCHA in their webpage that basically fall into two categories: puzzle CAPTCHAs and text CAPTCHAs (as shown in Figure 3.1). We will focus on the puzzle CAPTCHA, which is the truly innovative proposal. Much work has been done previously on OCR/text CAPTCHAs, and most of these proposals if not all can be considered either susceptible to attack or too difficult to solve even for humans (Bursztein et al., 2014, Gao et al., 2016). In the rest of this chapter, when we refer to the Cappy CAPTCHA, we will implicitly mean the puzzle variant.

Cappy works by creating a simple puzzle in which there is only one puzzle piece (see Figure 3.1). The user has to drag and drop the puzzle piece into the correct location within the challenge image.

The Cappy designers claim to have put some effort into its security. For instance, the puzzle void within the challenge image is not filled with

¹From *e27 technopreneurship* news source, retrieved from <http://e27.co/sick-captchas-cappy-makes-game-20140619/> on 20th June 2014.

²As of August 2017.



(a) Text recognition HIP.

(b) Puzzle CAPTCHA.

Figure 3.1: The two different challenge types offered by Capy.

a random color; instead it is filled with a portion from another image and sometimes from the same image. As we will see later in more detail, Capy sends to its server not only the final position of the puzzle piece (where we drop it, within the challenge image), but the log of the whole drag movement through the screen. This would allow them to further examine the complete pointer movement log in their servers.

In the production version presented in their web-page, only one puzzle piece is required per image. In a video presentation of their idea, they show the possibility of more than a single puzzle piece per image. We will focus on the production version, while discussing later whether the found weaknesses extend or not to a possible multi-puzzle-piece version.

3.2 Capy CAPTCHA analysis

Capy presents an image of 400×267 pixels and a puzzle piece of approximately 76×87 pixels - this size might vary as the puzzle piece shape can change.

An image CAPTCHA like this, in which the solution space is roughly $400 - 76 \times 267 - 87 = 58.320$ possible answers (possible positions), and with no further information, will provide a security of $\frac{1}{58.320} = 0,0017\%$ against a random (brute force) attack. This result is pretty good and strong enough for a CAPTCHA. The CAPTCHA design goal is that automatic scripts should

not be more successful than 0,01% (Chellapilla et al., 2005b) or the least restrictive 0,6% (Zhu et al., 2010a).

We have used an HTTP protocol analysis tool to understand and replicate the communications of the JavaScript client scripts with the Capy CAPTCHA server. In this phase, we learned that the communication protocol sends all the positions through which the piece travels while being dragged. They are encoded in base 32, adding the character *x* for separation. For example, one possible solution string would be *ax8exax84xax7qkx7gkx76kx.....ixax1ixkx18kx*, where $(a, 8e) \dots (18, k)$ are the base-32 encoded positions (encoded as displacements from the initial position of the puzzle piece). This information would allow Capy to further examine the solution sent to their servers, detecting whether this pointer (mouse, finger) movement corresponds to a human, and thus enhance the human/bot discrimination.

3.3 Capy CAPTCHA design flaws

Soon after starting using this CAPTCHA, the first important design flaw was evident: the puzzle piece only moves in discrete 10-pixels steps. This means a brute force attack would have a chance of $\frac{400-76}{10} \times \frac{1}{267-87} \approx \frac{1}{32 \times 18} \approx 0,173\%$ success against it. A bit too high to put it into production. This is a weak design idea that makes the CAPTCHA much more susceptible to attacks. It is in itself not an irresolvable problem for the idea behind this CAPTCHA, as can be solved with bigger images, several puzzle pieces, and/or smaller step increments (5 pixels), among others.

The major problem with this design decision is that it opens the door to other attacks in which there is a noticeable difference between a correct solution and a solution 10 pixels away from it (and not just 1 pixel away).

Capy sends to its server the mouse movement log. Unfortunately, after using the CAPTCHA, we detected that Capy discards most of the movement log. We run some experiments in which we discovered that Capy CAPTCHA does not accept to send only the final puzzle piece position, returning *False* (test not passed) if we just sent it. Also, we noticed that reducing the drag log size (*jumping* over positions) did not seem to affect its marking.

At the end, we learned that sending just two positions, the initial one (always the same) and the final position (where the puzzle piece goes) got *True* (test passed) replies every time the solution position was correct. We were surprised that Capy was not using this information to further discriminate humans and bots, maybe using some ML clustering algorithm. We think that not trying to take advantage of this information is also a minor weakness in its design.

3.4 Foundations of the side-channel attack

One property of the correct solutions to the challenges of this CAPTCHA is that the resulting images are *more natural*, in the sense that both colours and shapes are more *continuous*. The puzzle piece target inside the challenge image *has to be* visually disruptive for the human eye to be able to easily locate it. When it is covered with the puzzle piece, the resulting "original" image is better in terms of *continuity* - shapes around the figure are more continuous, as are colours and textures (shape repetitions).

At this point of the security analysis, we had not found design flaws that seemed strong enough as to lead to a successful attack. Thus, we proceed to design metrics that could allow us to extract information from the challenges, and possibly characterize their correct solutions on a number of cases.

Among the different metrics we decided to try, we thought about how lossy compression algorithms would process the different challenges. As explained before, the correct solution to the challenges is more *natural*, that is, typically uses less colours and textures, and more texture and colour repetitions, than the image with the puzzle piece target not covered. This property of the correct solution leads us to think in creating a metric based on the JPEG compression algorithm. JPEG can compress any image, but will do its best on photographs of realistic scenes with smooth variations of tone and color.

The JPEG image compression Wallace (1992) works roughly by dividing the image in squares (for example, blocks of 16 pixels in each direction, in the case of 4:2:0 chroma sub-sampling) and computing their discrete cosine transform (DCT). After the data is divided in blocks of 8×8 pixels, the DCT converts the spatial image representation into a frequency

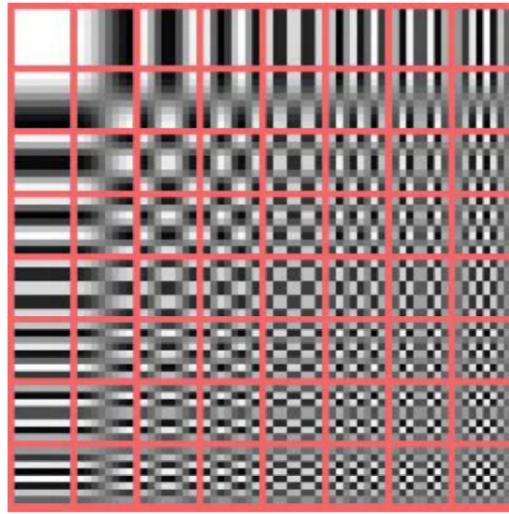


Figure 3.2: Representation of plane-waves corresponding to each Discrete Cosine Transform (DCT) coefficient. Low-order terms represent the average colour in the block, while the successive higher-order terms represent the strength of more and more rapid changes across the width and height of the block.

map: the low-order terms represent the average colour in the block, while the successive higher-order terms represent the strength of the more and more rapid changes across the width and height of the block (see Figure 3.2). The amplitudes of the frequency components are quantized, that is, represented with lower accuracy. High frequencies (sharp changes) are typically discarded, and further reductions in fidelity are done. The precise quantization table (integer divisors) are embedded in a table that will have higher values if we want higher compression ratios (lower quality). The resulting bit-stream is further compressed using a lossless algorithm.

The important aspect for us is that for JPEG, both light pattern regularity (texture) and colour pattern regularity play a major role in the size of the resulting compressed image. For this reason, the image size once compressed will probably be a relevant metric.

We first thought about using this JPEG-derived metric along with other several ones to try to find patterns that would allow us to distinguish the correct solutions. This was not necessary, as when we were testing the different metrics, we noticed that this metric alone seemed to have quite a good performance.

3.5 Side-channel attack

In order to test the real-life usefulness of this metric, we conceived two attacks. The first one, named **basic attack**, tries to find the correct answer by placing the puzzle piece in all possible locations, and for each resulting image, computing its JPEG size. The image that has a smaller size will be considered the correct one, and thus that position of the puzzle piece will be sent to the Capy server as our answer.

The second one, called **modal attack**, runs the JPEG compression algorithm on the resulting answer images with different quality settings (from 10 to 100 - the maximum). We then let each quality setting choose one *correct* solution, *voting* for it. The most voted solution among the different quality setting is the chosen one, and sent for verification to the server.

The hypothesis here is that whenever the smallest size for the JPEG does not correspond to the correct solution, for a particular quality factor, another set of compressions with a different quality factor will not pick the same wrong solution. That is, the idea behind this attack is to check if, when the JPEG compression result is wrong, it does fail consistently (picking up most of the time the same wrong solution) or not. We expected to obtain better results than the **basic attack**. We proceed to test this hypothesis experimentally.

Our initial estimation while designing this attack was that, in a good case scenario, using just this JPEG-size discrimination, we were going to be able to break Capy CAPTCHA with an estimated success rate of over 3% to maybe 5%. We thought that several problems, like partial puzzle piece overlapping (thus reducing the image size with JPEG compression), small size variation (images in which the image chosen to fill in the puzzle void piece was already in colour and texture harmony with the background) and others would prevent this attack from getting a better result. We planned on this to be a first stage of an attack, later improved with some additional information extracted from the images.

The attack might be affected by the compression quality chosen for the JPEG algorithm, so we tried a grid search through all compression levels in increments of 10.

3.6 Experimental results

Next, we describe the results of the two proposed attacks explained in the previous section along with the reasons for the attacks success or fail in solving some of the challenges during the experiments³.

3.6.1 Basic attack results

The file size of the image compressed using JPEG is dependant on the quality setting, that in turn affects the lossy compression algorithm. At first we performed our attack with different compression (quality) settings to discover for which one it seemed to have a better success rate. Since downloading a Capy challenge takes on average 4.33 seconds, we performed this test only for 200 challenges in each quality setting. We carried out an exhaustive search, using all quality settings from 10 to 100 in steps of 10.

After 4:42 hours, we obtained the results shown in Figure 3.3, that depicts the success rate of the attack (number of correctly solved challenges, in percentage) depending on the JPEG compression quality rate (setting from 10 to 100, the maximum). The dotted line represents the linear regression estimate of the function. Even though the relation between a JPEG quality setting and the success rate is not linear, it is clear that there is a tendency to improve the success rate of the attack if we use a higher JPEG quality setting. Also, the maximum compression quality was the one able to differentiate best the correct solution, with a 61.5% success rate for these 200 experiments.

In average, the JPEG compression algorithm takes longer to compute the compressed image when the quality setting is higher. We checked that the computation requirements for a higher quality setting were not much higher than for a lower quality settings (Figure 3.4), being 3.65 secs./image the minimum average computation time, obtained for quality setting equal to 20, and 5.78 secs./image when the quality was set to 100 (mean figures), that is, a mere 58% more. Given the improvement on the success rate, from 43,5% using a compression quality of 20 to 61,5% using a compression quality of 100, it implies a 41% relative improvement, we thought the extra computing time was well worth it.

³The full results of the experiments are available at <https://github.com/carlos-havier/Capy-analysis.git>

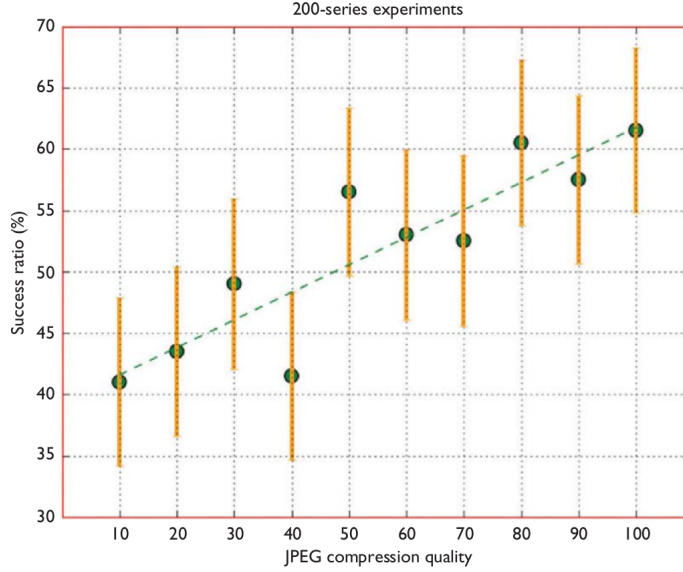


Figure 3.3: Success rate by JPEG compression quality for 200-series experiments.

We set our quality to maximum and run this attack for 1000 experiments. Our attack took 2:48 hours, as each image challenge took on average 4.33 seconds to download, and then an average of 5.79 seconds to be processed. The idea of this experiment was to get a better estimate of the real maximum success rate. In this case, our program was able to correctly solve the Capy CAPTCHA on 65.1% of the occasions. This is a result two orders of magnitude above what is needed for a CAPTCHA to be considered broken⁴ and quite extraordinary, especially for such a direct, low-cost attack.

3.6.2 Modal attack results

We wondered if, when our basic attack failed, a different JPEG quality setting would also give the same wrong result, or possibly the correct one. Thus, we focused on the mode of the frequencies with which a position was chosen, for all JPEG image qualities from 10 to 100. The idea behind this attack is that, whenever the attack will fail and pick a wrong position, this wrong position is not the same for different JPEG qualities. If that is indeed the case, then this attack might be able to show better results than our basic attack.

⁴0.6% is enough to consider a CAPTCHA broken (Zhu et al., 2010a).

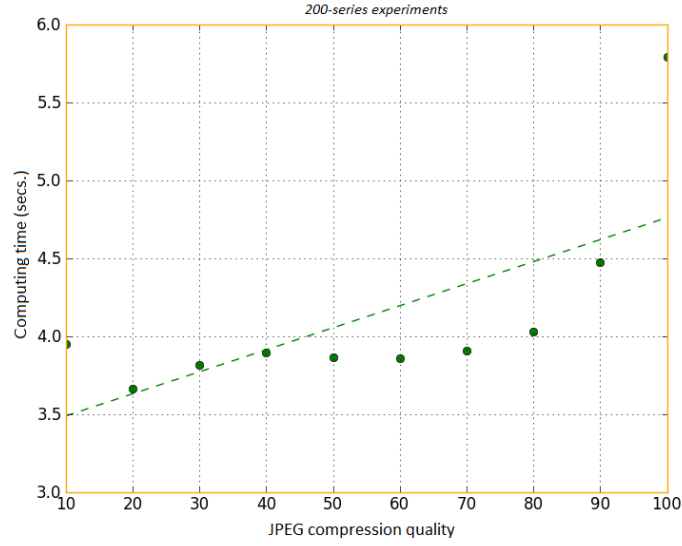


Figure 3.4: Computing time per JPEG compression quality for 200-series experiments.

As this attack was much more time consuming, 33 seconds per challenge on average, we decided to limit it to 500 experiments, that were run in different series due to network problems, actually totalling 504. The results were disappointing, as we did not improve them from the basic attack, getting a success rate of 56,5%. We noticed that, when the attack failed, the JPEG smaller size results correspond more than once to the same -wrong-position, thus rendering seemingly useless this approach.

After this unfavourable result, we wondered whether the same voting scheme, weighted by the JPEG image quality, would give any better results, given that typically, the higher the quality, the better the success rate. We decided the vote to count $\frac{quality}{10}$. We launched this new experiment for 382 challenges. The results were better, correctly solving the Capy CAPTCHA 59,9% of the times, but still inferior to the basic attack.

3.6.3 Results analysis

It is quite interesting to observe where our basic attack succeeds and where it fails, to understand why it works so well as well as its possible limitations. In Figure 3.5, we can observe a few correctly solved challenges. We have named the challenge images from top to bottom as *rock*, *wood* and *water*, *city at*

night, *sandwiches* and *lion*. The first column represents the challenge images, with the puzzle piece attached to their right. Within the challenge images, it is possible to see where the puzzle piece should go. The second column contains the image proposed by our algorithm (in this table, correctly solved).

Each row in Figure 3.5 contains thus a challenge and its proposed solution. In the first challenge, we can appreciate that the puzzle void in the image has been filled with a very detailed and colourful image, thus containing a lot of high-frequency information, difficult to compress. It is clear that if we put the puzzle piece on top of it, the image as a whole will have less high-frequency information. In the other cases we can also distinguish that the filling of the puzzle void has different colors than the rest of the image, in some cases combined with a *noisy* texture with lots of high frequency information.



Figure 3.5: Correctly solved challenges.

Figure 3.6 is more interesting as it shows some cases of failed solutions to the challenges. The *city at night* image is very interesting: the puzzle

void in the background has been filled with an almost plain colour, that happens to appear frequently in the background picture. The puzzle piece has two differentiated parts, being the bigger one also a low-detail one. Our algorithm finds that putting this piece on top of a high-detailed part of the background produces a smaller (less information) image than if we put it in its correct place. The reason for this is that the algorithm is basically *erasing* high frequency (detailed) information. In the other challenges, we appreciate similar patterns: covering high detail parts of the background picture renders smaller images.







Figure 3.6: Wrongly solved challenges.

Analysing these failures, we appreciate trends in them, depending

on properties of the background image used for each challenge, the filling of the puzzle void within the background, and also the filling of the puzzle piece. We wondered if the background image affected the success rate of our attack, so we re-analysed our results. We have analysed the four backgrounds available when we conducted these experiments. The success rates for the four backgrounds using different JPEG quality factors are depicted in Figure 3.7, while their exact values are reported in Table 3.1. The results suggest that the background indeed affects the success rate. Also, the JPEG compression settings affect differently each of the different backgrounds.

Table 3.1: Success rate per image type and JPEG quality setting, data corresponding to Figure 3.7

image	JPEG quality									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
	0,47	0,39	0,57	0,36	0,63	0,64	0,54	0,73	0,64	0,71
	0,19	0,22	0,32	0,25	0,46	0,38	0,42	0,41	0,49	0,55
	0,10	0,18	0,16	0,11	0,20	1	0,16	1	1	0,35
	0,89	0,87	0,93	0,94	0,96	0,17	0,98	0,23	0,23	0,92
mean	0,41	0,41	0,49	0,41	0,56	0,54	0,52	0,59	0,59	0,63

Note that the results in Table 3.1 and Figure 3.7 are for different number of experiments. In particular, for all the JPEG image qualities among 10 and 90, we have performed 200 experiments, whereas for a maximum image quality (100) the total number of experiments performed is 1200, so these statistics are more reliable. All in all, we obtained a minimum 10% success rate, and a maximum 100%. If we focus on the more reliable results with a quality setting of 100, the minimum was 35% and the maximum 92%. It is interesting to realize that the image background that shows the minimum success rate, is also the one that has more variability, that is, the one that mixes parts of very high and very low detail levels.

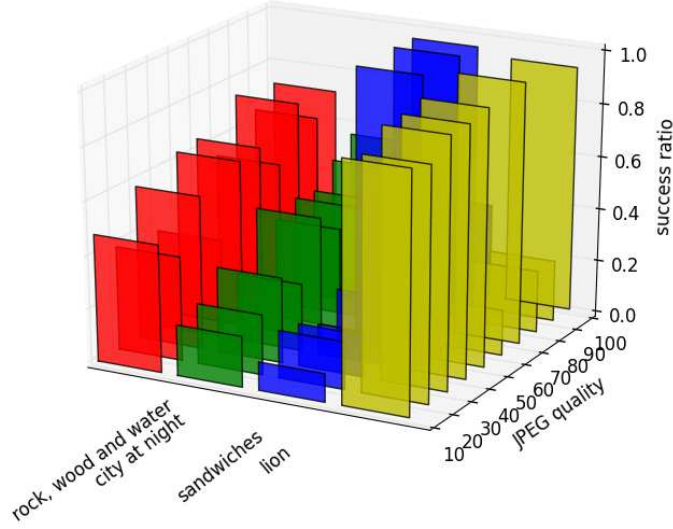


Figure 3.7: Success rate per image type and JPEG quality setting, with 200 experiments for each setting, except 1000 experiments for $q = 100$.

3.7 Other CAPTCHAs affected

There are several puzzle HIPs that use similar ideas to Capy using their own variants. Among them, two were the most interesting to us. The first one, KeyCAPTCHA, is based on typical puzzle pieces, yet it uses non-strictly delimited borders produced by anti-aliasing. The second one is Garb CAPTCHA, that presents an image divided into four pieces and shuffled. The user has to reorder them to recreate the original image. Next we describe the results obtained using our previous attack with each of these CAPTCHAs.

3.7.1 KeyCAPTCHA

KeyCAPTCHA offers several variants or versions that lie within two groups: the free one, with rectangular pieces that do not have a border, in which the user has to drag them with 1-pixel precision, and the *magnetic* ones, in which, as with Capy CAPTCHA, there is a n -pixel step movement from valid position to valid position, thus making it easier for humans to solve. Both

are shown in Figure 3.8, being the first row the *free version*.

We selected KeyCAPTCHA because it includes challenges with up to three puzzle pieces; it uses antialiasing on the borders thus making the match of the pieces not perfect, which will render an increased JPEG size when compared to the original image; and it uses white background, which compresses well in JPEG. These factors constitute a potential challenge to our algorithm.

It severely relies on Security through Obscurity to try to hide its internals. Mangled JavaScript code, along with random *iframe* names, etc., try to make it harder to analyse. Anyhow, it is easy to learn that it basically depicts several *canvas* elements, one of them containing the background image, and one for each one of the puzzle pieces.

As explained, one interesting aspect of this HIP is that the voids in the background where the puzzle pieces need to fill in is of plain *white* colour. This is a problem for our attack, as this white areas are good for producing images with a small JPEG size, so our attack will *tend to respect* them - that is, not putting any puzzle piece on top of these areas. For this reason, we explored two alternatives: 1.- using a filter, in which we only consider a valid position if the puzzle piece covers at least 90% white pixels; 2.- adding some high frequency noise to the white background pixels.

Note that using a plain white background could be regarded as a weakness for other types of attacks, for example, those relying on matching the shape of the puzzle piece to the different *voids* in the background image - this decision makes such an attack too easy.

After experimenting with KeyCAPTCHA, we were able to detect some repetitions in the objects used as background images. In particular, after the first 25 attempts, we saw only 20 different images. In the next 25, we saw 20 different ones, and among them, again 8 of the 20 we had already seen in the first batch. We can use the mark and recapture method (Seber, 1974) to estimate the image library size, stating that 20 are the ones *marked* in the first visit (different images in the first 25 attempts), 8 are the ones *captured* on the second visit, and thus the estimated population size is depicted in Equation 3.1, where N is the estimated population size, which with our data, is an estimate of 50 different objects. Again, not enough for a production CAPTCHA, given that once one background is solved, this information can be used to correctly solve it again.



Figure 3.8: Different versions of KeyCAPTCHA.

$$N = \frac{20 \times 20}{8} \quad (3.1)$$

We just wanted to validate our attack, and not KeyCAPTCHA design limitations and internals, so we decided to download 50 challenge images locally and proceed to solve them using our attack. Of these, 18 were served as 3-puzzle-pieces challenges, and the rest were 2-puzzle-pieces challenges.

We applied our attack with no modification, apart from adding noise to the white background. Of these 50 challenges, 10 of them were completely solved, that is, a 20% success rate on passing KeyCAPTCHA overall. Some

example results are shown in Figure 3.9, being the first three wrongly answered challenges, the second three partially solved ones (not counting as passing the challenge), and the third three, completely solved challenges.



Figure 3.9: Wrong, partially and completely solved challenges for Key-CAPTCHA.

3.7.2 Garb CAPTCHA

The Garb CAPTCHA is implemented as a *WordPress* plug-in that creates a CAPTCHA requesting the user to reorder image parts into their original order. We selected this puzzle CAPTCHA because it is of a different type (reordering), and it is unknown to us whether the property of the JPEG size will behave well or not in this scenario, that is, there will be a reordination with smaller a size than the correct answer, or not. Apart from this, Garb takes 1-pixel lines from the borders from each puzzle piece, thus making the match not perfect. This lack of continuity makes also the compression slightly harder for JPEG.

The standard Garb CAPTCHA installation comes with 62 sample images of 150×150 pixels each, which the user can change or add to. To create a challenge, it divides one random image into 4 equal parts, and shuffles their order. Then, it presents this to the user, who has to interchange the puzzle pieces (using drag and drop) to their correct position. The 4 puzzle pieces are associated with an ad-hoc random *id*, and once the user solves the challenge, those *ids* are concatenated and sent back to the server.

Note that a CAPTCHA with only $4! = 24$ possible answers per image can be considered already too weak (4.16% brute-force attack success), more so when after one image is solved, we can use that information to solve it again.

We also detected that the JavaScript client library downloaded the solved image and then divided and mangled it locally. This in itself is a major flaw, and then, it does not require for the answer to be sent back to the server. But still, we were interested in analysing its strength, as if these flaws were corrected.

Even though the Garb CAPTCHA takes away one pixel line in the border of each one of the four sub-images that constitute the puzzle, that is not a major problem for our attack. We do not take into account this 1-pixel border, as it would be disruptive of the *continuity* of the image.

For our attack, for each image of the library, we created a random permutation, and to solve it, we considered all possible $4!$ permutations, and we chose the one with the smallest JPEG file size using maximum JPEG image quality. Note that for our attack, the initial permutation chosen for the challenge does not really play a role: if the correct image is the one with

the smallest JPEG file size, it will always be selected.

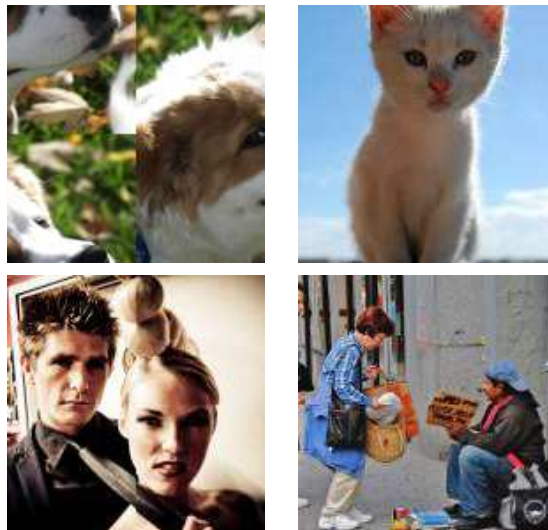


Figure 3.10: Several computed solutions for the Garb CAPTCHA.

Our attack performs very well against this CAPTCHA. In particular, it is able to correctly solve 61 of the 62 images in the standard library. We did 1000 sample tests, and obtained a 98,1% success rate, in line with the $\frac{61}{62}=98,3\%$ expected. The image that is incorrectly solved is consistently solved incorrectly, as we would expect (the first image in Figure 3.10). Note that, in an improved version of our attack (for a larger image library), it would be easy to try the second, third, etc. best solutions to each failed challenge, once we learn that the first best solution of our JPEG attack is not correct. This is possible because there is an order of *goodness* associated to each permutation, its JPEG file size. This improved attack would theoretically reach a 100% success rate.

3.8 Possible improvements

In this section we discuss possible ways to enhance the security of the Capy and other puzzle CAPTCHAs. First, we comment the possibilities of using a broader solution space, thus making it more resilient to brute force attack. Next, we present how to use our attack to filter out challenges that are easily solved, and discuss its possible drawbacks. We also discuss on the possibility of in-depth analysing the interactions of the solvers with the CAPTCHA to

try to gain information on whether they are or not humans. We comment the benefits (and proper ways) to present a bigger image library, thus trying to avoid attacks that, once a particular challenge is solved, can solve it again challenges with the same background. Finally, we analyse whether adding puzzle pieces (instead of just one) might or not be a proper solution to strength the CAPTCHAs.

3.8.1 Broader solution space

We can think that one possible solution might be enlarging the solution space, approximating it to the maximum possible given the dimensions of the background image (a $width \times height$ solution space). This would not only bring the success rate of a brute force attack down to $\frac{1}{width \times height}$, but also, at least theoretically, make more difficult any other attack that takes advantage of the fact that a n -pixel-away solution is not in the solution space (for $n < 10$) (i.e., puzzle pieces that are almost in their correct position, rendering the smallest JPEG size).

The drawback is that this would make it much harder to be solved by humans: placing the puzzle image exactly in its position is not an easy task. We wonder whether there is another way of having that broad solution space, while at the same time maintaining the user friendliness.

One might think that we can check, once in the Capy server, that the solution given is *close enough* to the perfect solution within a distance. Imagine that we allow the puzzle piece in the client to be dragged and dropped anywhere but on the server, we calculate the distance to the correct position, and accept the solution when it is less than 10 pixels away, for example.

The problem with this idea is that once the attacker determines that a 10-pixel distance to the correct solution is accepted, then again, she can create a grid in 10-pixel steps and try only those solutions, knowing that at least one of them will be accepted. Now, the attacker will not necessarily step over the perfect or best solution, so her decision algorithm will have to be able to pick up the best solution within a set of bad but close ones. If her algorithm gives a continuous measure of *goodness* of a solution, this would not be a problem. Thus, depending on the attack, this possibility might not really be an improvement.

We run this test for our attack: what would happen if we consider

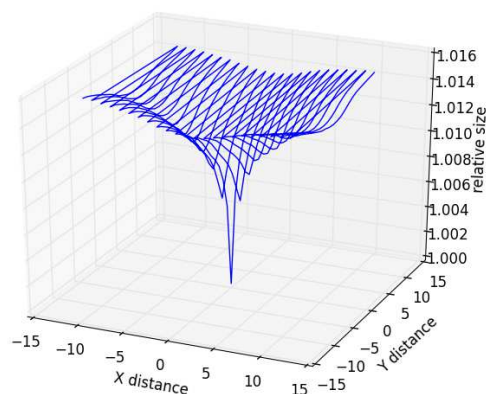


Figure 3.11: JPEG size proportions at different distances from the correct solution.

also puzzle positions that are *almost* correct, that is, with distances of 1-pixel to the correct one, 2-pixels, etc. Would still the correct solution have the smallest JPEG image size?. We run this experiment for all previously correctly solved challenges, with maximum quality setting.

Unfortunately for Capy, we obtained that even with the solutions as near to the correct as 1 pixel distance, they can be detected as wrong by their larger JPEG size - in all occasions. Figure 3.11 gives a medium of the relative JPEG file size when the puzzle piece is put 1 or more pixels away from the correct position. This result is probably due to JPEG having difficulty to compress the high-frequency 1-pixel (or more) differences, and then, increasing the resulting file size.

The idea of using a broader solution space might still be of interest. For example, Capy can decide on its server the allowed distance to a correct solution based on other parameters, like how many attempts from the same IP address had been made within some time, success rate and even including an analysis of the full drag log to estimate the likelihood that it comes from a human. If the allowed distance is variable, an attacker will be forced to always use the lowest possible in her attack. Also a bigger image would force the attacker to try more positions, at least making the attack a bit more expensive in computational resources.

3.8.2 Challenge pre-filtering

Our basic attack does not correctly solve all the challenges presented. This can be used by Capy designers to pre-filter the challenges served by their server, that is: offer only those challenges that are not solved by this attack. Note that this will not require a major modification to their CAPTCHA, and would make it resilient to our attack. This will in principle work for KeyCAPTCHA, but not for Garb, as we can try the *next best* solution, after failing once, if we identify the challenges. The negative point of this solution is that it will make it resilient to the attack we present here, but probably, only to these types of attack.

Another type of pre-filtering is possible, this time, based on selecting the puzzle piece void to be filled using another image that has similar patterns and colours to the real puzzle piece. In section 3.6.3 we learnt that we can, for instance, fill the puzzle void in the background challenge with portions of the same image, so colours and textures are alike. We also learned that background selection can help this CAPTCHA: images that mix very plain and very detailed zones are interesting as backgrounds, as our attack will sometimes try covering the detailed ones with less detailed puzzle pieces, to minimize image size.

These countermeasures might not only be able to prevent the attack we present here, but also other attacks based on image continuity. The problem with them is that the resulting CAPTCHA might not be as user friendly.

3.8.3 Bigger image library

Having a small image library for the backgrounds of the challenges is a major problem. Here, the meaning of *small* is any number that we can download, store and analyse programmatically with a computer. Instead, what we need is a really large number of possible backgrounds, at least from the point of view of a computer algorithm.

Using a much larger image library, and also using some image distortion algorithms so the images are not pixel-per-pixel similar in a way that the distortion is impossible for any other algorithm to undo, even if given several samples of the same image once distorted, may also help.

This is not straightforward, but it might be possible given the current technology. Angle transforms plus local and global image warps, light changes, and colour changes, with added low-frequency noise, might not affect the human eye ability to recognize the image, but will make it much harder for an algorithm to rebuild the original background, even given several distorted samples of it. This could also apply to KeyCAPTCHA and Garb CAPTCHA.

3.8.4 Client interaction analysis

The client-side JavaScript libraries of the Capy CAPTCHA send to the server not just the final position where the user locates the puzzle piece, but the whole record of positions that the piece visits in its way from the initial position to the solution provided by the user, the whole log of the mouse or finger *drag*.

It remains to be seen if it would be possible to categorize these drag logs into human and non human. We can apply Machine Learning clustering algorithms to try to find clusters of *typical* drag movements for different sets of directions.

As the client side JavaScript libraries can be easily modified, timing data can also be added to the position stream, making for another possibility to classify typical human behaviour.

In any case, this is an example of Security Through Obscurity, as the detection algorithm would be proprietary. It as a way to improve the strength of the CAPTCHA, but should never be the main security discriminant in use.

3.8.5 Several puzzle pieces

It is possible to argue that the idea mentioned by Capy designers of using more than one puzzle piece at a time might improve its security. This will be very possibly the case, but remains to be seen to what extent. For example, if just placing one puzzle piece (of, say, three pieces) in its correct position gives the correct properties to the resulting image - which in the case of our attack can be described as *continuity*, then the attack can solve first one piece, then the second, then the third. If the general success rate of the attack is $X\%$, the three-piece success rate would be $\frac{X^3}{10^6}\%$. For example, for our basic attack

with a success rate of 65,1%, for a three piece puzzle it would solve it 27,5% of the time, what is still a very successful attack.

The idea of using several puzzle pieces is already present in Key-CAPTCHA, that uses 2 to 3 puzzle pieces per challenge. This adds more workload for the user, but as have been shown, does not protect the CAPTCHA if its basic idea is flawed.

3.9 Discussion

In November 2015 this work was presented at The Computer Laboratory at Cambridge University, kindly invited by Prof. Ross Anderson. Prof. Markus Kuhn made an interesting remark regarding the possibility that this JPEG-size attack was so successful thanks to the image having been previously compressed loosely (like in JPEG). We thought this was a very interesting point and run an experiment to test this hypothesis⁵. This experiment uses as input the RAISE dataset of RAW images⁶, which consists of images that have never been compressed before. In particular, it contains "8156 high-resolution RAW images, uncompressed and guaranteed to be camera-native, never touched or processed before" (Dang-Nguyen et al., 2015). Using these images as a background for a puzzle, and taking a 80×90 pixel square with a sub-image from another random image, we performed the previously described JPEG-size attack using quality = 100. We got a success rate of 26% when the RAW images were resized and cropped, and of 30% when they were just cropped to the original Capy image size (405×270 pixels). That is, the attack would still be able to break the CAPTCHA in this case. This result evidences the efficacy of using this metric to measure visual information.

In this chapter, we have presented a low-cost, side-channel attack, easy to implement and able to break (bypass) Capy CAPTCHA with a 65,1% success rate. Then, we have shown how this attack, with minor modifications, can break other image CAPTCHAs that use the same puzzle ideas - although in slightly different ways. In the case of the KeyCaptcha our attack is successful in 20% of the cases and in the case of Garb CAPTCHA 98% of the times.

⁵This experiment is available online at <https://github.com/carlos-havier/jpeg-experiment/>.

⁶This dataset is available from <http://mmlab.science.unitn.it/RAISE/>.

We have also discussed some ideas to make puzzle CAPTCHAs resilient to this attack and others. Although theoretically it should be possible to increase the strength of these CAPTCHAs by correcting these design flaws, some of these corrections would possibly compromise its usability, perhaps to a level that would make it too difficult for humans.

More importantly, the ideas presented in this chapter used for puzzle CAPTCHAs can easily be applied to other cases. In particular, the analysis of the challenge domain and answer domain, although straightforward, can give us significant feedback from an attacker's point of view. Also the use of well-known metrics and/or creation of new, derived ones, can be extremely useful, as shown in this case. Although these metrics were not designed to be able to break a CAPTCHA, the information they can give us can be useful, if not determinant, to do so.

In this chapter we have presented a new type of security analysis for a type of CAPTCHAs that have not been analysed before. This security analysis is based on an initial estimation of the strength by studying the challenge size and distribution, and a study of the answers to the challenges using different metrics. This type of analysis is the base for the other analysis on the following chapters. It is also related to the initial phases of BASECASS, the methodology we propose in chapter 6.

Chapter 4

Case Study: The Civil Rights CAPTCHA

OCR CAPTCHAs have been thoroughly analysed, yet the Civil Rights CAPTCHA (*CRC* from now on) intends to increase their security by adding a completely new type of test, an empathy test. In NLP, there has been research done in sentiment analysis, but on the sentiments of the writer, not the reader. A text can be objectively written and still produce an emotion on the reader. One of the aims of this dissertation is to study the security of new, original CAPTCHA proposals. We consider that this new dimension is extremely interesting, and as it has not been studied before, we selected this CAPTCHA for its study.

In this chapter we analyse the Security of the *CRC*, that as we explained, is an original CAPTCHA that aims to increase the strength of a typical OCR-CAPTCHA reinforcing it with an original challenge: an empathy test. This combination purposely leads to a stronger, more secure CAPTCHA overall, while making users aware of Civil Rights news around the world.

We further present this CAPTCHA in Section 4.1. In section 4.2 we analyse the security of the *CRC* to find possible weaknesses, that we discuss in section 4.3. Then we further examine the risk associated to these shortcomings using ML. Once we find them exploitable, we design an attack and checks its results . In section 4.8, we conclude with a discussion about the *CRC* and also the potential of the method we used to break it.

We demonstrate a novel attack against it that can bypass the *CRC*

20% of the time. Interestingly, our attack does not use any OCR technique nor any of the techniques to attack OCR CAPTCHAs that have been used before. The attack we present is a side-channel attack. It does not try to solve the problems used as a foundation for the CAPTCHA, i.e. solve the empathy problem, nor the general OCR/word recognition problem. We do not claim to be able to solve all OCR instances neither extract the empathy of any text. Instead our attack solves both of these problems for this particular instance, for the subset of challenges that the *CRC* is based on. We achieve so by identifying the security issues in the design of this HIP and by applying well-known ML algorithms to exploit them.

4.1 Civil Rights CAPTCHA description

The *CRC* has been designed by the Civil Rights Defenders, an international Human Rights organisation from Sweden founded in Stockholm in 1982, with the help of the Bärnt & Årnst digital production company. The *CRC* has received quite broad media coverage¹ (as they claim on their web-page, see Figure 4.1). It has been awarded several prizes in the field of Civil Rights and marketing.

Another objective of the CAPTCHA, apart from protecting web-based services, is to enhance the diffusion of Civil and Human Rights news along the World. To do this, it is fundamental for the *CRC* to become widely implemented, and thus, its news will be presented to a broader audience.

The *CRC* is based on the human ability to feel empathy after being presented with a news excerpt, typically containing some news about Human Rights and/or Civil Rights around the world. This CAPTCHA is also based on Securimage, a word-distortion OCR/text CAPTCHA. The Civil Rights CAPTCHA works picking up a Civil Rights news from its database, using Securimage to create three possible answers and presenting them to the user. These images contain words describing feelings (i.e. "agitated", "happy" and "angry"). The user has to write down the most appropriate one based on the emotions originated from the news headline presented to her. This news bit is related to Human or Civil Rights, and supposed to create an empathy feeling on a human reader. If we consider the *CRC* well designed, and Securimage

¹It has been covered and praised by The Huffington Post, Discovery, Wired, NBC News, the Daily Mail, and others.

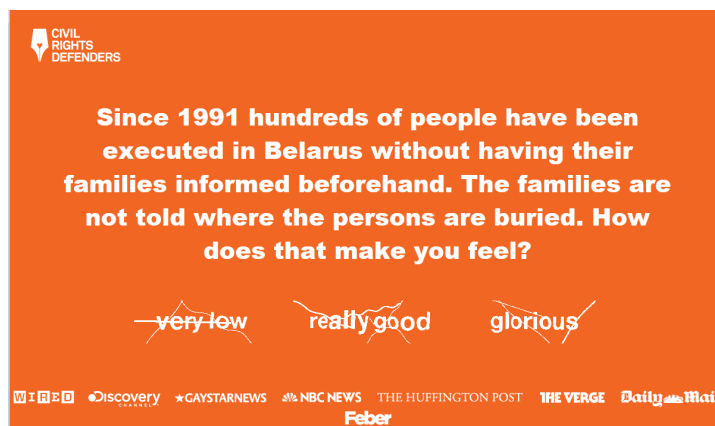


Figure 4.1: Civil Rights CAPTCHA main web-page.

to provide a security level X , this CAPTCHA design should increase the security to $3 \times X^2$, as for a robot, picking up the correct answer should not be easier than random guessing. We will see briefly that the security of this CAPTCHA is unfortunately below X .

4.2 Civil Rights CAPTCHA analysis

CRC is provided as a service directly accessible using an API. This API allows a programmer to connect to it and download a challenge composed of a news text along with three images. Each image contains one or two words distorted using *Securimage*. One of the images contains the word(s) that are the correct solution to the challenge. The same API allows sending the text the user inputs to the *CRC* server, to check if it is a right answer (human) or not (program). As using the *CRC* API directly can be a bit too much trouble, the *CRC* also provides a full library written in *PHP* encapsulating it that just presents the challenge, and checks the answer.

The three answer images to each question, provided as *PNG* images, contain words distorted using *Securimage*, a very popular Open Source CAPTCHA library written in *PHP*. *Securimage* is one of the few Open Source OCR CAPTCHA that has been improved and maintained to date. It is rather flexible, letting the CAPTCHA designer choose, among other parameters:

²That is, divide the force of a brute-force attack, or any other attack against *Securimage*, by a factor of 3.



Figure 4.2: Example of challenges created with Securimage.

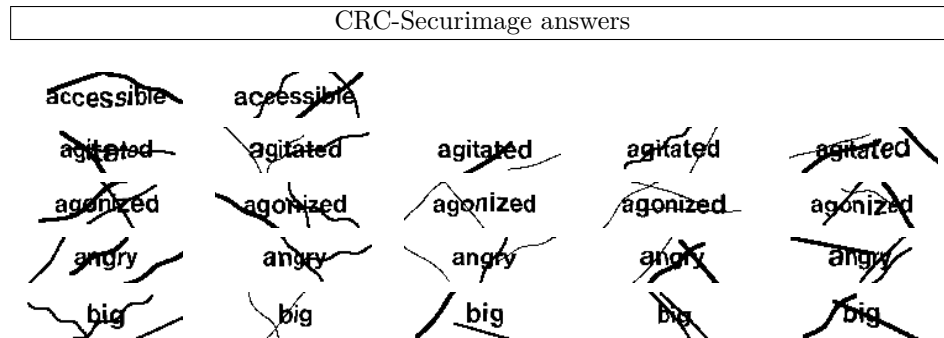


Figure 4.3: Each row shows different CRC image answers created with Securimage that contain the same words.

fonts, colours, grade of distortion (affecting its difficulty), the number of lines crossing the text, characters/words to use, etc. Figure 4.2 shows some of the possibilities of image generation of *Securimage*.

The number of different news excerpts used by the *CRC* is not public, but seems not high: during our interactions with *CRC*, we frequently saw some repetitions. From the *FAQ* on the *CRC* web-site: "How large is the current data set, and do you plan on adding more data to it over time? The statements will change over time, both to update to current events, as well as improving the security of the CAPTCHA. Work will also be done to minimise the possibility of automated sentiment analysis. The limitations of using only emotions for the correct answers are also under consideration. However, at the present, the data-set is still large enough to meet the recommendations of well-proven open-source solutions."³

It is also common to see some answer word repetitions, even though the images themselves do not repeat, as each one is a unique creation with *Securimage* (see Figure 4.3).

In order to analyse the *CRC* we decided to examine its client-server

³Retrieved from <http://captcha.civilrightsdefenders.org/> on the 1st of March 2014.

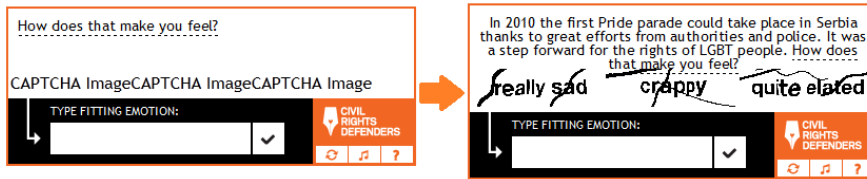


Figure 4.4: Initial HTML body from the *CRC* API (left), and after being filled with data (right).

communication from the endpoint, the same viewpoint a real attacker would have. We used an HTTP traffic analyser. After a few interactions and tests, we were able to decipher the core of the client-server protocol needed to further analyse it:

1. The first step is to request the main content for the CAPTCHA. The answer, if presented as-is to the user, will be an *empty* HTML structure where the real content (news-bit, answer images) will be filled in later using JavaScript (Figure 4.4). Another important function of this HTTP answer is to set the value of the *ci_session* cookie. This cookie is a meta-cookie containing several bits of information, like a *session_id*, the *IP* address of the client, information about its *user-agent*, etc.
2. Once this is loaded, the client JavaScript code makes another request with the same URL and *?sessid=1*. This sends back an answer containing the *PHPSESSID* cookie.
3. In the next request, the parameters *callback*, *newtext* and *lang* are added to the URL, causing the server to send back the text of the challenge - the news bit. This text comes back as a JSON-encoded text.
4. The browser downloads the three images containing the answer. To download each one, the browser requests a unique (and random) 20-character id. The server keeps track of the ones to send using the previously provided cookies. All the elements are now visible to the user (Figure 4.4).
5. After the user has written the answer, this is sent to the server encoded in the URL of the next request.

After learning the cookie handling and the JavaScript requests of the *CRC*, the rest of its mechanisms are quite straightforward, and we were

able to advance to the analysis of its functionality, gathering some basic data automatically.

4.3 Civil Rights CAPTCHA design flaws

We mentioned that the three images downloaded contain different possible words or expressions. We wondered what would happen if we keep asking the server for more word-image answers. We confirmed that the server keeps providing us with new word-images, independently of the *newset* parameter. We wanted to know whether the server is keeping track or not of the word-images sent, and if it only checks that the answer is valid (positive, negative) according to the news bit. To check this hypothesis, we wrote down a few positive answers and a few negative ones from other questions. Then, we proceeded to the next question, *"In October 2012 the Ukrainian parliament took the step to approve a law, which criminalises 'propaganda of homosexuality'. How does that make you feel?"*. The corresponding word-image answers were *very crappy*, *elastic* and *hopeful*. Being a negative news bit, we decided to respond with a negative answer present in other questions but not in this one, choosing *horrified*. The server did not accept our answer as correct. We tried the same attack a few more times, without success.

Our conclusion at this point was that either the answers are divided into finer categories than good/bad or, more probably, the server keeps track of the sent word-image answers (probably the last three). To find out the correct hypothesis, we proceed with the attack, collecting logs of wrong and correct answers. Then we tried again using these logs to find correct answers for each question. And again, we got a fail. The only possible conclusion is that the server keeps track of the word-image answers sent.

We also learned that most (if not all) answers were shared as correct for more than one question. For example, the answer *pleased* was accepted as valid for at least four challenges. We did not observe clustering of correct answers in several groups. These lead us to believe that the classification applied to the answers by the *CRC* is coarse: *positive* or *negative*.

Once we finished testing the *CRC* and got familiar with how it operates, we analysed the challenges it could present to the users. We wanted to know their number, distribution, and if any characteristic of them were not uniform. For this purpose, we wrote a program able to follow the protocol

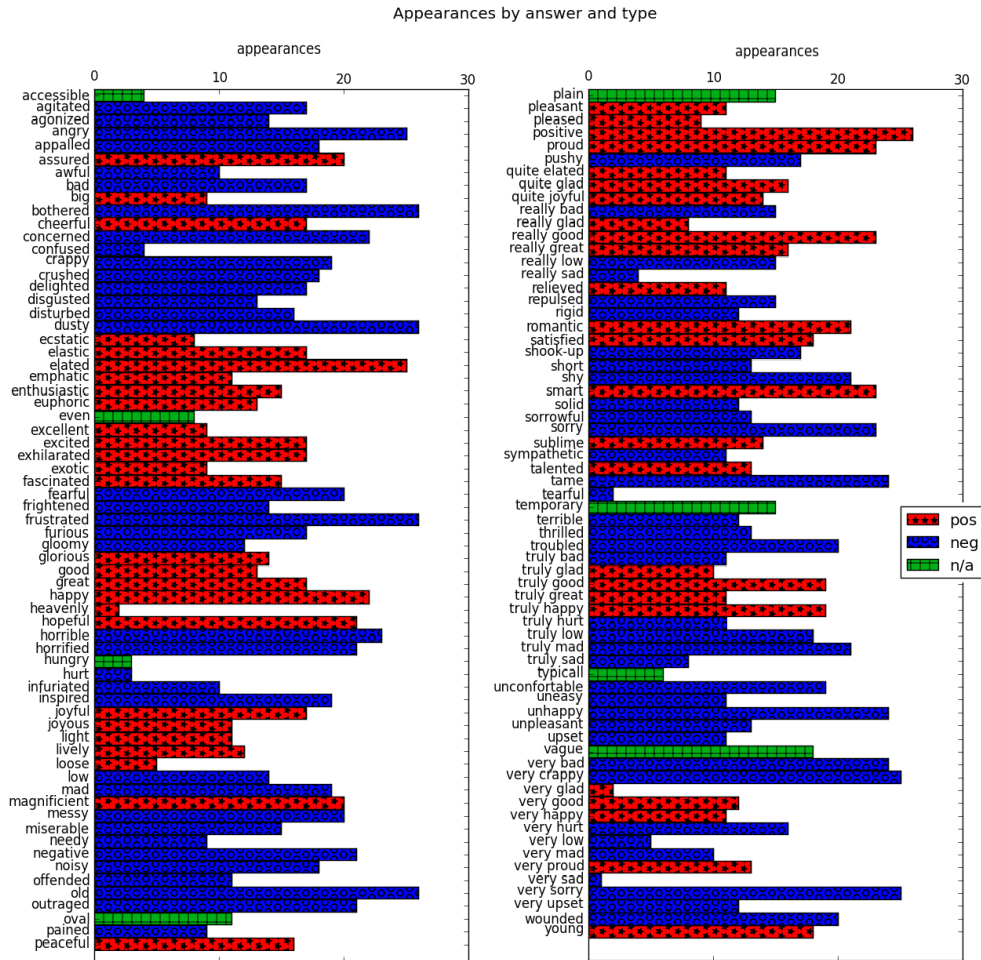


Figure 4.5: Number of appearances of each of the 133 answers (from 1 to 26 (1.3%)).

of the *CRC* and mimic a regular user, downloading and interpreting the information. After downloading 1000 challenges, we saw that there were only 21 different challenge texts. This is in itself a flaw, as it is too low for a CAPTCHA.

We also checked how many times the *CRC* server shows each question to the user. This distribution follows a seemingly uniform distribution of appearances, with a χ^2_{20} with a p-value of 0.336.

Each challenge comes with three different answers. During our experiments, we have been able to observe 133 different answers, 30 of them compositions of the words *quite*, *really*, *truly* and *very*, and some of the

remaining 103 basic categories. Again, this is a problem. A CAPTCHA with only 133 possible answers is a CAPTCHA that can be broken 0.75% of the time just by answering any of them.

The answer type distribution is not uniform. There are 73 (55%) answers describing a negative emotion, 52 (39%) describing a positive one plus 8 (6%) not describing a valid emotion (like the answers *accessible*, *oval*, *plain*, *temporary*, *typical*...). The distribution of their appearance, taken from 1989 manually classified images, seems to be uniform within the different categories, with 59,2% positive, 36,8% negative and 4% neutral. It is not *uniform*, as the value of its χ^2_{132} is 482,12, giving a p-value of 0 (or more precisely $2.32e - 41$).

This can be further exploited in a blind brute-force attack. We can pick up randomly one answer from the top 5 more probable ones, and this would pass the CAPTCHA approx. 1,2% of the times. This kind of attack is not our purpose here: we want to study if we can exploit its other vulnerabilities and improve this result.

In summary, the *CRC* provides us with two problems: reading distorted words, as in any other OCR/text CAPTCHA, and tagging emotions to news excerpts. Neither problem is new to ML. The first one has been solved several times for several particular implementations, some of them without the need of ML. Interestingly, OCR/text CAPTCHAs like Securimage are still widely popular. Securimage tries to avoid segmentation, a well-known weak spot of OCR/text CAPTCHAs, placing several curved lines over the letters, even though recent attacks might cope well with these anti-segmentation techniques (Bursztein et al., 2014, Gao et al., 2016).

Regarding the second ML problem of text emotion, several ML algorithms have been proposed that can deduct emotions from texts. However, to our knowledge, all of them focus on trying to infer the feelings of the *writer* of the text, and none attempts to estimate the emotion that the text would produce on a human *reader*. This aspect is somewhat crucial, as these news excerpts that are written in an objective language use both few adjectives and neutral nouns, both of which are an essential part of many ML approaches to deduction of text emotion.

Because of this, we consider a further analysis of the *CRC* especially attractive not only from a security standpoint but also from an ML perspective. In the following section, we will get further insight on both problems and will

find solutions using ML that can be later used in an attack on the *CRC*.

4.4 Foundations of the Machine Learning attack

In this section, we will analyse the design flaws of the *CRC* and how they can be exploited using ML. The results of this analysis will be the foundations of our attack to the *CRC*.

Solving the *CRC* can be divided into two phases: *reading* the Securimage-protected answers and *classifying* the challenge text according to the emotion it should create on the reader.

Given the design flaws of the *CRC*, classifying the news excerpt texts is not strictly necessary to solve it. We would like to know how to cope with a better designed *CRC* with a bigger news database. Also, classifying the news excerpt texts will improve the attack results.

In the next sections, we explain these two phases in detail and present an attack based on them in section 5.4.

4.4.1 Reading the answers

The current iteration of Securimage might be a good OCR/text CAPTCHA, but the way the *CRC* employs it in makes it weaker. The problem is Securimage was originally designed to work with a large alphabet, and either random words, or a huge dictionary. If we restrict it to just 133 words, its disguising capabilities might not be good enough for a strong classifier. This is what we decided to test.

The metrics we gathered from the images to feed our classifier were general ones: black pixel count and pixel count per column. These two values will be affected by the presence of the two or three random black lines that Securimage is told to produce by the *CRC*.

The lines drawn are typically of the same thickness all along their length. In this case, a derivative of the number of the vertical pixels in each column will be affected by their presence only at their start and end (ideally, as an intersection of lines and letters would affect too). Thus, we decided to

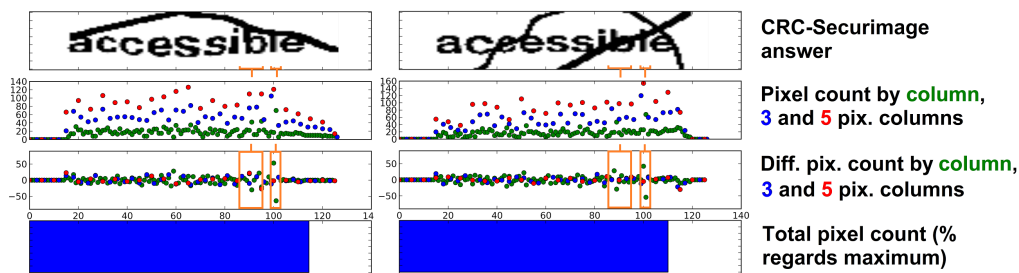


Figure 4.6: Example metrics of some CRC answers: pixel count by column (green), and groups of three (blue) and five (red) columns; differential for these values, and the total pixel count, as % of the maximum from all answers.

add this statistic.

Figure 4.6 shows an example of the value of these metrics for two CRC answers that pertain to the same category. From top to bottom, the first graph shows the pixel count by column (green), and groups of 3 (blue) and 5 (red) columns. The second graph shows the differentials for the first ones, also per column (green), and groups of 3 (blue) and 5 (red) columns. Highlighted in this graph, two rectangles in orange that explain how the differential is less affected by horizontal or diagonal lines, and more affected by vertical lines (sudden changes), due to characters, as we want. The last bar graph is the total pixel count, as % of the maximum from all answers.

We decided to sample the image at every column, and also in groups of 3 and 5 columns. We used a very simple approach - our intent is to do the least possible analysis and let the ML algorithm do it for us. Note from Figure 4.6 that the derivative of the pixel count per columns provides a good representation of the start and end of letters, as well as vertical strokes, even in the presence of distortion lines.

Choosing typical metrics, we try to replicate the approach we would expect from a *low-cost* attack, that is, an attack seeking to obtain the most results investing the least effort. A low-cost attack is the main risk for a CAPTCHA that has not gained widespread use yet. No significant image processing, nor OCR techniques, are used in this attack. All we do is to compute these simple metrics for each image and let the ML algorithms cope with the data.

To create a training set, we downloaded 1989 answer word-images, and manually classified them into the 133 possible categories.

We fed them to Weka (Hall et al., 2009), using all compatible classifiers. We were not able to use all the classifiers available in Weka 3.7, due to several errors from some of them such as MODLEM (Greco et al., 2001), Logistic, Bayes Average 1 and 2 Dependence Estimators, SMO, and with other algorithms running for too long without producing the model, such as DTNB (Hall and Frank, 2008), JRip (Cohen, 1995), MultiLayerPerceptron, SimpleLogistic, K* (Cleary and Trigg, 1995), LWL (Atkeson et al., 1996), NBTree (Kohavi, 1996), LADTree (Holmes et al., 2002) and LMT (Landwehr et al., 2005). We used both 2-fold and 10-fold CV, depending on the time to build and run the different models. Typically, we did a 2-fold CV of all algorithms, and again a 10-fold CV of the most promising ones, if time was available.

Table 4.1 shows the results of the best-behaved classifiers during these experiments. Each row represents a different ML classifier of the ones available in Weka. The first column is its name, the second shows the test mode (number of CVs), the third column is the accuracy obtained (represented as % images correctly classified), and the fourth column is the κ statistic (a measure of accuracy vs. a random classifier).

After all these tests, we realized that best out-of-the-box classification was obtained using LibLINEAR, that is based on Linear Regression and Linear Support Vector Machines (Fan et al., 2008), although good results were also obtained using Random Forests (Breiman, 2001), Additive Logistic Regression (LogitBoost (Friedman et al., 1998)), Voting Feature Intervals (VFI (Demiroz and Guvenir, 1997)), Nearest-neighbor using Non-nested Generalized Exemplars (NNge (Martin, 1995)), Naive Bayes, and with J48 trees (Quinlan, 1993), etc.

As Table 4.1 shows, we were able to correctly *read* the answer 59,3% of the time. This result was obtained without any kind of image processing or any other traditional OCR technique. This result shows the weakness of using Securimage with only 133 possible categories as answers.

4.4.2 Classifying the challenge text empathic emotions

As there are only 21 challenges, we can memorise their positive/negative classification. The reason why we do not do it here and instead try this ML approach is because we want to know whether, if the number of challenges is raised properly and actively maintained, it is still possible to successfully

Table 4.1: Best classifiers for OCR the Sercurimage challenges of the *CRC*, ordered by accuracy and κ statistic.

Weka Scheme	Test mode	Correct (%)	κ statistic
LibLINEAR	10-fold	59,35	0,58
Random Forest	2-fold	51,30	0,50
LogitBoost	2-fold	47,73	0,47
VFI	2-fold	45,82	0,45
NNge	2-fold	42,80	0,42
Naive Bayes	2-fold	40,59	0,39
Multi Class Classifier	2-fold	38,48	0,37
IB1	2-fold	36,51	0,35
J48 graft	10-fold	33,45	0,32
Random Sub Space	2-fold	32,59	0,31
J48	10-fold	32,19	0,31

attack the *CRC*.

Given the *CRC* design flaws, it is possible to do better than 50% (random pos/neg classification). *CRC* presents only 21 different questions, 7 of them positive (33%), and 14 negative (66%). The problem with this is that a *lazy* all-negative classifier would have a 66% success rate. Improving the success of our attack would be as easy as discarding all read answers that are positive.

This result can be further improved. Several projects are available to classify the emotion of a written text (Bird et al., 2009, Nielsen, 2011). The problem with most of them is that they typically classify the feelings of its author by checking the adjectives and/or nouns used.

This approach is not suitable for our case because the news are objectively described, with no or little use of adjectives, but still, can create an empathic emotion on the reader according to the positive or negative impact on other people.

We tried some of these approaches, but they did not give good results. For example, the classification of sentiment provided by SentiWordNet proved to be not significant for our purpose. To illustrate this, we can mention that the sentiments for the verbs *imprison* and *incarcerate* were both neutral, as happened with many others.

After examining different possibilities, we decided to use the Python

Natural Language Tool-Kit (*NLTK*) library (Bird et al., 2009). This library provides several algorithms for treating Natural Language problems, some of them to classify text, including decision trees, maximum entropy, SVMs or Naïve Bayes, just to mention some.

To be able to train our model, we needed to manually classify a set of similar news excerpts as either positive or negative. We found two primary sources of news extracts of similar thematics, the Human Rights Watch (HRW) association, and the Civil Rights Defenders (CRD). This last one happens to be the one associated with the *CRC*. We downloaded 152 news from the Human Rights Watch association (most of them of negative content), and 643 from the Civil Rights Defenders, of which 21 are related to the questions on the *CRC*.

After some initial testing, we saw that the HRW corpus was not very relevant, being one of its main flaws that only five of the 152 news had a positive character. We decided then to only use the CRD news corpus, in two versions: the 622 version (without the 21 news related to the *CRC*), and the complete 643 version.

We followed these steps with different input data from the CRC news corpus:

- Data cleaning: as an initial step, we took out of the bags of words the name of any country (and the corresponding adjectives), the name of the civil & human rights organizations and other related organizations, and of course, the NLTK stop-words for English, so they were not used for classification.
- Data transformation: we processed the news corpus with pos-tagging to translate it into WordNet synsets, wanting to know if adding some of the knowledge represented in WordNet would help the classification (WordNet can be considered somehow an Ontology, given the relationships among its synsets). We built different corpuses: one with the original plain news excerpts, another one with the synsets of those news words, a third one with synonyms of such synsets, and four others that included the hierarchies of hypernyms, from the root (0) to the 4th level (if present).
- Preprocessing: we converted the corpuses to TF-IDF (term frequency - inverse document frequency) normalised vectors, using a cut-off value of

Table 4.2: Best Empathy classifiers, by algorithm and data.

Algorithm	Best 10-CV f_1	Input	Best 21-CRC f_1	Input
Max. Ent.	0.29	original	0.54	synonyms
N. Bayes	0.46	original	0.78	synonyms
SVM Linear	0.55	original	0.86	synonyms

Table 4.3: Best parameter results in 10-CV, by algorithm and data.

Parameter	Best value	Algorithm	Input	Other values
f_1	0.55	SVM ⁴ Linear	Original	Recall 0.48, accuracy 0.68, precision 0.65
Recall	0.48	SVM Linear	Original	Recall 0.48, accuracy 0.68, precision 0.65
Precision	0.95	Max. Ent.	Hypernyms-3	Recall 0.11, accuracy 0.66, f_1 0.20
Accuracy	0.72	SVM Linear	Hypernyms-3	Recall 0.44, precision 0.70, f_1 0.52

2, that is, not considering words that do not appear at least two times in the corpus, and n -grams for $n=1,2,3$.

- Classify TF-IDF vectors: in order to get better results, we tried three classification algorithms: Max Entropy, Naïve Bayes and Linear SVMs.
- Test: we used 10-CV on the whole annotated news corpus, without the 21 questions that were used in the CRC. For each algorithm and input, we tested f_1 , *recall*, *precision* and *accuracy*. Additionally, we also tested them on the 21 questions of the CRC.

Tables 4.2, 4.3 and 4.4 describe the results obtained when using the 622 CRD corpus. Table 4.2 shows which data set obtains best classification results for each ML algorithm, both for 10-CV and for the 21 questions of the *CRC*. The first column is the ML algorithm. The second is the best f_1 value obtained for 10-CV (being the f_1 score a combined measurement of the classifiers precision and recall). The third column is the data transformation applied to the 622 CRD corpus related to WordNet, being *original* no transformation, *synonyms* their replacement for WordNet synsets of synonyms, and *hypernyms- X* their replacement for chains of WordNet hypernyms to the X -level (being $X = 0$ using only the root synset). The fourth and fifth column show the same information but related to testing the algorithms with the 21 *CRC* questions only.

Table 4.4: Best parameter results for the *CRC* questions, by algorithm and data.

Parameter	Best value	Algorithm	Input	Other values
f_1	0.85	N. Bayes	Synonyms	Recall 0.71, accuracy 0.85, precision 0.83
Recall	0.85	SVM Linear	Synonyms	f_1 0.85, accuracy 0.85, precision 0.90
Precision	1.00	all	Hypernyms-3	Recall 0.28, accuracy 0.76, f_1 0.44 (SVM Linear)
Accuracy	0.90	SVM Linear	Synonyms	f_1 0.85, accuracy 0.85, precision 0.90

Table 4.3 shows the highest values obtained for each parameter tested (f_1 , recall, precision and accuracy). For each parameter, the second column shows the best value obtained, the third column shows the ML algorithm that achieves it, the fourth column shows the input data used: whether it was the original CRW corpus, or some transformation of it using WordNet, as in columns three and five of 4.2), and column 5 shows the rest of measurements for that particular combination.

Table 4.4 is equivalent to Table 4.3, but testing all classifiers against the 21 questions of the *CRC*, instead of doing 10-CV.

Table 4.2 shows that SVM Linear is able to obtain an f_1 of 0.55 for 10-CV (10% of the training set reserved for test). When we confront this model with the 21 questions of the CRC, it is able to obtain a slightly better result of $f_1 = 0.60$, using the original corpus composed of the raw news bits, converted into TF-IDF vectors. Transforming the input data using WordNet knowledge, we got similar results using hypernyms of level 2 and 3. In that case, SVM Linear is able to obtain $f_1 = 0.52$ for 10-CV.

The best performance against the 21 challenges of the *CRC* was achieved by SVM Linear using WordNet synonyms, reaching $f_1 = 0.86$ (precision of 0.90), although only $f_1 = 0.41$ for 10-CV (tables 4.3 and 4.4).

It is interesting to observe that when the training set is of limited size, using WordNet synonyms does clearly improve the classification result for unknown tests (Table 4.2), thus improving the *generalisation* abilities of the classifier.

Given that the *CRC* challenges are indeed included in the news

source, we also tried training using the complete 643 CRW corpus. The results were all similar or slightly better than in the previous experiment. In this case, we obtained the best results using WordNet hyponyms at the fourth level. Specifically, we obtained $f_1 = 0.57$ for 10-CV, and $f_1 = 1$ (correct classification) of the 21 challenges.

Due to the design of the *CRC*, there is another very precise way of classifying each challenge text as positive/negative that would not use ML algorithms. Each time there is a new challenge we, as an algorithm, do not know whether the correct answer should be positive, or negative. We can still *read* each answer 59% of the time, and if there is a small number of them (133 in our case), classify them as either positive or negative.

Thus it is possible to keep answering randomly, using one of the answers we read, and when we succeed, look at the type of answer that was successful (positive/negative).

In this way we can use the *CRC* as an oracle that will tell us, after certain time, the category of each question through this brute-force search. This will allow us to correctly classify any new challenge text.

Adding these questions does not seem to have a significant beneficial impact on the security of this CAPTCHA. The fact that the empathy classification of these questions appears to be quite coarse (positive/negative) means it will not significantly add security to the CAPTCHA.

4.5 Machine Learning attack to the Civil Rights CAPTCHA

In this section we will introduce the attacks we conceived using the previous knowledge about how to exploit the *CRC* design vulnerabilities using ML. Any attack to the *CRC* can be broadly divided in *reading* the Securimage-protected answers and *classifying* the challenge news-excerpts. Given the design flaws of the *CRC*, classifying the challenge text is not strictly necessary to pass it, but it can help improve the attack efficiency. We specify how we use well-known ML algorithms for solving both problems to a level that breaks the *CRC*.

We designed two attacks. The first one simply uses the previously

tested ML algorithms to bypass the *CRC*. We called this one the *basic attack*. The second is a slightly improved version that we will call the *improved attack*. It saves the data about the answers already submitted to each challenge and classified as right or wrong. That information is used to prevent sending wrong answers again or picking up a right answer if present.

To test our attacks, we have created a program in Python and used Weka to classify the answer images (*reading* them) using our pre-trained classifiers. Our program downloads the challenge text and corresponding images from the CRC server. Then, it analyses the three images that contain the possible answers to the question using a previously trained classifier, as explained in section 4.4.1.

Among the possibilities, we chose the SVM Lineal algorithm, translating the texts to chains of WordNet hypernyms, which obtained 1.00 precision during our tests, as shown in Table 4.4.

In brief, our basic attack consists of the following steps (Figure 4.7):

1. Our program connects to the CRC server to download the challenge text and the three images, taking care of the cookies.
2. Use our previously trained LibLINEAR classifier to read the three images, obtaining both the word(s) in the image (with a 59% success rate) and the *certainty* of the classifier in the classification (between 0 and 1).
3. Look up the words in our manually created list to annotate them as negative or positive.
4. Translate the challenge text into chains of synsets representing their hypernyms, from levels 0 to 4, using WordNet.
5. Use our previously trained SVM Linear classifier to classify the text of the challenge as creating a negative or positive emotion (with 100% success).
6. Filter the words obtained earlier by the corresponding empathic emotion of the challenge text.
7. Pick randomly one among the remaining words as the answer to the *CRC* challenge. This random selection will be weighted by the *certainty* of the classifier over the different words.

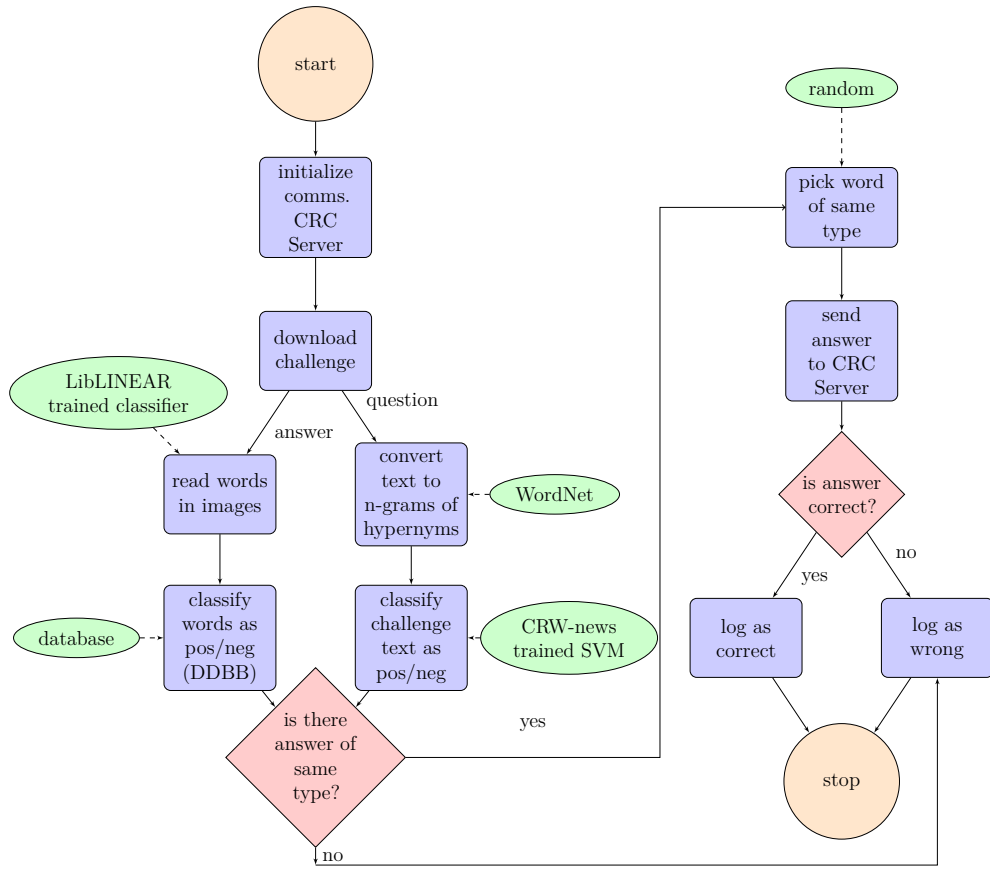


Figure 4.7: Flow chart of the CRC basic attack.

8. Send the chosen answer to the *CRC* servers.
9. Process the server answer to see if it was correct or not, and log accordingly.

4.6 Experimental results

In this section, we explain in detail the results of the two attacks introduced before, and compare them. These two attacks join the results of the previous sections, showing that the use of ML to exploit the *CRC* design flaws is indeed able to break it.

Due to technical problems, including the large amount of time taken

by the *CRC* server to provide a full challenge (up to 45 secs. during our tests) and the low reliability of the same (it was typical that the server stopped responding for some minutes on occasions), it was easy to incur into time-outs. It was difficult to finish a large series of experiments. With these restrictions, our lengthiest experiments consist of series of merely 1000 challenges for the basic attack, that took almost 15 hours for each of the four experiments.

Tables 4.5 and 4.6 show the logs of our attacks, showing some examples of the attempts of our algorithm at classifying the question and reading the answer words. In the different columns of Table 4.5 we show the text of the question, its classification according to our model, the three answer images and how they are classified by our image classifier including, for each answer, the *certainty* of the classification, and whether this is a negative or positive answer. The following column contains the selected answer sent to the *CRC* server. The last column is the server answer, and whether it is correct (*True*) or not (*False*).

Table 4.6 has a similar structure, depicting the initial log of our improved attack. One additional line is added per attack, explaining what the program is doing, thanks to the gathered knowledge: either removing known wrong answers or, upon finding a correct one within the ones read, choosing it.

Table 4.7 shows the rate of success of both attacks, using both the 622-news CRW corpus and the complete 643-news CRW corpus for training the question classifiers, in each row. The first column is the size of the CRW corpus. The second column shows the bypass rate of the simple attack, that is, the number of correctly solved CRC challenges vs. the total. The third column shows the same rate for the improved attack.

The basic attack reaches a 17% success rate using the complete 643-news CRW corpus, that lowers to 14% for the smaller CRW corpus. These success rates are regarded as a complete bypass of the *CRC*. The success rate of the improved version tends to increase with the gathered *knowledge*, with a mean of 20,7% for the total experiment (16,5% for the smaller CRW corpus).

This result is clearly better than a brute-force attack, that would break the *CRC* on average $\frac{1}{133}$ (0,75%), or $0,66 \times \frac{1}{72}$ (0,92%) if we restrict ourselves to negative answers. A brute-force attack would not be able to learn the correct answers to each question, as it is not *reading* which answers are present each time.

Table 4.5: Program answers for the basic attack.

Question	Model pred.	Img. answ.	Class.	Secur.	Type	Sent answ.	Result
Members of the Russian g...	neg	pushy	pushy	0.89	neg	shook-up	True
		shook-up	shook-up	0.66	neg		
		vague	happy	1.0	pos		
Serbian authorities bann...	neg	big	old	1.0	neg	terrible	True
		noisy	angry	1.0	neg		
		terrible	terrible	1.0	neg		
In Kosovo people are tor...	neg	excited	excited	0.82	pos	upset	True
		vague	proud	1.0	pos		
		upset	upset	0.8	neg		
In february 2012 the Uga...	neg	disturbed	disturbed	0.93	neg	disturbed	True
		exhilarated	exhilarated	1.0	pos		
		good	even	1.0	N/A		
Members of the Russian g...	neg	bothered	bothered	0.89	neg	bothered	True
		solid	young	1.0	pos		
		truly happy	truly happy	0.667	pos		
Members of the Russian g...	neg	shy	big	0.93	pos	sympathetic	True
		light	even	1.0	N/A		
		sympathetic	sympathetic	0.8	neg		
In february 2012 the Uga...	neg	talented	sublime	0.82	pos	frightened	True
		pushy	angry	0.67	neg		
		frightened	frightened	0.8	neg		
Human rights defender Na...	neg	very proud	temporary	0.92	N/A	disgusted	True
		pushy	pushy	0.75	neg		
		disgusted	disgusted	1.0	neg		
Swedish-Eritrean journal...	neg	pleasant	offended	1.0	neg	horrible	True
		sublime	sublime	1.0	pos		
		horrible	horrible	1.0	neg		

Table 4.6: Program answers for the improved attack.

Question	Model pred.	Img. answ.	Class.	Secur.	Type	Sent answ.	Result
Swedish-Eritrean journal...	neg	proud very proud	noisy	0.89	neg	mad	True
			very proud	0.67	neg		
		mad	mad	1.0	neg		
removing not ok answers to 2 answers,after using knowledge, answers are noisy,mad							
Swedish-Eritrean journal...	neg	noisy crappy joyful	lively	1.0	neg	crappy	True
			crappy	1.0	pos		
			vague	1.0	N/A		
none of the answers is among the known correct,removing not ok answers to 1 answers,after using knowledge, answers are crappy							
In 1948, the UN General ...	neg	frustrated plain plain	frustrated	0.93	neg	frustrated	True
			even	1.0	neg		
			typicall	0.8	N/A		
removing not ok answers to 1 answers,after using knowledge, answers are frustrated							
Swedish-Eritrean journal...	neg	shook-up romantic sublime	shook-up	0.83	neg	shook-up	True
			romantic	1.0	neg		
			smart	1.0	pos		
none of the answers is among the known correct,removing not ok answers to 1 answers,after using knowledge, answers are shook-up							
Swedish-Eritrean journal...	neg	old agonized great	big	0.83	neg	agonized	True
			agonized	0.5	neg		
			lively	1.0	pos		
none of the answers is among the known correct,removing not ok answers to 1 answers,after using knowledge, answers are agonized							
When told that there are...	neg	miserable assured happy	miserable	1.0	neg	miserable	True
			assured	1.0	neg		
			happy	1.0	pos		
removing not ok answers to 1 answers,after using knowledge, answers are miserable							
In 1948, the UN General ...	neg	hopeful frustrated joyful	hopeful	0.83	neg	frustrated	True
			frustrated	0.83	pos		
			upset	1.0	neg		
trimming by known ok answers to 1 answers,removing not ok answers to 1 answers,after using knowledge, answers are frustrated							

Table 4.7: % of successfully solved CRC challenges.

CRW corpus	Basic attack	Improved attack
622	14, 1%	16, 5%
643	17, 1%	20, 7%

If somehow an attacker creates a database of correct answers to each question, and then uses it to answer a random correct answer, its success rate would never be over $\frac{\sum_{i=1}^{i=21} \frac{1}{|solutions(i)|}}{21}$, where $solutions(i)$ is the set of all possible correct solutions to question i .

This would be in a scenario in which the attacker has learned *all* possible right answers to each question: even in our 1000-length attacks we were not able to learn all the correct answers, with some questions having 5, 7 or 9 known correct answers, but others still none.

If the *CRC* had a well maintained database of challenges, such an attack would take extremely long to *learn* all the right answers to all the questions. Our attack will still be able to attain a minimum 17,1% success rate. Once automatically learned some correct and wrong answers, a 20,7% success rate or greater would be possible.

4.7 Possible improvements

Securimage provides many more possibilities than the ones employed by the *CRC* authors. One possible improvement is to use Securimage to its maximum, allowing the use of more typefaces, sizes, more random number of lines, more degradation, etc.

It is important to avoid using such a limited set of possible answers. These should be increased at least a hundred times, but it would be much better if it is increased at least one thousand times. With this type of CAPTCHA, it is quite problematic, but it is a necessary measure not to render the protection provided by Secureimage completely worthless. How can we describe 130,000 different possible empathy feelings? Even more, in a way so they are presented in a random, uniform manner, to the user. It is not a simple question for this CAPTCHA, but one that needs a solution, not to render the protection provided by Secureimage completely worthless.

This CAPTCHA is designed to be based on one or several news sources. Limiting this source of information to one, like in this case, is clearly problematic. The authors should use different news sources, different news writing styles, and also enlarge the subject of those news to cover some topics not directly related to Civil and Human Rights (but somehow related), thus making machine classification harder. It is important that the authors create

big enough corpus, properly maintained, and that they try on it several Natural Language classification algorithms to check that they do not offer much better results than random.

Similarly, it would be important to have many more categories of news, not just positive and negative, and corresponding answers.

All these improvements might be able to protect the CRC from low-cost attacks, but due to the advances in DL, an OCR CAPTCHA cannot be considered secure and the authors should look for alternatives.

4.8 Discussion

In this chapter we have analysed the *CRC* CAPTCHA from a security standpoint. Using simple metrics and ML algorithms, we have been able to break it with a 21% success rate consistently.

We have shown how Securimage is rendered weaker by using it out of the scope it was designed for. We have also shown that the idea of a CAPTCHA based on empathy about text excerpts is not necessarily good, especially if this empathy test can only be administered as a choice between two main categories.

Finally, we have shown that the combination of two CAPTCHAs is not always more secure than one of them alone, as the way the *CRC* uses *Securimage* lowers its security, and in turn allows us to break the *CRC*.

More importantly, we see that also in this case with the *CRC*, it is useful to do a challenge domain analysis and an answer domain analysis. We also see that simple, general metrics, along as some other metrics slightly modified for the case, can give enough information about the challenges as to allow various ML algorithms to break the CAPTCHA a significant number of times.

The attack we present is quite general and does not use any OCR technique nor any of the conventional methods to attack OCR CAPTCHAs. Instead, we use very simple, general metrics, and allow ML to do the heavy-lifting of finding sufficient enough information to create a side-channel attack.

The analysis of challenge and answer domain that we also have

presented in section 4.2 together with the combination of simple, general metrics and ML is promising in its ability to test for compliance with a basic security level in many other types of CAPTCHAs. In fact, these two analyses constitute two fundamental steps of BASECASS, the methodology that we propose in chapter 6.

Chapter 5

Case Study: FunCAPTCHA

This dissertation aims to study the security of new CAPTCHA proposals that present original aspects and have not been studied before. That is why we chose FunCAPTCHA, the first production CAPTCHA that is based on gender recognition of faces.

There are different ML algorithms that we can apply for gender recognition (also known as gender detection or gender classification), as *Fisherfaces*¹ or Delaunay triangulation (Delaunay, 1934) to extract some features (distances between spots) and classify them using Functional Trees (Khryashchev et al., 2012, Gupta, 2015). Recently the accuracy of the different ML image recognition/classification tasks has drastically improved thanks to the advances in Deep Learning (DL), in particular in CNN. They achieve 80% accuracy on a complex dataset of full-body images including both frontal and rear views (Ng et al., 2013). Other authors achieve a 86% accuracy using the much more challenging Adience benchmark², resulting in state-of-the-art accuracy (Levi and Hassner, 2015). Related techniques, as ensembles of DNNs that perform 3D alignment, frontalization and classification, have been used for other face-based problems as face identity verification, attaining an accuracy of 97%, almost at the human level, with medium datasets of 4000 identities in 4 million images (Taigman et al., 2014).

¹There is a Fisherfaces example implemented in OpenCV, available at http://docs.opencv.org/2.4/modules/contrib/doc/facerec/tutorial/facerec_gender_classification.html.

²The Adience benchmark consists in *in-the-wild* pictures that include 2284 subjects in 26580 photos taken from Flickr albums released under the Creative Commons licence. It is available at <http://www.openutd.ac.il/home/hassner/Adience/data.html>.

It is known that facial expressions, poor lighting, complements as glasses, partial occlusions and others can significantly difficult ML approaches to face detection, identification and classification. It is unclear to us whether the designers of FunCAPTCHA were able to find a subset of the gender recognition problem that is particularly hard for the current ML methods. Thus, in this chapter we perform a security analysis on it.

In this chapter we first describe FunCAPTCHA is described in section 5.1. We further study its design in section 5.2. Then, we focus on its security and in its potential weaknesses in section 5.3. To understand the exploitability of these weaknesses, we study how ML can leverage them. To do so, in section 4.4 we define some general metrics and study their behaviour using ML. In section 5.4 we present a novel attack against FunCAPTCHA that we demonstrate in section 5.5. In section 5.6, we discuss whether some potential improvements can help FunCAPTCHA cope with our attack and others. Section 5.7 comments on both the strength of FunCAPTCHA and the characteristics and potential of our attack.

5.1 FunCAPTCHA description

FunCAPTCHA is not the first CAPTCHA design to be based on image orientation and gender recognition (Gossweiler et al., 2009, Kim et al., 2014). However, it is the first readily available wide-scale implementation of a gender recognition CAPTCHA. FunCAPTCHA claims better strength and usability than a typical word-recognition CAPTCHA. More so, FunCAPTCHA decides to implement their genre recognition challenges using 3D synthetic images. This method has the potential benefit of control over all variables affecting the challenge creation, thus potentially rendering a secure CAPTCHA.

As ML facial recognition copes better with frontal pictures, this can be the reasoning that explains the design of FunCAPTCHA, that rotates the 3D heads and then renders them in 2D. Given this challenge generation algorithm, it is unknown how well current state-of-the-art gender recognition algorithms would behave in this scenario. We assume that the company behind FunCAPTCHA did some testing with current state-of-the-art ML algorithms.

FunCAPTCHA generates two different types of challenges, each one appearing roughly 50% of the time. The first type requires the user to rotate

an image in 40° increments until she puts it in its correct vertical orientation. This implementation is weak, as a brute-force attack would pass it $\frac{1}{9} = 11\%$ of the time for one test, 0,13% for challenges of three tests and 0,0016% for 5-test challenges. This idea is also not new (Gossweiler et al., 2009) and has known drawbacks (Zhu et al., 2010a) that make it of little interest.

The second type of challenges is a gender recognition challenge that presents a 9×9 tile box with 8 faces, one of them representing a female. It requires the user to select a picture of a female face among 8 images and drag & drop it to the centre of the tile box. Because of its novelty, this is the test that interests us and that we will study in this chapter.

Each one of this two types of CAPTCHA varies regarding how many tests are required to be solved sequentially to pass the CAPTCHA. In our tests, the whole CAPTCHA challenges have been comprised of either one, three or five individual tests.

FunCAPTCHA has implemented different versions of the gender recognition test over the time, as seen in Figure 5.1. We are aware of at least four different versions: using real human models, rendering different 3D facial models in 2D in colour, using only one model per gender in colour, and rendering in greyscale. It is unknown to us why FunCAPTCHA designers did these changes. In communications with FunCAPTCHA authors, they claim that they update their CAPTCHAs to stay ahead of the advances in ML.

Apart from its security, another main advantage according to FunCAPTCHA marketing is that it offers a significantly higher conversion rate than other CAPTCHAs, as "FunCaptcha has a 96% completion rate" and "is completed 28% more than twisty-lettered CAPTCHAs".



Figure 5.1: Different FunCAPTCHA gender recognition iterations.

5.2 FunCAPTCHA analysis

We analysed FunCAPTCHA from the viewpoint of an attacker that wants to bypass it as a means to gain automatic access to some rewarding on-line service. Thus, we did not register with the API of FunCAPTCHA, nor installed a client in our machines. We analysed its protocol directly from the browser, using HTTP analysis tools.

5.2.1 FunCAPTCHA initial analysis

It is correct to argue that a CAPTCHA with a 12,5% brute-force success ratio ($\frac{1}{8}$) is already flawed. The chances of passing the three and five-test challenge by brute-force would be 1,5% and 0,003% respectively. Only the later is good enough for a production CAPTCHA. FunCAPTCHA seems to rely on some tracking, possibly based on IP tracking, to decide when to *harden* the test after the user sends one or several wrong answers.

During our analysis, we found that FunCAPTCHA uses several obfuscation techniques. Among them:

- JavaScript code obfuscation at two levels.
- Cyphered communications, using the AES Cypher in Counter-mode for the transmission of some values. This is in addition to all transmissions using HTTPS.

- The order in which FunCAPTCHA presents the face images on the client's browser is also obfuscated.
- 2-level cross-domain IFrame nesting to prevent easy JavaScript debugging.

Each of these measures was rendered at least partially useless. This was possible after the following findings:

- It was possible to partially revert JavaScript code obfuscation, as a different JavaScript code was found thanks to caches using a less obfuscated version.
- FunCAPTCHA uses the AES library from Chris Veness³. Thanks to this finding, it was possible to decipher its communications easily. In particular, it was possible to see that the value of the parameter *guess* was being used to send back the answers to FunCAPTCHA after each drag & drop. Its value was ciphered using AES in Counter mode, initialized with a value partially time dependent and partially pseudo-random. This value was added to the message to allow for its decoding at the FunCAPTCHA server. The key used for ciphering was the *session_token*, passed from the FunCAPTCHA server to the client during the initial set-up of the test.
- The other three obfuscation measures were all bypassed by using a regular browser to analyse and later bypass the CAPTCHA. More details about this in section 5.4.

The reasoning for these obfuscation measures is not clear to us. Some instances, as encoding the answers using AES and a key already delivered from the server do not seem to serve any real purpose. Others are more a nuisance to some analysis than a real impediment to any attacker. The use of these obfuscation levels is a clear case of trying to implement Security Through Obscurity.

When we contacted FunCAPTCHA authors, they replied that in fact this is the case, but that "obfuscation is [an] asymmetrical effort" and

³This library can be found at <http://www.movable-type.co.uk/scripts/aes.html>. Even though this AES library is protected by a MIT license and requests a link to the original page and the original copyright notice, we were not able to find those in FunCAPTCHA's site.

that it is "surprisingly effective" at delaying attackers. We think that the attack we present in section 5.4 proves this not to be the case.

5.2.2 FunCAPTCHA image repository

After automatically downloading 500 images, we calculated their MD5 and SHA1 Cryptographic Hash functions, used here as mere fingerprints of the file contents. We found no coincidences. This result was somehow surprising, as many of the faces look quite similar to the eye.

This finding leads us to affirm that FunCAPTCHA does render the 3D model each time, using a slightly different angle, illumination and distance parameters, so that not two images are identical at the bit level. This initially looks like a sound implementation decision for FunCAPTCHA.

5.2.3 FunCAPTCHA protocol analysis

Even though FunCAPTCHA uses several obfuscation mechanisms, it was possible to relate its client-server communications to the different events happening in the browser. We were able to easily decipher the communications cyphered with AES and analyse the FunCAPTCHA communications protocol. In brief, it follows the following main steps:

1. A web-page that contains the FunCAPTCHA UI is loaded⁴. FunCAPTCHA creates dynamically the part of the page that contains the CAPTCHA, which includes an IFrame that loads another IFrame that contains the UI. FunCAPTCHA uses several dynamic parameters to create it, including *token*, *r*, *quitextcolor*, *metabgclr*, *metaiconclr*, *meta*, *surl* or *source-url*.
2. The browser loads then the IFrame using these previously created parameters, sending them to `https://funcaptcha.co/fc/gc/`.
3. In the loading process of the IFrame contents, one particular URL contains the additional references to the rest of the contents of the

⁴As `https://www.funcaptcha.com/contact-us/` or `https://www.funcaptcha.com/demo/`, both retrieved in September 2015.

challenge. This is a POST petition at <https://funcaptcha.co/fc/gfct/> (probably *gfct* for *get FunCAPTCHA test*).

The server answers with a full description of the challenge, including:

- The new variables *challengeID* and *challengeURL*.

There are two possible values for *challengeURL*: 001 indicates we are having an image orientation test, and 002 indicates a gender recognition test.

- The URLs of the images to download, included inside the *__challenge_imgs* variable.

These are always multiples of 8, as each test shows 8 images to the user. A full challenge can typically have one, three or five tests, so 8, 24 or 40 total images.

- The extra images for the "pick your favourite activity" screen, that has no security relevance.
 - Other elements including images with logos, additional texts messages that might be shown to the user in different scenarios, etc.
4. There are certain *events* that the client JavaScript notifies to the server. Among them we find: when the browser starts to display the challenge; when the user clicks on the *verify* button; if the user requests a different challenge, and others. The JavaScript at the browser posts these events to <https://funcaptcha.co/fc/a/>, and the typical server answer is:

```
{"logged":true}
```

5. If we are dealing with the gender recognition test, every time that an image is drag & dropped to the centre, the client JavaScript sends the information to the server using a POST to <https://funcaptcha.co/fc/ca/> (challenge answer).

The server replies with different strings depending on the case: if the user just sent an answer to a test that is part of a challenge; if the answer is wrong, or if it is correct:

```
{"response":"answered","solved":true,"incorrect_guess":"","score":3}
```

If the answer is wrong, the server returns the parameter *incorrect_guess* with the ordinal of the answer that was wrong. Providing this unnecessary feedback is not a sound idea, as it allows an attacker to know which tests within a full challenge have been correct, and thus, to correctly label a subset of the images of the challenge and gain knowledge for other attacks, for example creating a labelled training set.

We were able to programmatically intercept the communications between the client (the web browser) and the server using a proxy. This monitorization allowed us to determine what type of challenge we were facing -rotation or gender recognition- and also how many tests it was composed of. When we were dealing with a gender recognition challenge, we were also able to download the challenge images. Finally, it allowed us to easily know whether the answers sent to FunCAPTCHA were correct or not according to their servers.

5.3 FunCAPTCHA design flaws

At this point of our research, we could list some decisions of the FunCAPTCHA design that might be key to its security:

- It uses only one male and one female 3D model.
- The model does not show facial expressions, nor it includes other distortions, as the addition of glasses, different haircuts, etc.
- Even though the served 2D images do not repeat at the bit level, some of them *look* similar or very similar to images shown before.
- The background is always plain white.

Other characteristic is that the images do not have the same distance from the model. For example, some of the images include the shoulders, others show the neck partially, while others show mostly only the face. Visually, there is no obvious way to classify male from female pictures. The number of white pixels seems to be more affected by the distance than any other factor. Similarly, the amount of use of the different grey shades does not seem different depending on gender.

5.3.1 ML analysis of the flaws and strength

We employed a simple classifier with the aim of distinguishing male faces from female faces. We wanted to check whether the similarities of the FunCAPTCHA images would allow a classifier to efficiently detect male vs. female images if fed with very simple image statistics.

To test this hypothesis, we downloaded and manually classified 4320 images from FunCAPTCHA. Note that this was not strictly necessary. Due to the vulnerabilities in the design of FunCAPTCHA, it would have been possible to solve the 1-test challenges with a $\frac{1}{8} = 12\%$ success rate and use these solved challenges as a training set. Of those 4320 images, only 535 were images of females (not exactly 1 in 8 due to some time-outs during the downloads).

We extracted some very basic statistical information from these images: the percentage of white pixels; the histograms of the use of different grey intensities, in groups of 5, 10, 15 and 25 intervals; and the size of the image compressed with JPEG using different quality factors (from *quality*=0 to 100).

As an initial classifier for this test, we decided to use the k -Nearest Neighbours algorithm. k NN has little parametrization: the number of neighbours considered, how the weights are calculated and the algorithm to use for the search. k NN is a good representation of the idea of using similarities between examples to classify.

Another benefit of k NN is that it can also produce the previous known examples that are found to be *similar* to the one being classified. That way we can check if the metrics and distances are relevant for the classification we are trying to achieve.

We *trained* k NN using all the manually classified images. To test it, we downloaded additional 148 challenges, each one composed of five tests (with the exception of a few download errors). We proceeded with a semi-exhaustive search trying different values for k and the rest of the parameters. We ordered the results by their Cohen's κ statistic values, that measures a classifier against the expected accuracy. This metric is more relevant than the accuracy for such imbalanced data.

5.3.2 Results of the ML analysis

The best result was typically obtained selecting only the closest neighbour, reaching an accuracy of 97% and a κ statistic of 0.84 when tested on new images.

We run again our experiment selecting now the closest image to each unknown image. The result of this analysis can be partially seen in table 5.1. In this table the first six rows show a training image and the value of its different metrics, and the next six rows are a test image classified as pertaining to that *class* along with the same statistics.

The metrics in table 5.1, in order of appearance from higher to lower, are: the number of white pixels (% from maximum), the histogram of appearance of the different gray-scales, grouped in 5 bins, 15 and 25 bins, and the sizes of the image compressed with JPEG and different quality settings. Table 5.1 shows two wrongly and two correctly classified images by gender, and the closest one to each query.

Even though these simple statistics allow the correct classification in a 97% percent of the cases, there are several occasions in which they completely miss. Selecting more neighbours and averaging the resulting class, or weighting it by distance, is not the solution to these errors, as Table 5.2 shows.

A question that naturally arises is *how many* labelled faces does k NN need to perform at a good level. Or, put it another way, how many faces are enough to "have seen them all" (or most). For this reason, we experimented with k NN and different sizes of the training set.

To test this, we performed 25 experiments for different sizes of the training set (measured in % from the total training set size), each one using 5-CV, and calculated the mean and error margin of both the accuracy and the κ .

The result is shown in Figure 5.2, where the shadowed area is the margin of error at 95% interval of confidence. As can be seen, with just 0.05% or 250 images of faces for training we still can obtain a $> 85\%$ classification success ratio for a single image. This seems like a good result till we check the κ value for this classifier, which is 0.3, and we see in the confusion matrix that it has classified more females as males than as females. This is why accuracy

Table 5.1: Some FunCAPTCHA wrongly and correctly classified faces, and their statistics.





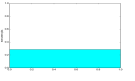
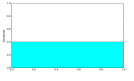
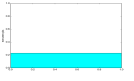
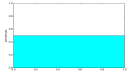

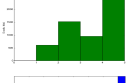
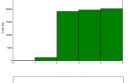
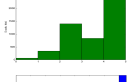
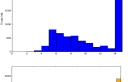
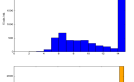
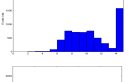
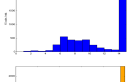
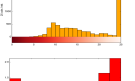
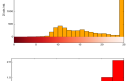
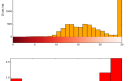
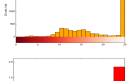
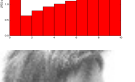
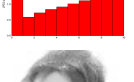
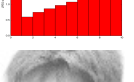
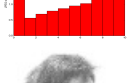
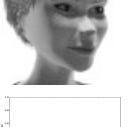

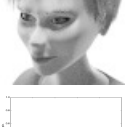
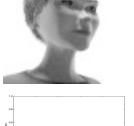

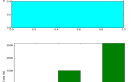
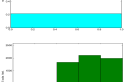
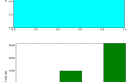
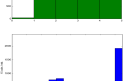
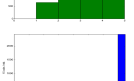

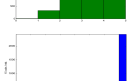
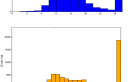
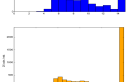
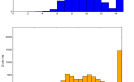
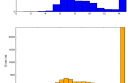
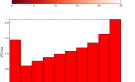
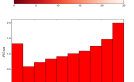
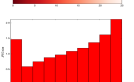
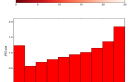








	Wrongly classified		Correctly classified	
Class (training)				
White pix. (%)				
Color histogram (5 bars)				
Color histogram (15 bars)				
Color histogram (25 bars)				
JPEG sizes				
Problem (test)				
White pix. (%)				
Colour histogram (5 bars)				
Colour histogram (15 bars)				
Colour histogram (25 bars)				
JPEG sizes				
Diff. img. ($\times 5$)				

Table 5.2: Classification success rates for different k NN parameters. The first column shows the search algorithm used. The second column shows if the classification is averaged or weighted by the distance. The third column shows the number of closest neighbors used to calculate the answer. The next four columns show the different elements of the decision matrix. The last column shows the corresponding κ .

Search algorithm	Weight of neighbours	Number of neighbours	Correct males	Correct females	Males as females	Females as males	κ statistic
auto	distance	1	5142	640	8	130	.89
brute	distance	1	5145	631	9	135	.88
ball_tree	uniform	1	5121	651	13	135	.88
brute	distance	2	5127	634	14	145	.87
kd_tree	uniform	1	5106	651	13	150	.87
...							
brute	uniform	3	5108	578	6	228	.81
kd_tree	uniform	3	5117	571	5	227	.81
ball_tree	uniform	3	5127	554	6	233	.8
kd_tree	distance	5	5092	550	2	276	.77
brute	distance	5	5116	527	1	276	.77
...							
kd_tree	uniform	50	5180	16	0	724	.04
kd_tree	distance	50	5197	8	0	715	.02
ball_tree	uniform	50	5189	8	0	723	.02
auto	uniform	50	5187	8	0	725	.02
brute	uniform	50	5182	8	0	730	.02

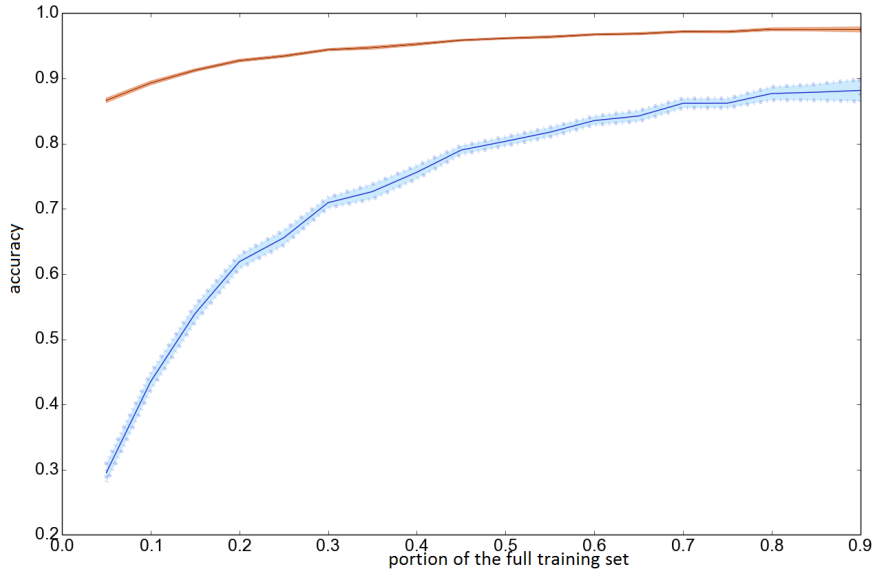


Figure 5.2: Success rate of the kNN classifier when using smaller training sets than the full one.

is a weak performance evaluator when used with such an unbalanced set, and we have to restrict ourselves to using the κ statistic. With that restriction in mind, we can still obtain decent values of the κ statistic of 0.8 for 50% of the original training set (1954 images).

As a summary of our findings, we can say that even though there is no evident way to classify the gender of the images based on the metrics we have selected, a very simple ML algorithm is able to do so with great accuracy with as few as 8754 images in the training set, even when it is quite unbalanced. This means that even though the strength of FunCAPTCHA seems ok to the untrained eye, it is actually not able to provide a strong-enough ML problem. We wonder now whether other more sophisticated ML algorithms will present a better outcome, and might be used in a real attack scenario.

5.3.3 Machine Learning attack parameters

We know that FunCAPTCHA is not strong enough against simple metrics and a simple ML algorithm as kNN. This means that FunCAPTCHA challenges do not show the maximum strength that the gender recognition problem can

have. Far from that, they present some simple similarities that are simple to identify even using a simple ML algorithm that does no heavy image processing nor image recognition.

The results obtained so far allow for some accuracy in the classification, but this can be countered by using more faces and/or a greater number of challenges, among others. We want to know if other ML algorithms can cope possibly even better with the gender classification problem proposed by FunCAPTCHA.

To try other algorithms, we checked the use of different ML frameworks that allow the use of several ML classifiers and have some integration with *Python*. In particular, we looked at Orange and Weka (Hall et al., 2009). We decided to use Weka because of the many more classifiers that Weka has out-of-the-box (79 vs. 11 in Orange).

As we also did in section 4.4.1, we compared all compatible Weka classifiers using 5-CV. The selection of the best-performing algorithms was done using the Cohen's- κ metric. This metric behaves better than other metrics for very imbalanced training sets such as the one we have here, with one image of a female per seven images of males.

The results of these tests are available in Table 5.3. This table shows the best and worst 12 performers of the whole set. It turned out that the multilayer perceptron, IB1/k, KStar, and tree-based algorithms are the ones that perform best.

It is interesting to see that while some ML algorithms can cope out-of-the-box with unbalanced data, there are a few that completely fail with such an unbalanced training set and decide to classify all pictures as males.

The best value obtained for the κ statistic is 0.96 (99.19 accuracy), a much higher value than the maximum obtained before with the kNN (.89). This good result allows us to envisage a potentially successful attack to the gender recognition challenge of FunCAPTCHA.

Table 5.3: Best and worst classifiers for off-line gender recognition with FunCAPTCHA.

Algorithm	Correct (%)	κ statistic
MultilayerPerceptron	99.19	0.96
KStar	98.94	0.95
IB1	98.91	0.95
IBk	98.91	0.95
LMT	97.73	0.89
Logistic	97.59	0.89
MultiClassClassifier	97.59	0.89
SimpleLogistic	97.43	0.88
FT	97.36	0.88
SPegasos	97.43	0.88
Decorate	96.85	0.84
SMO	96.83	0.84
...		
VotedPerceptron	88.63	0.13
RBFNetwork	88.17	0.13
LWL	87.75	0.03
ClassificationViaClustering	55.56	0.01
DMNBtext	87.71	0.01
BayesianLogisticRegression	.	0
Grading	87.64	0
MultiBoostAB	87.64	0
MultiScheme	87.64	0
ConjunctiveRule	87.64	0
ZeroR	87.64	0
DecisionStump	87.64	0

5.4 Machine Learning attack to the FunCAPTCHA

Once we determined the effectiveness of the ML classifiers for bypassing the different challenges presented by FunCAPTCHA, we needed to assess the strength of its design.

For that purpose, we created an attack that comprises the following steps:

1. Start a local proxy for the HTTP and HTTPS protocols. We use the *proxyp* Open-Source proxy .
2. Open a web-browser (Mozilla FireFox) and direct it to the web-page at <https://www.funcaptcha.com/contact-us/>. We control this browser instance thanks to the Selenium library (Huggins and Hammant, 2014). This web-page contains the FunCAPTCHA CAPTCHA at its bottom. We decided not to use the web-page at <https://www.funcaptcha.com/demo/> because we noticed frequent changes in it during our analysis, including a period of over a month during which the demonstration challenge was not available.
3. After we initiate the request, we wait for the proxy to capture the value of the *challenge_url* variable that indicates if we are facing an image orientation challenge or a gender recognition one.
 - (a) If FunCAPTCHA is serving an image orientation challenge (*challenge_url* = 001), we restart the process, unless we have done it two times already, in which case we wait a random time in between 25 and 115 seconds⁵.
 - (b) If we are served a gender recognition challenge (*challenge_url* = 002), we read from the answer how many images it is composed of by looking at the contents of the array variable *image_urls_str*.
4. We wait till the browser downloads all the images.

⁵This waiting time interval was chosen because it was seen that too small waiting times lead to increased chance of being served 001 challenges. After e-mail exchange with the FunCAPTCHA designer, he confirmed that they use some sort of IP-based reputation system. Even though the details were not disclosed, we have observed that requesting another challenge too soon leads to increasing chances of it being of the same type as the last one.

5. We run one of the classifiers over each one of the sets of 8 images (one, three or five sets or tests). We use the Weka ML framework and the previously trained models. We check that for each set, one and only one image is classified as a woman.
 - (a) If the classifier fails to do so, that is, does not classify one and exactly one as a woman in each group of 8 images, then the challenge is declared failed. We consider this both a classification failure and an attack failure. A log is saved, and the process starts again. Note that we can improve this step using the reported accuracies from the classifiers, but decided not to for clarity.
 - (b) If the classifier classifies one and only one image of each set as a woman, we proceed to send the answers to the server.
6. To send the answers to each test of the challenge:
 - (a) We look for the solution face on the screen using the SWIFT algorithm implemented in the OpenCV library.
 - (b) We drag & drop the face to the centre of the challenge using the *pyautogui* library.
 - (c) We wait for the answer from the FunCAPTCHA server. It could be:
 - "not solved": we proceed to send the next answer.
 - "solved:false": we log the challenge as failed, both for the attack and the classifier.
 - "solved:true": we log the challenge as correct.

Figure 5.3 shows a summarised flow chart of this attack. All steps of the attack have a set time-out that, when reached, would declare that challenge as failed and restart the process.

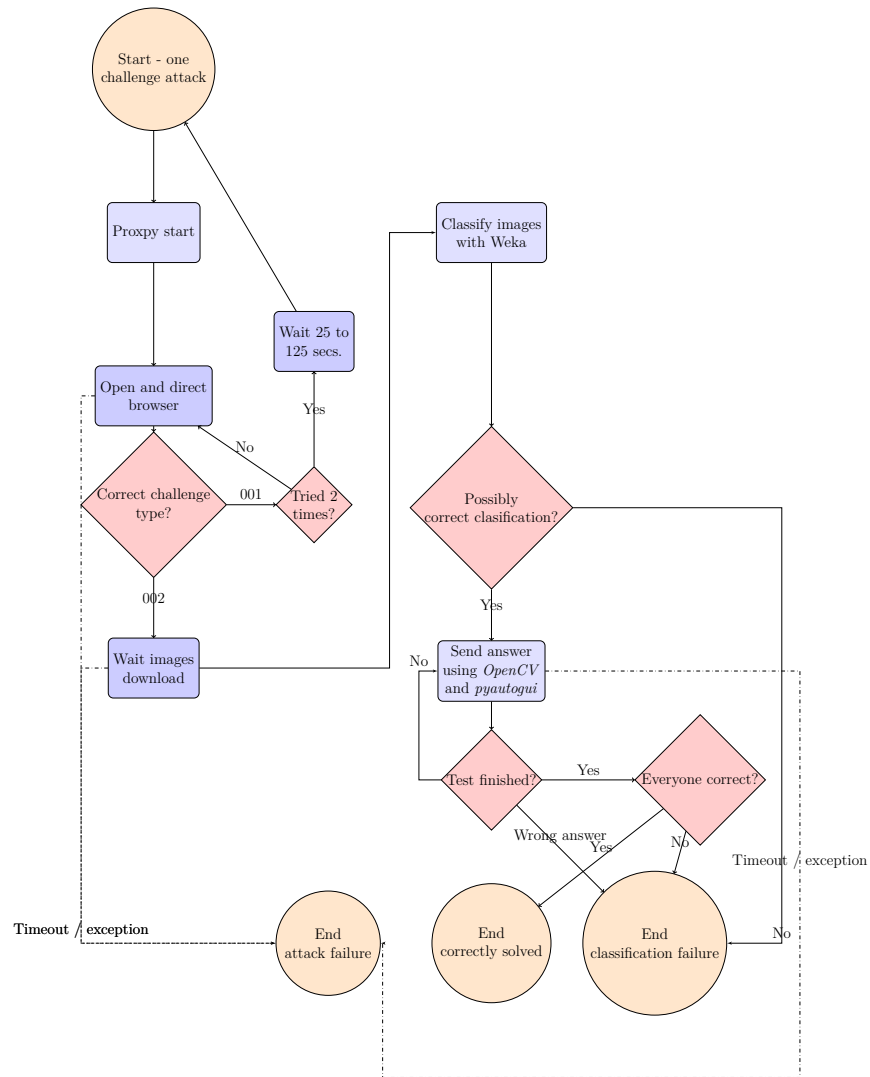


Figure 5.3: Flow chart of the attack to FunCAPTCHA.

5.5 Experimental results

We ran our attack using the classifiers that performed best on our off-line classification test, and also with the original k NN implementation. We noticed that in the cases when the attack kept solving correctly the gender recognition challenges composed of only one test, FunCAPTCHA almost never served us the more difficult 3-test or 5-test challenges. For this reason, for each ML algorithm picked, we ran each experiment in two versions:

- The regular one (that we will call *basic*), trying to solve all gender recognition tests presented to us by FunCAPTCHA.
- The *hardened* one, in which we randomly answered all 1-test challenges (failing most of them), to receive more 3 and 5-test challenges.

Given the set-up restrictions on speed as not to overload the servers, our experiment consisted of various series of around 255 full challenges for each one of our experiment configurations. The total of 255 challenges was very seldom reached, as we frequently run into timeouts, errors downloading information, or problems with the iteration on-screen.

Table 5.4 presents the success rate of the attacks to FunCAPTCHA by different classification algorithms. In this table, the first column contains the Weka classifier name. The second column shows the classifier accuracy during the attack, counted per groups of 8 images (thus the accuracy per image is higher). The third column shows the success rate of the attack itself. The classifier accuracy during the attack is measured per complete challenge. This means it is not differentiating between 1, 3 or 5-test challenges.

FunCAPTCHA is typically going to serve to us more 3 or 5-test challenges the more 1-test challenges we fail. Because of this behaviour, a slightly worse classification rate in the 1-test challenges triggers a feedback mechanism that can have a major effect on the statistics.

The first half of Table 5.4 shows the success rate of our attack in the current FunCAPTCHA implementation, that is, as it was in the moment of the attack. The second half of Table 5.4 answers the question "what would be the success rate of our attack if FunCAPTCHA used only the harder 3-test or 5-test challenges?" It presents the success rate of the attacks to FunCAPTCHA when using different classification algorithms. We can see that

Table 5.4: Success rates by classifier, for the basic and hardened attack.

Basic attack					
Classifier	% Classifier	% Attack	number of n -test challenges		
			1	3	5
IB1	$94,02 \pm 0,02$	$90,42 \pm 0,03$	448	58	0
KStar	$93,15 \pm 0,03$	$89,19 \pm 0,04$	252	1	0
IBk	$92,61 \pm 0,03$	$88,15 \pm 0,04$	264	98	0
MultilayerPerceptron	$94,68 \pm 0,03$	$85,27 \pm 0,04$	266	6	1
Logistic	$77,3 \pm 0,05$	$76,05 \pm 0,05$	248	51	9
FT	$80,59 \pm 0,05$	$72,9 \pm 0,05$	251	2	0
kNN	$55,65 \pm 0,06$	$54,07 \pm 0,06$	70	69	107

Hardened attack					
Classifier	% Classifier	% Attack	number of n -test challenges		
			1	3	5
MultilayerPerceptron	88.35 ± 0.03	82.69 ± 0.04		110	46
IBk	83.07 ± 0.04	72.97 ± 0.05		98	48
KStar	75.83 ± 0.05	62.43 ± 0.05		116	47
IB1	53.63 ± 0.04	29.35 ± 0.04		72	255
FT	38.70 ± 0.05	28.98 ± 0.05		125	48
kNN	36.80 ± 0.05	23.71 ± 0.04		72	119
Logistic	24.18 ± 0.03	18.20 ± 0.03		236	132

the *MultilayerPerceptron* and the *IBk* are among the top overall performers. We can also see that the difference in success rate between classifier and attack is higher than in the basic attack, as each challenge now involves more communications with the server and thus is more prone to errors.

Figure 5.4 shows a combined result of both attacks, summing the results obtained during both the *basic* and *hardened* settings in order to obtain more 3 and 5-test challenges. The bars indicate the success on a scale from 0% to 100% for each subtype. Each bar is divided in two: the classifier success identifying the correct one and only one woman in each of the n groups of 8 images for the entire challenge, and the attack success for the whole n -test challenge. Along with each bar, we show the confidence interval, estimated for a binomial distribution using the Wald method. The multi-layer perceptron can solve 94.53% of the 1-test challenges, 91.23% of the 3-test challenges and 82.05% of the full 5-test challenges (68.09% attack success). Even if FunCAPTCHA decided now to use only their most secure 5-test challenges, this attack would break their CAPTCHA 68.09% of the time.

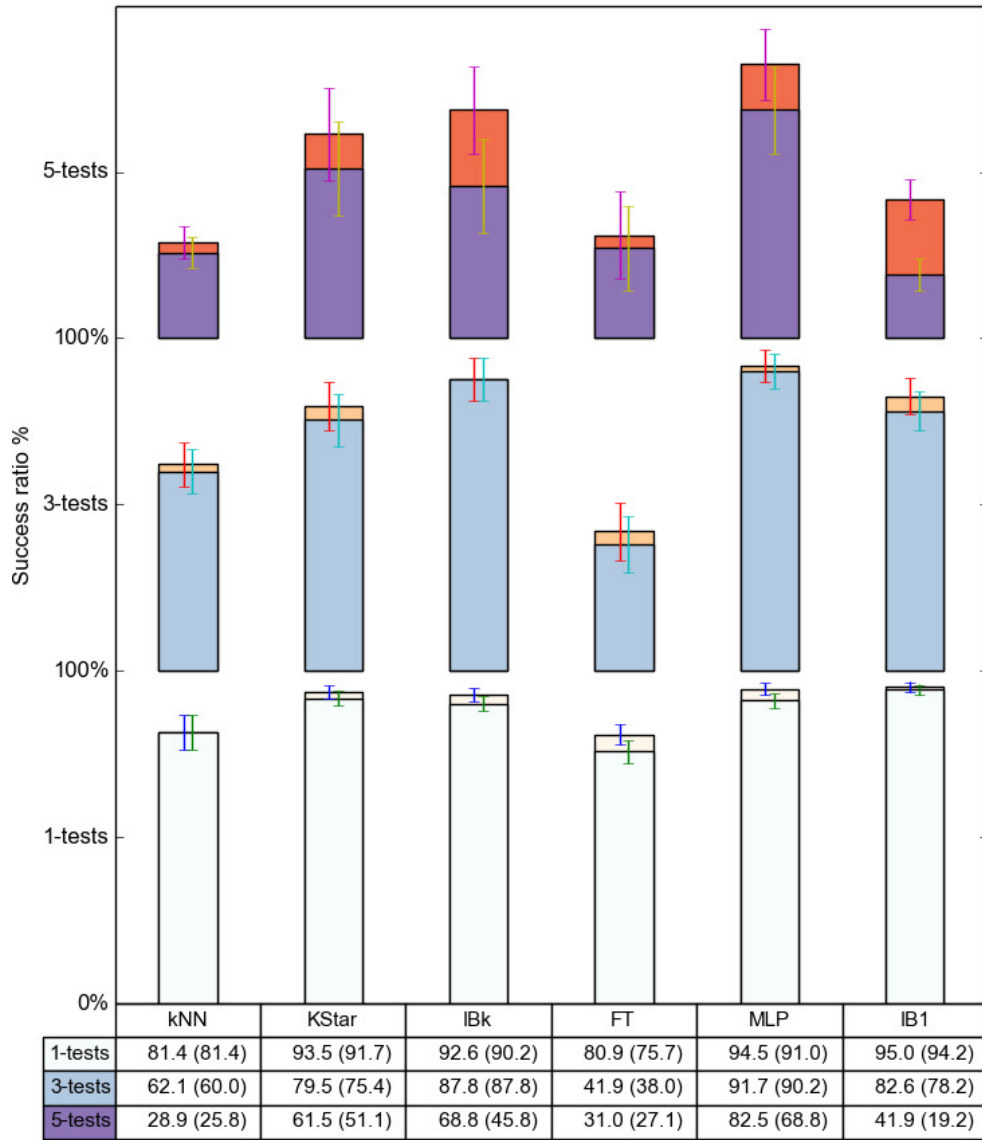


Figure 5.4: Success rate by classifier and challenge type, for both the *basic* and *hardened* attack. Each column corresponds to one classifier. There are three bars per classifier, one per type of challenge (1, 3 and 5-tests). These bars are subdivided each in classification accuracy (for 8-images tests) and attack success rate (lower, as it includes any additional problem during the attack). They show the corresponding confidence interval at 95%. The table below shows the same information numerically. The numbers are the classification success for the whole (1/3/5) 8-images tests, and the numbers between parenthesis are the attack success rate for the same challenges.

5.6 Possible improvements

In this section, we will discuss some possible improvements to FunCAPTCHA, both in general and in particular against this attack.

- **Answer space:** FunCAPTCHA should never serve 1-test or even 3-test challenges, only challenges composed of 5-tests. This is the only viable option to make it resilient to brute force attack, stopping the attacker from obtaining automatically labelled images. Unfortunately, even in this case, our attack can break the 1-test challenge 91% of the time. Thus, to reach a success rate lower than 0,6% (Zhu et al., 2010a), we would need to repeat this test $\log_{0,91}(0,006) \approx 52$ times.

Naïvely, we can think that another option for the FunCAPTCHA authors would be to show more possible answers in each test. As our best classifier is able to correctly differentiate the gender 99,19% of the time, that is, correctly solve a 8-image test $99,19^8 = 93,7\%$ of the time (actually it is 94,5%, but here we are extrapolating using our off-line results), we would need to have $\log_{0,9919}(0,004) \approx 629$ faces from which to pick one female. That seems a little bit too much from a usability point of view.

- **ML Analysis:** There are some ways to try to prevent the ML attacks that we have presented here. An obvious one would be to use a much larger number of models, along with other influencing factors as clothing, eyewear, facial expressions, etc. These additions have the potential to allow for a bigger chance of *collision* of the statistics. The models themselves can be studied using the commented metrics and ML algorithms to discard those that are too easily classified automatically.

Model rendering parameters could also have a wider range. It might be possible that there exists a sweet spot in the rendering parameters (angle, light, etc.) in which ML classification does not perform well while human classification still performs well due to a number of reasons (clues about hair, etc.).

It is also possible to include measures to distort or homogenise the result of basic statistics from the images (i.e. histogram of grey scales). The aim would be to render the most common and/or trivial statistics completely useless for ML classification.

- **Resilience:** Nothing prevents the authors of FunCAPTCHA from having new models in their reserve to make A/B tests, either in general or

against a particular client. This would allow not only to automatically detect attacks, but to repel them in real time.

If a large-enough number of models is present, this could mean that in reality, the CAPTCHA would be able to detect and adapt to many unknown attack scenarios.

Even using all the previously mentioned means, it is unclear to us at this point whether these measures would render this particular CAPTCHA secure. After a new redesign, a full new security analysis should be done. Even if the redesign can cope with this attack and variants of it, it is certainly unclear whether this subset of the gender recognition problem would be secure against the recent advances in image recognition, more precisely using Deep Convolutional Neural Networks.

5.7 Discussion

In this chapter we have analysed the security of FunCAPTCHA. It is the first CAPTCHA to our knowledge that implements the idea of gender recognition as the basic way to tell computers and humans apart and reaches a production phase.

Even though ML is currently good at extracting different information from faces (identities, gender, expressions, etc.) there are known cases in which this is extremely tough, and the success rates are low. FunCAPTCHA uses synthetic images, allowing them to control all the characteristics of their challenges. They claim to have a large number of clients. We analyze its implementation as of from July to October 2015. The authors of the CAPTCHA claim it to be broadly used, never broken, and with a high security level and conversion rate.

We analyze its security using both in traditional and novel ways and find what might be possible weaknesses in its design. Using well-known ML algorithms and extremely simple image metrics, we see that is possible to solve the subset of the gender recognition problem proposed by FunCAPTCHA. We confirm this through an attack that is able to bypass FunCAPTCHA 90% of the time. Even if the authors of FunCAPTCHA would use only their most difficult set-up, requiring 5-test challenges correct, our attack would be able to bypass it at least 68% of the time.

This is an unexpected result given the apparent complexity of the problem and the simple attack methods used. This attack uses no technique that can be considered image analysis, yet efficiently bypasses FunCAPTCHA. We conclude that it is not necessary to attack it by following the intended path of attack. We present some possible ways to partially solve these design flaws.

Checking a CAPTCHA challenge domain and answer domain can give significant information to an attacker. Also, using well-known metrics, we saw that some ML algorithms can solve the CAPTCHA a significant number of times, thus rendering it useless.

This result together with the ones from the previous chapters elicits a pattern of attack that can be useful to someone that wants to test if a new CAPTCHA proposal fulfils a basic security level. The most important aspects of the attack path are similar, and only some parts can be slightly tailored to each particular case. We consider that these analysis guidelines could constitute a methodology for security test. Given these results, in the following chapter we propose a methodology to assess a basic level of security for CAPTCHAs based on these ideas.

Chapter 6

BASECASS: A framework for BAsic SEcurity CAPTCHA ASsessment

In the previous chapters we have studied the security of state-of-the-art CAPTCHAs. All of them have been found vulnerable to attacks. The attacks found have certain common attributes that also appear in other attacks in the literature (Yan and Ahmad, 2007, 2008, SEO, 2008 a,b , Santamarta, 2008, El Ahmad et al., 2010, Zhu et al., 2010 b , Hernandez-Castro, Ribagorda and Saez, 2010, Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda, 2010, Hernandez-Castro et al., 2011, Mohamed et al., 2013). This suggest the possibility of creating a procedure to check the security of new CAPTCHA proposals that would be based on the common attributes previously mentioned. This security assessment can check that a new CAPTCHA meets a minimum level of security by checking whether its challenges *leak* enough information for a simple side-channel attack.

Based on these observations, in this chapter we introduce BASECASS, our proposed framework for testing that a new CAPTCHA proposal (design and implementation) meets a basic security level. This framework is the result of our security case studies and the research literature comprising other security case studies related to other CAPTCHAs. From now on, we will refer to our proposed framework as BASECASS, a framework for BAsic SEcurity CAPTCHA ASsessment.

First, we clearly state the objective of our framework (section 6.1).

Before going into detail, we give to the reader an introduction to BASECASS (section 6.2) that although optional, is recommended to read before the detailed description, presented in section 6.3. BASECASS is divided in three main steps, presented in sections 6.5, 6.6 and 6.7. The information gained from the application of BASECASS can be summarised in a table, that we introduce in section 6.8. Even though we present examples of the different parts of BASECASS while we introduce it, they are partial, covering only specific sections of BASECASS, like the domain analysis, the selection and creation of metrics, or the application of S/ML algorithms. Section 6.9 presents full examples of application of BASECASS to our previous case-studies, in order to validate whether BASECASS is able to find the previously found weaknesses. In this section we also apply BASECASS to two other CAPTCHAs that appear in the attack literature. Finally, section 6.10 summarises our findings and presents the main conclusions from this chapter.

6.1 Framework objective

The target of BASECASS is to partially assess the security of any new CAPTCHA proposal to check that it meets a minimum security level. Note that to completely assess the security lies beyond the target of this dissertation, and it is in general something difficult to prove empirically, if not impossible, and only possible to do in a formal way. This is something no one has done yet for any CAPTCHA, possibly because of the reasons explained in section 2.1. In IT Security, a security assessment typically will cover some limited aspects of a threat model, as well as a vulnerability analysis will not cover all possible attack scenarios, but instead search for the presence of well-known vulnerabilities and their variants.

Our framework is designed to check that a new CAPTCHA proposal does not have typical side channel attacks. This means that it does not leak *enough information* in a way that would let an attacker solve the CAPTCHA frequently enough, without the need to solve the base problem which the CAPTCHA is based on. This is a necessary condition for the proper transfer of the problem difficulty (and thus, security strength) from the base problem to the CAPTCHA design and implementation. This is not a sufficient condition though.

The objective of BASECASS is to leverage currently widespread technologies applicable to CAPTCHAs, in a semi-automated way, in order to assess that they do not suffer from well-known design flaws. This way, BASECASS provides certain minimum-level criteria for CAPTCHA security assessment.

In some cases, even if the application of BASECASS does not render vulnerabilities, it can hint to additional insight on the strength of a new CAPTCHA design. This can be so in the case that some of the tests are passed, but show potential weaknesses. This will be a symptom of possible future problems with such design if the attack techniques are further refined.

As an example, during an analysis, we might learn that a CAPTCHA gives away information to correctly classify its instances 45% of the cases, but as it requires e.g. 12 correct classifications. This would only lead to 0.0068% success ratio for an attack. It is not troublesome *per se*, but if a further refinement of our technique, or further weaknesses in the design that leverage this one, allow us to slightly increase our correct classification rate, the CAPTCHA would be broken. Even if our framework does not break a particular CAPTCHA, it can lead to meaningful insight in its strength. Some additional insight can be gained, including how the different security measures that are present in a CAPTCHA collaborate in its strength. For example, if we analyse the variables that affect the formation of a challenge, we can see how they affect the information available for a possible side-channel attack. We can learn which variables and values offer better security, and avoid weak values for them.

6.2 Introduction to BASECASS

The idea of BASECASS is to apply a series of partially-customized steps to analyse a particular design trying to find some possible vulnerabilities. In that sense, it is related to a vulnerability assessment or a penetration test. A vulnerability assessment will typically look only for well-known vulnerabilities in a semi-automated or automated way. In a penetration test, the testers will additionally look for variations in these vulnerability types. The pen-testers will try to find variations of them, using their previous knowledge of the system, the security measures in place, and the typical vulnerability scenarios.

Our framework proposes an analysis that lies closer to a penetration test. In it, the tester will have to apply her knowledge of previous CAPTCHA side-channel attack techniques, but also propose the use of possibly known useful metrics, and possibly come up with new ones which are variants more suitably tailored to the particular CAPTCHA being analysed.

The main difference between our framework and a typical penetration test lies in the particular steps we propose in it. In our case, these steps are tailored specifically for analysing CAPTCHA designs, and are generic, and thus applicable to most designs.

Our framework can be divided in three main steps or iterations: a black-box basic security analysis of the CAPTCHA, an additional analysis based on Statistical Analysis and/or ML, and a parameter-related Statistical Analysis and/or ML analysis. Depending on the CAPTCHA type, the third iteration might not be possible, as it will require further insight or access into the CAPTCHA design. If it is possible, it will typically provide more accurate information about the minimum security level of the CAPTCHA.

We will use the same analysis tools in the last two steps. Thus, we call each step an *iteration*, as the main difference between both is how much internal information on the CAPTCHA design is available and thus able to be analysed.

Next, we will give a brief overview of the different BASECASS steps: the challenge and answer domain analysis, the statistical/ML analysis, and the parameter-based analysis. After the reader has an idea of what each step does, we present them in detail.

Step 1. Black-box basic security analysis

BASECASS starts by doing a Black-Box basic, initial security analysis of the CAPTCHA. This is an external analysis, based only on public information. During it, we will not pay attention to possible clues about the challenge design. In a general way, our Black-Box analysis can be divided into the following steps:

Phase I Automatic interaction: the objective of this phase is to develop a way to interact semi-automatically with the CAPTCHA. We want to do so

in order to download challenges from the CAPTCHA, send the possible answers to the CAPTCHA server and receive its answer, so we can grade the answers.

Phase II Analysis of the challenge space: in this phase, we try to know what types and subtypes of challenges the CAPTCHA presents. For example, a CAPTCHA can present two different types of challenges: OCR and image-based challenges. The subtypes that it presents can be heavily distorted words or sentences (for OCR), and image classification and reconstruction (for the image-based challenges). We are interested into establishing what possible different challenge types are easily distinguishable by a bot. We will relate these subtypes to the base problem that the CAPTCHA is theoretically based on. Is the base domain easy to explore for a bot? If it is possible within a reasonable cost, we will also want to check statistically their distribution to search for deviations from uniform. When possible, we also compare its size to the size of the base problem of the CAPTCHA.

Phase III Analysis of the answer space: this phase focuses on checking the size and distribution of the possible answers to the challenges. Note that not always it will be possible to explore this space automatically. We might need to solve a number of challenges to study the distribution. This might be within reasonable costs or not depending on each case. Following with the previous example, we would like to know if all words or sentences are possible solutions for the OCR CAPTCHA, and what classes are used in the image-based CAPTCHA. We want to check their distribution, both globally and per challenge type. Are there any deviations from the uniform? If so, are they severe enough as to allow a successful attack?

Figure 6.1 represents the part of the phase I that interacts with the CAPTCHA in order to collect the necessary data for the analysis that takes place in phases II and III. The first part detects and downloads the different types of challenges, and estimates their number by calculating the percentage of them that have already been seen using statistical methods like Mark & Recapture (Seber, 1974). The second part uses human input to reply to a number of challenges enough to later check their distribution. This is done for each challenge subtype that we want to study.

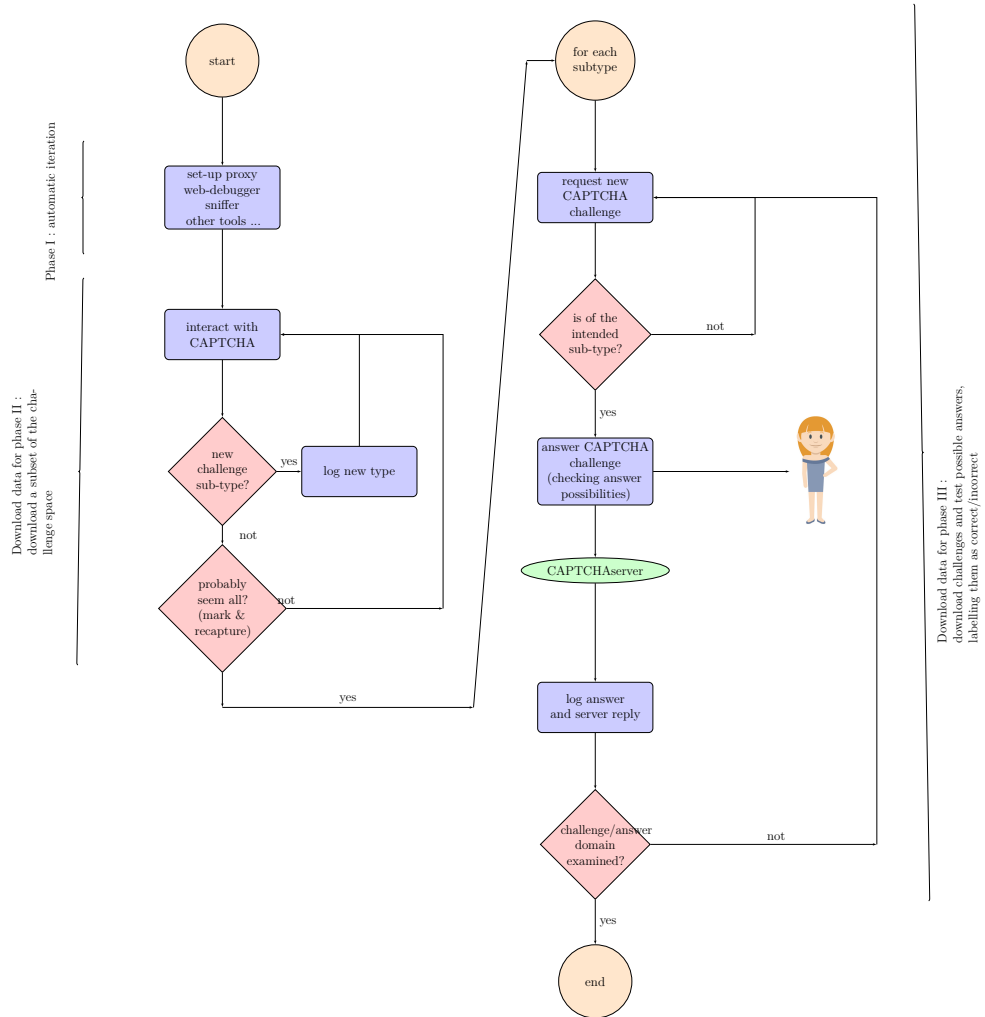


Figure 6.1: Generic flow chart for downloading the data needed for the Step 1 of BASECASS. This flow chart encompasses phase I. The data gathered will be analysed in phases II and III.

This black-box basic security analysis (Step 1) would render at least answers to the following questions:

1. What types and subtypes of challenges does the CAPTCHA present? What parameters affect when they are served to the user?
2. How many different challenges per subtype are there? If infinite, what is their domain?
3. Do all seem equally difficult both for a human and a machine?
4. How many possible answers are there for each challenge subtype?
5. For both the challenge space and answer space, are they uniformly distributed? If not, what are the deviations?
6. Is it possible to automatically detect challenge subtypes? If so, and if one of them is easier, is it possible to break the CAPTCHA at this point?
7. How is the communication with the server, regarding the grading of answers?

During this analysis other questions might rise giving further insight into the CAPTCHA: even if the domain and answer sizes are big enough, and their distribution is uniform, it is possible that we might find hints at some weak correlations between characteristics of the challenges and their correct answers. The next step deals with these kind of weaknesses.

Step 2. Black-box S/ML analysis

The previous step was our "first encounter" with the CAPTCHA. If it resists this basic analysis, we can move forward to the following step, that comprises a semi-automatic analysis of the side-channel statistics referred to the challenges.

In order to proceed, we will typically need to focus on one or a few of the subtypes of challenges served by the CAPTCHA, if there are many. This is so because possibly not all statistics will have sense for the different sub-challenge types. We will nevertheless focus on a subtype or subtypes that

comprise a significant amount of the challenges served, as it would be useless to break them otherwise.

The analysis presented in this step would render at least answers to the following questions:

- Is there or are there a metric or metrics that are somehow correlated with the answer of the challenge?
- Is this possible correlation linear (if the SA is successful) or not (only ML is successful)?
- Is it possible to explain this correlation in a human-understandable way? (Will depend on which ones are the most successful ML techniques)
- Is it possible to predict the accuracy of our correlation? Ie., it corresponds to some challenge subtype that can be classified by our metrics.
- Is this correlation possibly strong enough to base an attack on it?
- Which metrics contribute more to the correlation?

This step has four clearly defined phases. In the first one, we will prepare the challenges for processing. In the second phase, we select and/or create metrics (statistics) that are potentially useful to characterize the challenges. In the third phase we will use these metrics, together with the previously saved challenges and answers, to analyse the CAPTCHA statistically. This phase is optional. The fourth phase uses again the same metrics to analyse the CAPTCHA using different ML algorithms. A more detailed description of these phases follows.

Phase I. De-noising In some cases, a CAPTCHA designer might try to protect the information on the challenges by adding to them different types of *noise* or *distortions*. Sometimes, these can affect many of the metrics we can use on them. In these cases, we can think about de-noising techniques that might eliminate or minimize the influence of that noise in the metrics. Note that this phase is interrelated with the next phase, so they are complementary and not necessarily sequential.

Phase II. Pre-processing and transformations In some cases, we can think of a different domain in which the challenge might be easier to analyse. A typical case could be transforming an audio CAPTCHA from the time domain (wave) to the frequency domain using a FFT, or similarly transforming an image to a 2D frequency domain. Even though BASECASS does not emphasize to create anything like features to later analyse the challenges and answers, these kinds of transformations can be useful in some cases. This is something that should be done within the constraints of a low-cost attack.

Phase III. Definition of metrics For the selection and/or creation of statistics for the selected(s) challenge subtype(s), we will proceed as follows:

1. Selection of basic statistics: this step is done after we have examined a fairly broad subset of the CAPTCHA domain. Then, we will be able to select statistics that can be applied to the challenges. These will be general, broad sense statistics, that can be applied to the challenges in order to extract some information from them. The statistics will depend on media type, as they will be different for CAPTCHAs based on text, images, audio, or games. As an example of such general statistics, we can mention the randomness metrics returned by the ENT test applied to a binary file. These general metrics that can be applied to a very broad type of challenges, for instance, image challenges, to which we can also apply histogram of colour usage, pixel count, etc. These general metrics will depend on the media type of the CAPTCHA challenges, and on little else.
2. Selection of tailored statistics: in this step we select additional statistics that are more related to the CAPTCHA contents. For example, if it is a CAPTCHA based on images, then a statistic showing the quantity of image information can be useful. These statistics should be well-known for the CAPTCHA type or low-cost to obtain, having been previously defined. We are not interested in performing a full-blown CAPTCHA analysis here that will extract extremely significant, high-level information.
3. In-challenge relational statistics: this is an optional step. In-challenge relational statistics are those that relate different metrics obtained for different answers. If, for example, a challenge has 10^5 possible answers, instead of (or additionally to) giving the value of one of the metrics for those 10^5 possible answers, we can give the (for example) relative

order of those values, so that way the statistical or ML algorithm will know if this solution has the lower (or top) value among the possible solutions for that challenge. These statistics are useful to relate the possible solutions of a single challenge among themselves. This might be useful or not depending on the CAPTCHA type. For example, a value of a metric of 155 might be good for an answer to a challenge but bad for another challenge. But knowing that value is the lowest among all possible answers (or highest, depending on what we are looking for) might provide much more information. As explained, a typical way of doing this would be by ordering some of the previously extracted statistics within the possible answers to a challenge, and then registering this relative order, either absolutely or by percentiles.

Selecting and/or creating the metrics is a phase that requires some experience, as it is not fully automatic. Yet, in this phase we will use some general guidelines, which broadly speaking can be:

- Previous literature about well-known side-channel attacks. The detailed description of BASECASS provides a review of metrics used in the literature.
- Randomness metrics that can be applied to the challenge type. Among these, and of special interest, are cryptographic tests of randomness.
- Low-cost metrics: metrics that are already implemented and easy to use. These are typically extracted from libraries that can manipulate the media formats that contain the challenges (text, images, audio, video, ...).

This is an important phase, as the efficiency of the following S/ML analysis depends on it, so it is worth investing some time on it. If we cannot come up with any possible metric, we can just use the well-known ones for a basic security check. If possible, trying new metrics (always based on readily available software or procedures) can lead to interesting results.

When we have both the metrics and some correctly and wrongly solved challenges, we can proceed to the Statistical and ML analysis phases, which constitute the first iteration of BASECASS.

Phase IV. Statistical analysis (Optional) and ML analysis We will try to find correlations among challenge data extracted using our metrics and the solutions. To do so, we will apply statistical analysis techniques. If we skip this analysis or if we do not obtain positive results, ML techniques might be suitable. From an attacker point of view, we can skip the statistical analysis and proceed directly to a ML analysis, that renders more powerful tools than the statistical analysis, as some ML algorithms are able to automatically cope with non-linear classification and/or heavily unbalanced data sets. Yet from the point of view of a CAPTCHA designer, this step could be interesting, allowing us to learn significant statistical correlations that can clearly explain possible weaknesses.

From an attacker point of view, a ML analysis has the potential to provide for the most interesting results. For the ML analysis, we use the previously solved challenges, and the metrics data extracted from them to try to find a relation among the challenges and their correct answers. We do so using ML algorithms to look into the data trying to find significant patterns. We will try different families of ML algorithms with default parameters to search for the one that finds stronger relationships among challenges and their answers. In a second step, we can grid-walk its parameters to fine-tune the ML algorithm to obtain the best possible result. During this step, we will use either different test and training sets, or Cross Validation.

It is possible that, after this analysis, we will focus more on a subset of the metrics, and maybe come up with additional metrics that will require a re-run of this iteration. This is ok, as this iteration is fully automatic.

Step 3. Parameter-related S/ML analysis

This step explores possible weaknesses and correlations between the challenges and their correct answers, but does so taking into account the values of the different parameters that are used when creating a challenge. Note that these values are not always accessible nor easy to deduct from a produced challenge. Thus, this step is not always possible.

Next, we will comment when this step is applicable, as well as its utility: what additional information we want to extract.

This step will typically only be possible if either the CAPTCHA designer is collaborating with the analysis, if the CAPTCHA is open-source,

or when the value of the main parameters affecting the generation of the CAPTCHA challenges are evident given a particular challenge. If these circumstances are not met, then it is in general impossible, or costly, to learn the value of the challenge creation parameters from a particular challenge - this analysis itself can be more costly than the attack we are looking for.

For example, let's imagine a CAPTCHA shows synthetic images of people from different professions that the user has to categorize by social status or perceived income. When the CAPTCHA wants to create an image to be used in a challenge, it has to decide (typically randomly) the value of some parameters: the profession (among a certain number and type of professions), a particular 3D model (among a number of models of different types), what colors to use, the field of view of the image, what additional elements to use (number of clothing, tools, etc.), lightning conditions, etc. The value of all these parameters affects the challenge created. Their particular value might affect also the difficulty of the challenges created, and that is precisely what we are trying to discover.

The type of questions that this part of our analysis wants to answer are such as if given different values of parameters of the CAPTCHA generation algorithm, some of them especially weak and thus should be avoided, or if there are factors or measures that contribute more to the strength of the CAPTCHA, or what parameters are more sensible towards the CAPTCHA security. In a way, what we want to know is whether the CAPTCHA design seems to be correctly using the base problem to its full strength, or at least, be certain that it is avoiding specially weak cases.

Typical questions asked during this phase could be: is it possible for an attacker to identify identical elements (backgrounds, sprites, etc.)? Is it possible to automatically deduct some of the values of the parameters affecting the generation of a given challenge? How do the different design elements affect the strength of the CAPTCHA?

The tools for this second iteration are the same used in the earlier analysis. Now, we will use them with restricted parameter values and study how they perform in these cases.

If we do not have access to the CAPTCHA source code or the collaboration of its designer, in some cases we still can separate the correctly and wrongly solved challenges in sets depending on the different parameter values with which they were generated. If we have access to the challenge

creation mechanism, we can generate challenges automatically using different parameter values.

During the exam of these questions, we forget about the user friendly aspect of the CAPTCHA. What we want to know is only how they affect its security. To measure how these different design decisions affect the CAPTCHA security, we will use the same analysis tools as we used in the previous step. If during that analysis we find that certain tools are more promising than others, we will focus our efforts in those, but we will use in any case all of them, as a different parameter set for the CAPTCHA can render it susceptible for a different type of attack.

This analysis would render at least answers to the following questions:

- If we found a correlation in the previous step or in this one:
 - Does this correlation affect to all challenges uniformly, or does it depend on some parameter values?
 - How does each parameter and parameter value affect this correlation?
 - Is it possible to invalidate this correlation, using some parameter values?
 - Is there one or different correlations, depending on the parameter values?
- If we haven't found any strong correlation:
 - Is there any sub-domain of parameters that shows a hint at a correlation, and should be further explored with more examples or values?
 - What parameter values seem to give the most uniform distributions in the metrics used?

We have introduced the main steps of BASECASS. Now, we will describe it in detail so that security practitioners can use it for the security analysis of new CAPTCHA proposals.

6.3 Detailed Description of BASECASS

BASECASS is designed to try to find unexpected weaknesses that can be exploited to build side-channel attacks. These weaknesses will possibly allow paths of attack that are typically not the expected one, that is, the theoretical path of attack that the CAPTCHA designer considers the only possible way to solve the CAPTCHA.

BASECASS tries to find weaknesses using readily available tools and public knowledge in what constitutes a low-cost attack. The cost of an attack is an important variable in IT Security. Most IT Systems are considered secure to a certain level of means and involvement from an attacker. Thus, the cost of an attack is crucial factor.

Notice that in general, we will not be trying to evaluate the strength of the problem on which the CAPTCHA is theoretically based. Instead, we will try to find ways to solve it that are simpler than solving that problem. In certain scenarios, this can involve proving that the actual problem presented by the CAPTCHA is too weak compared with the theoretical base problem.

In the next sections, we introduce in a more precise way our proposed framework for a basic security assessment of a new CAPTCHA/HIP design.

Figure 6.2 shows a simple depiction of the iterations of BASECASS, and also the relation between the definition of the metrics and later use of them in the posterior analysis. This figure serves as a guide and reference to understand the different *iterations* of BASECASS (black-box analysis, and if possible and necessary, parameter-based analysis). It also shows the steps of BASECASS: the challenge and answer space analysis, the black-box statistical and ML analysis and the parameter-based analysis. Note that as soon as we find weaknesses and test that they are strong enough to enable an attack, we can finish our analysis. This can happen in any of the steps of BASECASS.

Security analysis or penetration tests are not new in IT Security. Our contribution here is threefold:

- Summarize the objectives that can be followed in order to assess a minimum security level for a new CAPTCHA proposal. These objectives can be regarded as a generic, high-level method, that

can have different implementations depending on the particular CAPTCHA.

- Present some methods that can be used in the case of some CAPTCHA tests, both medium-level methods and low-level precise techniques that can be used to implement them.
- Present examples of their application and the expected results that can be obtained from them.

Next, we will revisit the CAPTCHA definition. This will provide us with terminology and examples useful for later describing the first step of BASECASS. Then, we will present in detail the three steps that compose BASECASS. After we formalize the different parts of our framework and present them in detail, we will show examples of its application to some CAPTCHA designs.

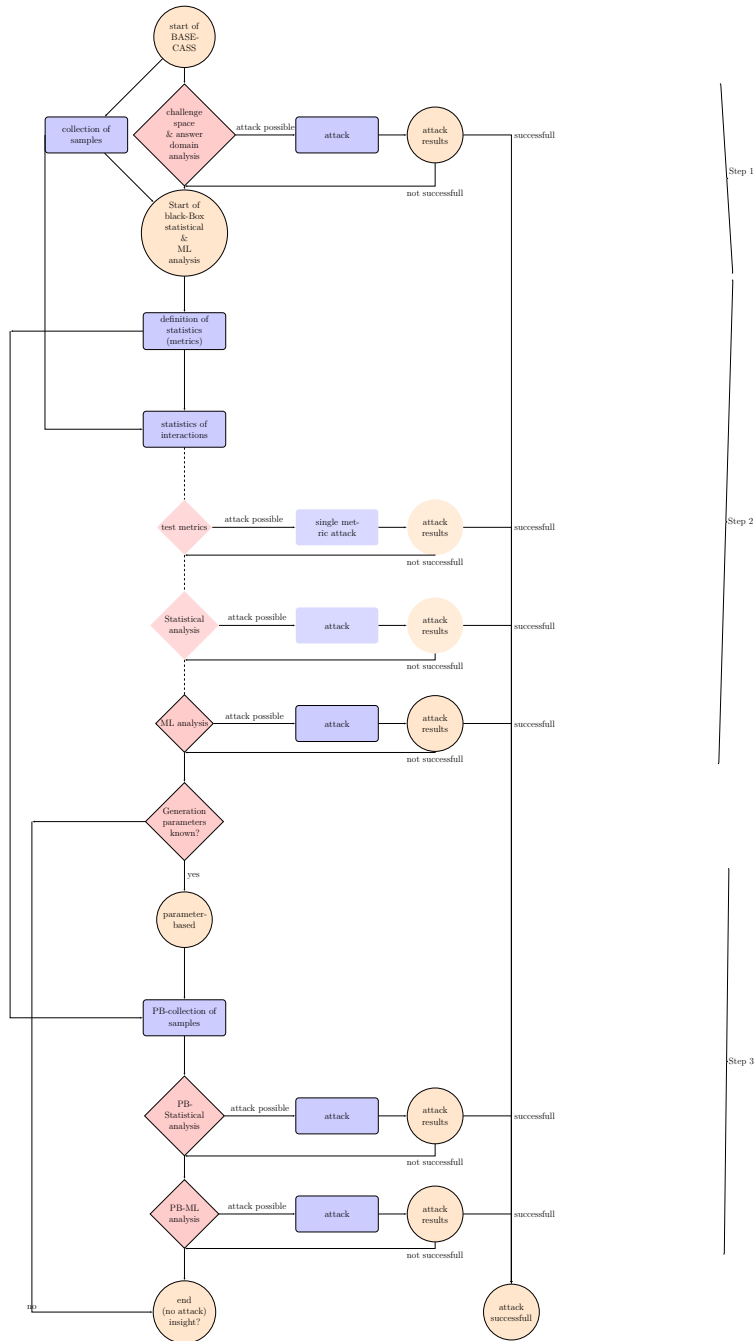


Figure 6.2: BASECASS generic flow chart.

6.4 Revisiting the CAPTCHA definition

In order to present BASECASS in relation to a typical CAPTCHA, we will re-visit the model of a CAPTCHA/HIP introduced in 2.1.3 and analyse it with further detail, focusing on those parts most relevant to our framework.

As we saw in section 2.1.3, a HIP/CAPTCHA H can be defined as a function f that returns a test and has up to two input parameters: a random seed, and optionally, a level of difficulty, $f(R, diff) \rightarrow t$. Typically, this difficulty level $diff = (f_1, f_2 \dots)$ can be divided into a number of factors that affect how the challenges are created.

As an example, in a *puzzle* or *image reconstruction* CAPTCHA, such factors can be: the number of puzzle pieces, their size, which other image they were taken from or how they are filled, etc.

As another example, the well-known OCR/text CAPTCHA Securimage (see figure 6.3) allows the following parameters that influence its appearance and difficulty:

- Font color F_c
- Background color B_c , and background image B_i (static, or random from a directory)
- Perturbation level ($P \in [0..1]$), sets how distorted the characters will be. 0 means there is no distortion, and characters would be rendered as the regular font
- Number of lines drawn in the image NL (typically on top of the characters)
- Font used to create the characters F_t
- Selecting the character set (for example, to avoid 1, 'l', 'I', and other characters difficult to distinguish among themselves, or not) $Ch = ('a' \dots 'z' \dots)$
- Image size (in pixels) $W \times H$
- Number of characters to show Nc



Figure 6.3: Example of a challenge produced with Securimage.

If we want to characterize at a higher level the CAPTCHA challenge characteristics, we need to specify the attributes that affect the challenge generation, so for example, for Securimage we can write:

$$f(R, diff) = f(R, (F_c, B_c, P, NL, F_t, Ch, W, H, Nc)) - > t$$

This terminology will be useful in the second phase of the step one of BASECASS, that we will describe in the following section.

6.5 Step 1.- Black-Box basic security analysis

This is the first phase of BASECASS. This phase encompasses our first contact with the CAPTCHA, and our first analysis from a completely external point of view, that is, without any prior information about the CAPTCHA, its generation process, its protocol, its validation mechanism, In this phase we will try to answer very generic questions, yet relevant in order to allow us to understand better the possible strength of the CAPTCHA against automated attacks.

We can divide this step of BASECASS in several phases. The first one is the automatic interaction phase, in which we will be able to build a way to interact automatically or semi-automatically with the CAPTCHA and record its challenges and responses. The second is the analysis of the challenge space, in which we will try to answer to questions about the challenge space size and distribution. Similarly, the third is the analysis of the answer domain size and distribution, in which we will try to answer questions about the possible answers to the challenges, both correct and wrong.

6.5.1 Phase I: Automatic interaction

In order to further analyse a CAPTCHA from an outside attacker's perspective, it will be convenient to have some basic means of automatic interaction with it. In particular, we will typically need to detect when the CAPTCHA challenge is presented, determine the different elements of the challenge, download these elements so as to be able to further work with them, submit the possible solution and receive feedback from the server to know whether our solution was correct or not.

There might be different obstacles that make it harder to construct a program that interacts with the CAPTCHA being studied. Among them, the most typical one is obfuscation. As most CAPTCHAs can be presented as elements within web-pages, it would typically be possible to analyse their elements and interactions with the use of third-party protocol analysers. Some CAPTCHA designers opt for the use of obfuscation techniques in an attempt to prevent further analysis and automatic interaction. This effort is futile for a number of reasons:

- Obfuscation is an example of what in IT Security is known as *Security by Obscurity*, a paradigm with a long tradition of not withstanding the trial of time.
- The effort to obfuscate source code in particular has a long tradition in software, with the initial aim of copy protection of software. To date, it has never been able to fulfil its requirements, given that enough interest is there to break them. This is so even if gigantic companies (as Microsoft) have put all the possible means to improve them in order to increase their revenue preventing software copying. Software and hardware protection has not had also much success to date. A good example of this are game consoles, in which even controlling the hardware and software and adding internal cryptographic mechanisms has not prevented reverse engineering.
- Lastly but more importantly, it is possible to create software stacks that completely simulate a regular browser and its environment (plug-ins, operating system, etc.) thanks to some browsers being open source, and also to the integration of automation tools that work with the most common browsers and allow interacting automatically with them (Huggins and Hammant, 2014).

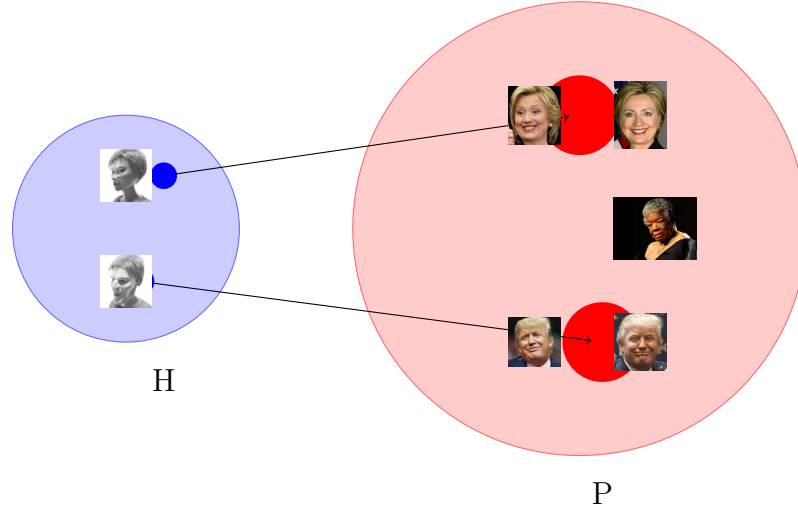


Figure 6.4: Example mapping between subsets of H and P .

- Even if for some reason, browser mechanization was not possible, there is always the alternative to create a completely virtual environment running into a virtual machine and automatically interact with it through the use of input and output drivers. Thus, browser automation, as well as mobile automation, is always a possibility.

6.5.2 Phase II: Analysis of the challenge space

The challenge space is the set of (possibly infinite) challenges that will be presented to the user by the CAPTCHA, that is, all possible $c \in (c, corr_c) \in H$. Some CAPTCHAs include different subtypes of challenges that can be presented to the users either randomly, or depending on user context: her past performance solving the challenges, IP domain she apparently connects from, etc. Here, we study the chances of appearance of each subtype, and decide to further analyse one or more of those subtypes.

Once we have focused on a type of challenge from a CAPTCHA, we will want to further characterize this set that is going to be analysed, our new H_a , that for simplification we will refer to as H . There are possible ways to further characterize it:

- If a challenge is composed of different elements $c = f(e_1, e_2, \dots)$, we will want to know the rate of appearance of them in the different challenges,

if they repeat from some sets E_1, E_2, \dots or are somehow related, etc. Also we will like to know if some of them appear to simplify the analysis by computer programs, thus possibly rendering weaker challenges. That is, does it include sub-domains that are easier, and that can be detected? This is an analysis that not always can be performed, as the different elements might be hard to discern using a program thus requiring manual tagging, which might be too expensive for a low-cost attack.

- Check the statistical distribution of the appearance of elements of the different types E_1, E_2, \dots to search for deviations from uniform. This can be done, for example, using a Pearsons' χ^2 test.
- Assess the domain size of H . If it is infinite, we can relate it to the base problem P , to have a broad estimation of how the difficulty of H and P relate. Figure 6.4 shows an example of mapping of H and P , where H requires the user to select the gender in synthetic images of faces, and P is the natural gender recognition problem in pictures of human faces. If H is finite, we would like to estimate its size. For that purpose, we can use the *mark and recapture* method for estimation of populations in their natural environment (Seber, 1974). In general we will want to know how big is H compared to P as a proxy measure for the difficulty of solving H compared to solving P . Thus, this is not the only important criteria, as even if the size of H is comparatively big regarding P , it might not be varied enough, that is, it might still be a subset of P with relatively low variance compared with P . If there is a way to measure or compare this aspect of H compared to P , this will also serve as a proxy measure of the difficulty of H compared to P .

To answer these questions, we can use our previously created automatic CAPTCHA interaction software to download a set of challenges and their solutions validated by the CAPTCHA server and analyse them off-line to decompose them into their corresponding $c = f(e_1, e_2, \dots)$. This analysis might be manual, but we will typically not need many examples to get the required results.

This analysis might seem unnecessary, as one might think that most CAPTCHA designers will create an infinite set of challenges H that would also be varied enough to cover a broad subset of P . In reality, it is often very difficult to do so, and H typically represents a very small subset of P . In this step we want to characterize it, and thus have a basic idea of the variability of the challenges.

We will present now two examples of H and P and their relation in real-world CAPTCHAs.

Example 1. FunCAPTCHA One might consider the relation between the problem of gender recognition in general, that is, in any possible situation in which a human being is able to tell the gender with a certain accuracy, and the problem presented by FunCAPTCHA.

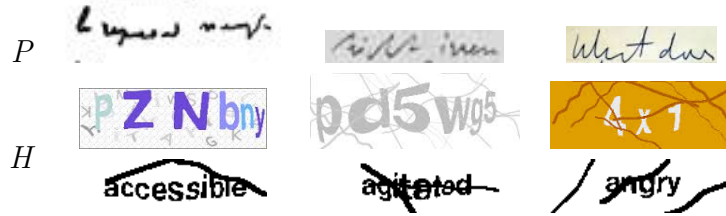
Table 6.1: Comparison of H and P for FunCAPTCHA.



Table 6.1 presents a very brief comparison of H and P for FunCAPTCHA. It can be seen that a number of properties that can vary in P are kept static in H . In particular, the color of the image, the model used, the facial expression, or the elements that can be worn (for example: glasses, scarfs, etc.), the facial paintings, the ethnicity, the age, etc. are elements that have two single values in H , while most of these dimensions have a number in the hundreds, thousands or millions (number of models) in P . Thus, for FunCAPTCHA, H is an extremely small subset compared to P .

Example 2. CRC - OCR. The CRC uses Securimage. This means that the words are *distorted* in the pretension of rendering them unreadable to a machine.

Table 6.2 represents three possible depictions of elements of P in the first row. The second row shows possible elements of $H_{Securimage}$ for Securimage with various values for the challenge-creation parameters (font and background color, perturbation level, number of lines drawn, font used,

Table 6.2: Comparison of H and P for the CRC-OCR.

number of shown characters, etc.), and elements of $H_{CRC-OCR}$ for the CRC-OCR. As can be seen in this table, $|P| > |H_{Securimage}| > |H_{CRC-OCR}|$. The unexpected part here is the $|H_{Securimage}| > |H_{CRC-OCR}|$ inequality, and in particular, the big difference between them. The range of parameters in the OCR-CRC is extremely small compared to what Securimage admits (and also other OCR/text CAPTCHAs). Worse, the CRC-OCR only codifies 133 different texts. Thus, the comparison between P and $H_{CRC-OCR}$ is extremely dis-favourable in terms of size and variance for $H_{CRC-OCR}$.

6.5.3 Phase III : Analysis of the answer space

The answer space and its statistical distribution can be even more fundamental than the challenge space. A small domain answer and/or a very non-uniform distribution can render it vulnerable to a brute-force attack, or can amplify the chances of other types of attacks.

Note that a proper analysis of the answer space is not always possible in terms of cost. The answer space could be too big to explore, even stochastically. Even if we explore it manually, for example solving a number of challenges with the aid of third-party services as Amazon Turk, the collected answers might be a set that is not statistically significant. In fact, if a proper analysis of the answer space can be done at a low cost, this is in fact a sign of possible weakness of the CAPTCHA.

In some cases though we can do a broad estimation of the answer space by analysing the types of challenges presented. We can use the previously downloaded challenges and their answers, either obtained manually or by other means, and study the total answer distribution. If the CAPTCHA is composed of clearly distinguishable elements, we can study the answer distribution for each particular element value or range of values. We will then compare this

answer set to the theoretically possible answer space, and its distribution with the uniform. It is possible to statistically test their distribution by a simple Pearson's χ^2 test, comparing it to an uniform distribution.

If we find that the answers aggregate in a few points or a small range, and/or there is a relation among their distribution and the challenge elements, we can analyse whether this weakness can be used in a brute-force attack, and if it might be, we can test it by using our automatic means of communicating with the CAPTCHA server developed in the step before, so we can check for the real success of such an attack.

Next, we will present two examples of problematic answer distributions in real-world CAPTCHAs.

Example 1. QRBGS CAPTCHA The Quantum Random Bit Generator Service (QRBGS) CAPTCHA requires the user to input the result to a mathematical formula that has been rendered in a low-resolution setting, as to make it harder for OCR tools to analyze it. It includes four sub-challenge types, that are the different types of formulas to solve: derivatives, polynomials with exponentials, polynomials expressed as multiplications of factors (for both polynomials, the least real zero is asked), and arithmetical challenges. Table 6.3 shows examples of these different challenge subtypes.

Table 6.3: QRBGS challenge subtypes. The mathematical expressions are shown in low resolution, as they are rendered by the QRBGS CAPTCHA.

Problem type	Expression	Solution
Solve:	$3 * 3 * 0 - 2 * (-2) + (-2) - 3 = ?$	-1
Find the least real zero of the polynomial:	$p(x) = x^3 - 2x^2 - 4x + 8$	-2
Find the least real zero of the polynomial:	$p(x) = (x + 1)(x - 4)(x - 3)$	-1
Calculate:	$\frac{\partial}{\partial x} \left[2 \cdot \sin \left(4 \cdot x - \frac{\pi}{2} \right) + 2 \cdot \cos(2 \cdot x) \right] \Big _{x=\pi}$	0

The security of this CAPTCHA has already been analyzed (Hernandez-Castro and Ribagorda, 2010). One of the main findings was that the distribution of correct answers to each type of sub-challenge is spread over just a few integer values, and is strongly not uniform in all cases, as can be seen in figure 6.5.

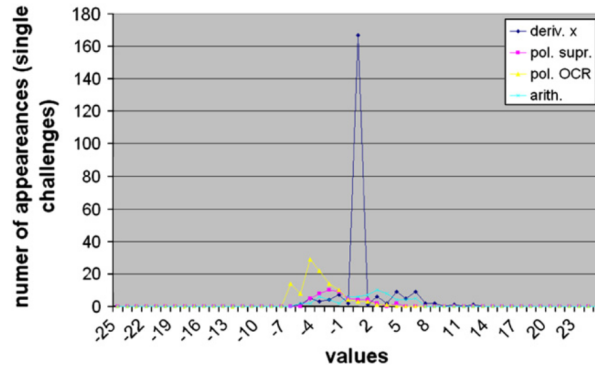


Figure 6.5: Distribution of correct answers of the QRBGS CAPTCHA by challenge subtype (Hernandez-Castro and Ribagorda, 2010).

This distribution is so skewed that in this particular case it leads to a straightforward attack with a 44% success ratio.

Example 2. CRC The Civil Rights CAPTCHA (CRC) analysed in chapter 4 includes two sub-challenges that need to be solved in order to answer a challenge. The first one consists in reading words (or two-word expressions) representing an emotion. These words are protected by rendering them using Securimage. The second one consists in categorizing a news bit as either positive or negative. The distribution of appearances of each possible answer is very non uniform, as shown in Figure 4.5, with a χ^2_{132} of 482,12, and thus a p-value of 0. We can conceive a brute-force attack in which we pick up randomly one answer from the top 5 more probable ones (just to avoid repetition, in case of filtering). This would pass the CAPTCHA approx. 1,2% of the times.

The second sub-challenge presents already a very small answer space: positive vs. negative. Worse, the distribution of them is also non uniform because even though the appearances of each news bit is uniform, the challenge distribution that it is inheriting from, which is the total set of news-bits, is strictly non uniform: only 33% of the news are positive.

6.5.4 Summary

We have presented the initial phase of BASECASS, that gives us an initial contact with the CAPTCHA, as well as an estimation of its difficulty, especially

compared to the general problem it might be based upon.

6.6 Step 2.- Black-box S/ML analysis

Once we get to this step, we know that the CAPTCHA challenges and answers have a broad enough domain and range and decent statistical distributions, that is, the CAPTCHA design is not fundamentally limited.

What we want to know now is whether it is possible to extract some *basic* information from the challenges that would allow us to have a good guess of the correct answer, thus enabling a side-channel attack. By *basic*, we mean information that is easily accessible, either directly or through readily available tools. This information is going to be extracted by applying some metrics to the challenges. We do not expect all of these metrics to be relevant, but we want to see if they can extract enough information so that, combined, they allow to improve our chances of solving the challenges. Given the volume of data, the guessing about which one is more relevant will be left to statistical and ML tools.

We do basically two things: first, select which metrics can be applied and might be relevant for the challenge types, possibly creating new ones based on software that is readily-available or partially tailored to our case. Then we will use them to create a file describing each of the downloaded challenges and answers, and their gradation by the CAPTCHA server (correct or wrong) possibly along with additional wrong answers. Secondly, we optionally use statistical tools and ML algorithms to analyse this file in order to search for correlations and less straightforward relationships among the challenges and their correct answers. The number of gradated challenges that we will use will vary depending on the CAPTCHA domain size, and the ML methods used. In general, we can examine the success of the ML at different number of challenges to estimate whether including more would improve or not the results.

Sometimes, before applying these metrics to the possibly solved challenges, we would benefit from some very basic pre-processing of the challenges. This is especially so in the case of OCR/text CAPTCHAs, in which a number of *noise* is added to the images in order to try to make them harder to process.

6.6.1 Phase I: De-noising

There are many cases in which a CAPTCHA designer tries to protect the challenges produced by adding to them different types of *noise*. This is typically more so in OCR CAPTCHAs, in which otherwise the solution to the challenges would be straightforward.

Some of these transformations have been designed to conceal information in a way that spreads uniformity on some possible metrics on the challenges. Thus, sometimes it would be useful to undo, even if partially, some of these transformations, in order to be able to get clearer data.

We will call *de-noising* to the pre-processing of challenges in a way that we partially undo alterations and other clutter that messes up with our metrics. This is an optional step, and is not necessary in many cases. It depends a lot on the type of CAPTCHA, and whether it has measures or not to try to conceal information and whether it is easy or not to undo some of these alterations. Note that this step has to have a low cost: there is not point in creating a costly de-noising step that is even more costly than the attack itself. De-noising only makes sense if it is straightforward to perform or very easily programmed.

Some of the possible transformations or pre-analysis targets can be:

- Distinguish among background and foreground elements.
- Undo global transformations as rotations, projections, addition of lines or curves, etc.
- Undo local transformations as warps, occlusions, etc.
- Other cleaning particular to the CAPTCHA proposal, that undo an alteration specific of that CAPTCHA design.

We will present now an example of how de-noising can be done in a real-world CAPTCHA.

Example. Captcha2 Captcha2 was a commercial CAPTCHA proposal (FusionQuest, 2009) in which the user had to click within a certain canvas on the position where a particular character was rendered. Captcha2 used several

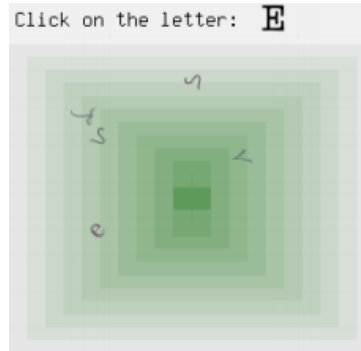


Figure 6.6: Example of a Captcha2 challenge.

measures to make this harder for a bot, in particular, rotation of characters, changes in sizes, local warps, and adding a background that was designed to not be easily removed, as their designer understood that the background would add noise in case of edge detection. Figure 6.6 shows an example of a challenge from Captcha2, in which it can be seen one of the many background styles used.

Captcha2 was analysed and broken (Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda, 2010). In this analysis, it was found that a simple algorithm could detect the colours used to render the border, no matter the different geometrical ways of rendering it. The algorithm used removed the background by colour and space similarity, checking the colours adjacent to each pixel and considering background colours those within a certain distance.

In this way, Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda were able to remove the background, and latter apply simple metrics to select the correct answer, breaking Captcha2 100% of the time. This is possible because even when their attack has a 87% accuracy per challenge, Captcha2 considers a bot only after 10 successive failures, which their attack never does.

6.6.2 Phase II: Pre-processing & transformations

In some cases it is useful to pre-process the challenges as a way to extract additional information in the form of metrics. This pre-processing can be minor, in order to increase the usefulness of a metric. For example, we can

apply a filter for edge detection to an image and later a metric to count how many pixels are detected as edges. Another type of filter are pre-programmed convolutions. Sometimes, this pre-processing can be more fundamental. One example would be transforms that translate the challenge into another, more relevant domain: a text into a vector of words or into its representation into the first layer of a Restricted Boltzmann Machine (RBM), an image into a spatial frequency domain, or an audio from the time domain into a frequency domain.

Next, we will present an example of pre-processing (example 1) used in a real-world CAPTCHA, as well as a very successful example of transformation (example 2) used against several OCR CAPTCHAs.

Example 1. KeyCAPTCHA Capy CAPTCHA and KeyCAPTCHA were studied in chapter 3. When we were analysing both, one of the metrics we created ad-hoc for the case was the file size after lossy compression. This metric seemed meaningful for Capy CAPTCHA, because the original image typically has more color and texture redundancies than an altered one. Yet KeyCAPTCHA presented a problem, as their challenge images present objects in white background. White is very *easy* to compress, so our metric will favour those cases in which most white background is respected - that is, the puzzle piece is **not** put on top of a puzzle void (white), but instead on top of some part of the object.

In order to solve this, we decided to add random noise to the background white pixels of KeyCAPTCHA. Random noise is hard to compress, so the metric would now favour the opposite: putting the puzzle pieces on top of the noise portions (the background).

Example 2. OCR/text CAPTCHAs An example of a more involved transformation was proposed by Gao et al. (2016), in which OCR/text CAPTCHA challenges are first transformed into their 2D Log-Gabor components. Figure 6.7 shows an example of such transformation for four different angles (from 0 to $\frac{3 \times \pi}{4}$), along with the final reconstruction. The first row shows the original CAPTCHA challenges. The following rows show the components detected for each angle, starting with *angle* = 0 (vertical components). The final row shows the reconstruction of the image using the components previously detected. Note that the attack of Gao et al. (2016) uses a representation of those components as the input for their ML classifier.

	Microsoft	QQ	Baidu
Angle			
0			
$\pi/4$			
$\pi/2$			
$3\pi/4$			
+			

Figure 6.7: Example transformation of different challenge images into their respective Log-Gabor components (Gao et al., 2016).

In their attack, they later group these components using different graph heuristics as well as k -Nearest Neighbours. With this approach, Gao et al. (2016) are able to break most well-known text CAPTCHAs with success rates varying from 5% to 77.2%.

6.6.3 Phase III: Metrics

In this section we explain a key part of BASECASS: the procedure to come up with a pool of metrics that can be applied to extract side-channel information. This information, extracted from the CAPTCHA challenges, will be useful later when trying to find correlations among these values and the solutions of the challenges, possibly with an accuracy good enough to break the CAPTCHA.

In the next sub-sections, we will present the different ways in which to select these metrics from previously used metrics. We will also comment on aggregating the values of these metrics if they are too many, and ordering them per challenge, in certain cases in which this can be advantageous. We will also comment about the use of original or adapted tailored metrics, that can be beneficial in some cases. Finally, we will comment on the usefulness of doing small tests on the behaviour of these metrics, either to discard them, adapt them, or maybe to use them directly, without the need to feed them into ML classifiers.

6.6.3.1 Selection of metrics

When we reach this phase of our framework we know what type of media holds the challenge we want to analyse: an image, a game (in JavaScript, Flash, etc.), sound, video, text, etc., so we are ready to select those metrics that are applicable to the particular type we are dealing with.

At this stage, we have already seen and analysed the domain of the CAPTCHA challenges. That will allow us to decide which metrics could possibly be more related to them, in order to extract more relevant information. Some possibilities for the different media types are:

- Video: there are basically two approaches for CAPTCHA videos. One is to analyse them as a sequence of images. The other is to perform an integral analysis. The bit-stream containing the video can be converted to a loss-less compression format, in order to use these analysis tools.
- OCR: in the case of text-based CAPTCHAs, the amount of information to analyse would be significant less. Here, we can model the texts using simple NLP processing techniques, like converting them to bags-of-words, or to their representation in semantic networks as WordNet.
- Audio: in the case of audio, we can either process the stream format itself, or convert it back to a non-compressed format. We can also apply some very basic and well-known audio processing techniques, like the Fourier Transform, to convert it from the time domain to the frequency domain.
- If the CAPTCHA challenges are presented to us in an unknown binary format, we still can extract some metrics from them that might be meaningful in some situations. For example, we can process the byte-streams with different compression algorithms, measuring size and dictionary size, or with randomness tests.

Generic metrics Here we will offer a list of possible metrics depending on the CAPTCHA media type. These can serve as a base-case if the practitioner using this framework is not familiar with CAPTCHA security analysis or is not able to come up with possibly better, more meaningful metrics. Much in the same way as some ML algorithms require the practitioner to pick up

relevant features, our framework proposes metrics but also encourages the practitioner to try to come with additional, more tailored ones.

The practitioner should keep in mind that even though these metrics can be in some cases relevant enough as to base an attack on them, it should not be expected that they alone will be that useful. Instead, what we will typically be looking for is that they add information to other more relevant metrics, so that all-together they are able to leak enough information about the CAPTCHA challenges.

Contrary to ML, in CAPTCHA security analysis we do not require a high level of accuracy. Unfortunately for our security scenario, a small accuracy, enough to bypass the CAPTCHA in a low number of occasions, as low as 0,01% (Chellapilla et al., 2005b) or 0,6% (Zhu et al., 2010a) will be enough.

In general, there are some possible metrics that can be used depending on which is the content media type of the challenges and/or challenge solutions. Note that, even if the challenge solution can be represented as a short response (some mouse coordinates, for example), it sometimes can be represented in a more meaningful way as a transformation of the challenge itself. This transformation result is what we will use in order to apply our metrics. Depending on the media type, some possibilities are:

Metrics for text

Even though all computer data is binary, whenever it consists of a text codification, we will consider it separately, as both the amount of information and their type are strongly different from other types of data codified. This is so because text uses a set of characters and the options are restricted by the language, forcing a very non uniform statistical distribution. In the following paragraph we will present some useful representations to extract frequency metrics, some of which can be applied to text analysis algorithms, such as Latent Dirichlet Allocation (LDA) (Blei et al., 2003):

- Index of Coincidence (IC): this metric comes from the Cryptographic world. It measures the inter-correlation of characters inside a text or a collection of documents. It is thus related to the redundancy (and thus, information quantity) of a text. Equation 6.1 shows the IC for a text using c different characters, where each one appears n_i times (and

$N = \sum_{i=1}^c n_i$). It is very useful to capture meta-information from a text, as for example, in which language it is written, as every language has a different *IC*. Table 6.4 shows the IC for large corpuses of text in each language. The higher IC means that the language is more redundant, thus less efficient.

$$IC = \frac{\sum_{i=1}^c n_i(n_i - 1)}{N(N - 1)/c} \quad (6.1)$$

Table 6.4: Index of Coincidence for some languages ¹.

Language	Index of Coincidence
English	1,73
Russian	1,76
Italian	1,94
Portuguese	1,94
Spanish	1,94
French	2,02
German	2,05

- N-grams: they are the statistics of appearance of each series of N consecutive characters. This is a typical way to characterize a text, having been used for text search and retrieval, detection of plagiarism, classification by text subject, etc.
- Bag-of-words: similar to n-grams, but using words, we used them to count the number of appearances of each word in each sentence or paragraph or word set that we want to model. In order to limit the size of the vectors, we have to limit the size of the vocabulary that we will focus on. We can do so by removing very common words with little meaning (articles, prepositions, etc.) and focusing then on the N most frequent words. This is a better representation than n-grams, yet it does not capture information as relations among words and distance in text.
- TF-IDF: Term Frequency - Inverse Document Frequency is a metric that portrays how relevant is a word in a particular document, given a collection of documents. It counts the number of appearances of the term w in document d , $f(w, d)$. It also counts the number of documents

¹Data from the Wikipedia, at https://en.wikipedia.org/wiki/Index_of_coincidence.

in which w appears, for every document from the collection of documents D , as in Equation 6.2. Then, $\text{tfidf}(w, d, D) = f(w, d) \times \text{idf}(w, D)$. This is the metric in its simplest form, but there are variations in order to account for different scenarios.

$$\text{idf}(w, D) = \log \frac{|D|}{|d \in D : w \in d| + 1^2} \quad (6.2)$$

The TF-IDF value for a term w is proportional to the number of times the term appears in the document, and inversely proportional to how many times it appears in the document collection, which is useful as some terms are more common than others. This metric, with variations, has been used in search engines, to filter stop-words (words too common), classification of text, and also summarizing text.

Metrics for binary challenges (static)

If the challenge type is not based on text, we will consider it binary, as it will normally be contained in some binary stream. If this binary stream is itself represented in the client as an image, video, sound, etc., but not run (as a code), we will consider it static. In this case, we can use some general metrics that might be useful in order to measure the information content they might have. Note that some of these metrics will be more relevant if the bit-stream has been unenclosed and uncompressed, as compression algorithms might affect them.

- Entropy: this metric tells us the amount of information contained in the bit-stream. The Entropy is always related to some model. Entropy can typically be measured for the byte-stream, that is, looking at the bit-stream in groups of 8 bits and comparing it to an uniform distribution, as in Equation 6.3.

$$H(B) = \sum_{i=1}^{2^8} p_i \times \log \frac{1}{p_i} \quad (6.3)$$

Where p_i is the number of times that the 8-bit value i appears in the byte-stream, divided by the total length of the byte-stream.

One possible and practical way to measure it is using the corresponding result of the ENT test of randomness.

- Loss-less compression algorithms: loss-less compression algorithms are generic byte-stream compression algorithms that do not loose any information present in the byte-stream while reducing its size. There are many such algorithms, with varying characteristics as their speed, ability to work on-line (one pass vs. several passes), etc. Some of the most well-known algorithms are the Lempel-Ziv-Welch (LZW (Welch, 1984), used in utilities such as *compress* (Unix) or the GIF graphic format) and the Zip (that allows different algorithms, being *deflate* by Phil Katz (Katz, 1996) the most commonly used) algorithms.

Metrics for binary challenges (dynamic)

The challenge type can be contained in a byte-stream which is intended to be run at the client. Typical examples of this are games run in the client's browser, normally using technologies such as Flash (now deprecated) or more typically JavaScript. In these cases, the elements of the interaction are sometimes passed as individual resources (e.g. images for sprites) that can be examined using the techniques mentioned previously. In other cases, both the code and the content are obfuscated in order to try to prevent any analysis. In these cases, we can use browser automation libraries to locate these elements and isolate them in a way in which we can analyse them using the previous methods. In the strange case that this isolation is not straightforward, we can always use these automation libraries to render what is shown to the client as images in order to extract some side-channel information, again using the previously mentioned techniques.

Depending on the subtype of the different binary items, we might be able to use different well-known metrics to further analyse the challenge and our possible answers. In particular, if the type of the sub-items are images or audio, we will use the different metrics applicable to them, as discussed below.

Metrics for images

The applicable metrics will vary depending on the image content. Because of that, we will need to differentiate between the two most common types: when the image contains a text (OCR/text CAPTCHA) or when it contains a general image. In this second case, we will also differentiate if the image is

natural or synthetic. Depending on the case, some of these metrics will be more or less useful:

- Histogram of colours: this represents the frequency with which each colour appears in the image. We can aggregate the different colours by distance in order to make this histogram more significant. We can do this either prior to the analysis, defining *bins* of colours, or at the time of the analysis, using some clustering algorithms as K-Means, SOM maps or T-SNE.
- Pixel count, possibly after applying a threshold, per rows or columns. This can be useful for segmentation or character count.
- Loss-less compression file size: we can use already developed loss-less compression algorithms to measure the information and detail content of the byte-stream, and its predictability. One of such compression algorithms, for images, is the one used by the PNG format.
- Lossy compression file size: binary data-streams that are used to represent different media types (images, audio, video) can be further compressed using specific compression algorithms. Many of these algorithms compress them allowing for a certain decrease in quality. This quality level can be sometimes specified by the user. This in turn translates into some loss of data that is typically hard to perceive for a human. These compression algorithms can be extremely useful, as they measure the amount of information in a way which tends to be closer to what the regular human would perceive. We have pioneered the use of these algorithms to analyse the security of CAPTCHAs (Hernández-Castro et al., 2015).
- Continuity of areas by flood-fill: this allows us to separate different areas, which we later can measure by dimensions, pixel count, relation to other areas, etc.

Next, we propose two examples of selecting and using metrics in production, real-world CAPTCHAs, containing both images, but representing different objects: faces and characters.

Example 1. FunCAPTCHA An example of these metrics in use is the analysis of FunCAPTCHA images to classify them by gender (Chapter 5).

FunCAPTCHA can be considered a basic game CAPTCHA, as the UI is based on images, and the solution is specified by drag & drop movements from the user. In the particular case of FunCAPTCHA there is no need of any processing in order to detect the elements of each challenge. There is no background, and each challenge is composed of 8 images, one of which represents a female figure, where the others represent a male figure. The drag & drop target area is always the same, the center of the 9×9 grid (shown in Table 5.1, in chapter 5).

The metrics we decided to try for FunCAPTCHA were both basic and generic. In particular, we used:

- Total number of white pixels.
- As the images are in gray-scale, we also used histogram of gray-scales divided in 5-value bins, 15-value and 25-value bins, where each bin is a particular metric.
- Sizes of the image compressed using JPEG with different quality settings, from low quality to high quality.

As we saw in Chapter 5, even these simple metrics, combined with some ML algorithms, were able to bypass the gender recognition challenge of FunCAPTCHA with 83% success, using a quite small training set of 4320 images.

Example 2. Captcha2 Captcha2 is a commercial CAPTCHA proposal (FusionQuest, 2009) that requires the user to click within a certain canvas over a particular character, that is shown over a background, rotated, rendered in a random font, and mixed with other characters. While analysing Captcha2, Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda (2010) discover that after removing the challenge background, they can perform a simple flood-fill to discover continuity.

As can be seen in Figure 6.8, after removing the background (step *b*) and applying a threshold to convert the image to black & white (step *c*), they find the contiguous regions and perform a pixel count of each contiguous group of pixels (step *d*). Then they choose the region with more pixels (step *e*). This simple pixel count provides enough information to attack the CAPTCHA and successfully select the correct answer 87% of the times.



Figure 6.8: Steps to automatically solve a challenge from Captcha2 (Hernandez-Castro, Hernandez-Castro, Stainton-Ellis and Ribagorda, 2010).

Metrics for audio CAPTCHAs

If the challenge type is based on audio, we can either study each challenge in the time or in the frequency domain. If the audio contains speech, we can also use readily available software as the Voicebox package, used by Tam et al. (2008) to extract MFCCs and spectral and cepstral coefficients from PLP and RASTA-PLP. If there is noise or some masking using frequencies, we can also use frequency filtering. Going into more detail, some of the tools and data we can use are:

- Fourier Transform: this transform and its variants (discrete or DFT, and fast or FFT) are able to translate a signal that varies in time (like an audio signal) into a sum of frequencies. We can then use these frequencies and their amplitudes to characterize the signal at a particular time. This has been used to break a simple audio CAPTCHA used in Google Mail (Santamarta, 2008).
- Frequency filtering: sometimes the CAPTCHA designers can add noise to the challenges trying to make them harder to process. This noise can be stochastic or not (like echoes). In any case, we can apply frequency filters, or time filters, to remove part of that noise and focus on the information of interest.
- Lossy compression size: we can apply a lossy compression algorithm (MPEG-1/2 Audio Layer III (MP3), Ogg Vorbis) and checking afterwards the result in terms of size and lost quality as a way to measure the amount of information.
- Metadata: play-back length and other information alike can also be of use, per-se or combined with the previous metrics.

Metrics for video CAPTCHAs

Video can be understood as a succession of images with sound (optionally). Intuitively, in comparison with an image, a video provides much more information, and thus is in principle more susceptible to a side-channel attack. Thus, the challenge requested from a video has to require a much more abstract ("high") processing than that required for an image. Given the high bandwidth and storage that video requires, video is typically delivered in a compressed format. The video can be compressed using a fixed bit-rate, or an adaptive bit-rate, the second being useful to analyse its information content. We can also extract the *key frames* and analyse them as isolated images using the techniques mentioned previously, and we can do similarly for the audio. Other well-known video indexing techniques can also be used to extract video content information (Hu et al., 2011, Asghar et al., 2014). Thus, some of the metrics we can use with video are:

- Metadata: bit-rate, number and separation of *key frames*, length of the video. Some of these measures are specially relevant if the video (or/and audio) has been compressed using variable bit-rate and that the compressor places *key frames* on demand. Note that this can also be achieved by re-compressing the video with desired parameters.
- Object analysis through image metrics: we can use readily available algorithms to detect objects in video (Viola and Jones, 2001, Kim and Hwang, 2002). Then, we can process the images of these objects with the metrics we already commented for images.
- Movement analysis using whole-picture metrics in which the movement zones are changed into a particular color, or analysis of such zones using image metrics
- Image analysis techniques applied to *key frames*: in video, key frames are those that contain a pure image, and from which posterior and prior images in the video sequence are derived (using delta frames that encode the variation). Depending on the encoder, key frames can be inserted into the byte-stream either by time or by scene change. In any case, they typically provide a higher quality picture of the original scene in a particular moment. They are good candidates for applying to them regular image analysis techniques as the ones seen before.
- Audio analysis techniques applied to the soundtrack: this consists on

extracting the soundtrack and applying to it the audio-related metrics we have seen before.

Metrics for game CAPTCHAs

As with videos, games present more information to the attacker than just a simple image. Current game-like CAPTCHA proposals do not have a high level of security and have been broken using quite simple methods (Mohamed et al., 2013). These methods also allow us to decompose a challenge into its constituent parts: background, draggable items and targets. These parts, images themselves, can be analysed using the metrics previously presented for images. In brief, once we have divided the challenge into its parts, we can perform an analysis of background and different items through image metrics, in order to characterize the correct solutions.

Summary

Table 6.5 lists some of the metrics discussed here, relating them to the different media types in which the CAPTCHA challenges might be. This table can be used as a *cheat-sheet* when trying to find metrics to use for a particular CAPTCHA type. These are just general ideas and guidelines, it is left to each practitioner to increase her own table of possible metrics, and find additional possibilities for each type.

6.6.3.2 Aggregation of the values of metrics

Sometimes, the values of a particular statistic can be many, providing information that is too detailed. That is the case for example in a color histogram. Even in a large image, it can happen that most pixels have unique color values. We would need huge amounts of data in order to start seeing a regular pattern, if there is any, that affects the colour distribution. Yet we can reduce this need using aggregation of values, that is, classifying them in bins and counting how many *appearances* of them we have.

Thus, in some cases in which counting exact values would render too detailed information, we can instead count not just a value, but in which bin of histogram it falls. Two problems arise with this approach:

1. How many bins should we use?, and
2. How do we aggregate the values?

These questions are similar to figuring out the most representative subset of features to represent a problem, from a given set.

If we use too few bins, we will aggregate too much data and lose the detail of the information, which will be counter-productive towards classification. If we use too many bins, we will not get any benefit from this aggregation, and the level of detail can lead to over-fitting of the ML algorithm.

In general, we can have an idea of how many bins we could need by looking at different histograms of different particular challenges and testing how many bins would be necessary to allow to differentiate among those histograms. This will also depend on the size of the training set used, and also on the particular ML algorithm. Thus, in order to avoid over-fitting, we will always use Cross-Validation. Once we have an estimation of a range for the number of bins N_b , we can use different bin-size values around N_b ($\frac{N_b}{2}$, N_b , $2 \times N_b$) and let the ML decide.

The second question can be more intriguing: how can we aggregate the different values? For example, if we are counting colours in (R, G, B) format, how will we decide that (R_1, G_1, B_1) and (R_2, G_2, B_2) pertain to the same bin? We can do so by using the Euclidian distance, or dividing the \mathbb{R}^3 in grids. Yet sometimes most of the information is contained within a few regions, which can lead to an excessive number of bins in order to get the necessary detail. Another option then is to pre-process this information and create aggregations through ML algorithms like k -Means or PCA. Then, the main results of that aggregation (centroids and sizes of each cluster, or components) would be the metrics for the next phase.

Next, we present an example of the aggregation of values in a metric used with a production CAPTCHA.

Example: FunCAPTCHA We have already presented an example of the use of histograms in the analysis of FunCAPTCHA images to classify them by gender (Chapter 5). FunCAPTCHA presents images in gray-scale, so we take the histogram of the different levels of grey.






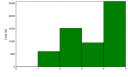

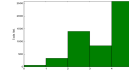
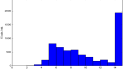
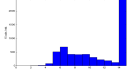
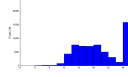
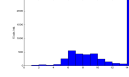
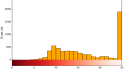
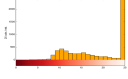
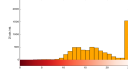
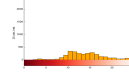








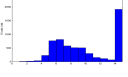
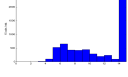
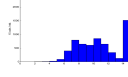
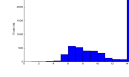
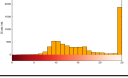
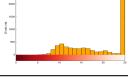
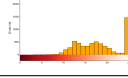
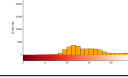
We decided to divide the histograms in 5 bins, 15 and 25 bins of values. The reason for these divisions is that 15 seems to be a number that divides the histogram in the most representative values, allowing for learning while still being representative and (hopefully) not encouraging over-fitting.

Table 6.6 shows some of the values of the histograms for different images. Note that this table shows the histogram metrics for both the training set (first 4 rows) and the test set (last 4 rows). We can see that there are indeed similarities in their histogram values and how they aggregate in bins. These similarities allow, in some cases, to find the category using k -NN. In the particular case shown in Table 6.6, this approach does not work for the first two test images, in the first two columns, as the images with a closer histogram of gray-scale usage are not from the same genre. It does work for the remaining two test images, in the last two columns, correctly classified as females by finding a closest neighbour that is also an already known image of a female face.

Table 6.5: Base-case metrics depending on challenge media and type.

Media	Type	Metrics
Text	Any	Index of Coincidence (IC)
		N-grams
		BoW
		IF-IDF
		Hidden representaion of a RBM
Binary	Any	Entropy
		Loss-less file size
		Lossy file size
Image	Classification	Entropy
		Loss-less file size
		Lossy file size
		Histogram of colours
		Histogram of colours: goodness of an adaptation to an index of k values
		Pixel count per: colour index, geometrical region or contiguous region
	OCR	Continuity of areas (flood-fill)
		Pixel-count of groups
		Pixel-count per columns
Audio	Any	FT and derived metrics
		Entropy
		Loss-less file size
		Lossy compression size
		Metadata
	Speech	MFCCs, coefficients from PLP and RASTA-PLP
Video	Any	<i>All image metrics, applied to key frames</i>
		<i>All image metrics applied to detected objects</i>
		<i>All image metrics applied to detected movements</i>
		<i>All audio metrics applied to the soundtrack</i>
		Metadata: bit-rate, number of <i>key frames</i> , etc.
Game	Any	<i>All image metrics, applied to the different elements: backgrounds, draggable objects and targets</i>

Table 6.6: Some FunCAPTCHA faces and their histogram values. The first four rows show faces from the training set and their histogram of usage of greys in bins of 5, 15 and 25 values. The last four rows show the same for some test samples, each one having as closest neighbour the previous training samples.

class (training)				
color histogram (5 bars)				
color histogram (15 bars)				
color histogram (25 bars)				
problem (test)				
color histogram (5 bars)				
color histogram (15 bars)				
color histogram (25 bars)				

6.6.3.3 Metrics of order

Sometimes, the CAPTCHA challenge offers a very high number of possible answers (hundreds, thousands, or more), and the solver is requested to pick up only one which is the correct one. This has the problem that, just by probability, if we are using a classifier to find the correct answer, we will need to have an extremely precise classifier in order to be useful.

Imagine that we have a ML classifier that is able to correctly classify a possible solution to a CAPTCHA challenge on 99% of occasions, which is a figure that would be considered very good in typical circumstances. This means that our classifier will fail to correctly classify an answer 1 in 100 times. Let's imagine that this CAPTCHA offers challenges with 10,000 possible solutions (for example, a 100×100 grid), of which only 1 is correct and 9,999 are wrong. Our ML classifier would incorrectly classify $9,99 \approx 10$ solutions that are wrong as right. Now if we assume that it would also classify the correct solution as correct, we would need to blindly pick one from 11 solutions. By chance, we would be able to pass it $1/11$ of the times. A slightly worse classifier with 95% accuracy would give us a $1/51$ chance. If the CAPTCHA requires the user to pass two challenges our chances would go down dramatically.

Thus, we need to find a way in which we can still use ML classifiers even when the number of possible solutions is huge. We need a way to boost their classification accuracy in these situations. A possible way to do so, that we propose here, is to create *in-challenge answer ordering metrics*, that is, metrics that relate the different possible solutions among each other. That way, the classifier is allowed to compare solutions to the same challenge. One possible way to do so is adding metrics of order.

For example, imagine that we have created a metric M_1 that we want to use with a CAPTCHA H . Now, H gives us a challenge c . This challenge has N possible solutions, $s_1, s_2 \dots s_N$. We apply our metric M_1 and obtain $M_1(s_1), M_1(s_2), \dots M_1(s_N)$. This is the usual scenario. Now, we also calculate the order in which they'd be ordered by this metric result, that is, we order this list so that $M_1(s_{i_1}) \leq M_1(s_{i_2}) \leq \dots M_1(s_{i_N})$, so $O(M_1(s_{i_n})) = n$, that is, $O(M_1(s_{15})) = 5$ if and only if the value of $M_1(s_{15})$ is the fifth in this order. In a similar fashion, we create other derived functions that give us in which percentile each measure is, etc.

Note that a metric of order is equivalent to normalizing the results

of such metric within the values obtained for a single challenge. For example, the corresponding normalized results for $M_1(s_1), M_1(s_2), \dots, M_1(s_N)$, all metrics for a single challenge c with N possible solutions, in which $\alpha = \min(M_1(s_1), M_1(s_2), \dots, M_1(s_N))$ and $\beta = \max(M_1(s_1), M_1(s_2), \dots, M_1(s_N))$, would be $(M_1(s_1) - \alpha) / (\beta - \alpha), (M_1(s_2) - \alpha) / (\beta - \alpha), \dots, (M_1(s_N) - \alpha) / (\beta - \alpha)$.

This derived metric can have the ability to boost the classification accuracy because it can be extremely useful to the classifier in order to compare among solutions and pick the correct solution when there are several good candidates.

Next, we present an example of the use of answer ordering metrics in two real-world production CAPTCHAs.

Example: Capy and KeyCAPTCHA These two CAPTCHAs are examples of *Puzzle CAPTCHAs*, image-based CAPTCHAs in which the user has to drag & drop one or more puzzle pieces in their correct position, to revert the image to its original form. We analysed these CAPTCHAs in Chapter 3.

While we discussed these CAPTCHAs, we came to two conclusions:

1. The answer space is broad, in the thousands of options, as the image size is 400×267 pixels and the puzzle size is $\approx 76 \times 87$ in Capy, and 449×177 pixels for KeyCAPTCHA. Capy uses a 10×10 grid and KeyCAPTCHA a 5×5 grid for pointer movement. This means that the answer spaces are $\approx \frac{400-76}{10} \times \frac{267-87}{10} = 2916$ for Capy, and $\approx (\frac{449}{5} \times \frac{177}{5} \times .7)^n \approx 2225^n$ for KeyCAPTCHA, when using n puzzle pieces.
2. As only one solution is correct³, any classifier able to select the correct solution and only it would require a very high accuracy. This is because if only one answer from the answer space is correct, then for a CAPTCHA with an answer space size of $|as|$ and a classification accuracy of each answer of cl , the success rate with which we will pass the CAPTCHA challenge acc is given by

$$acc = cl \times \frac{1}{1 + ((|as| - 1) \times (1 - cl))} \quad (6.4)$$

³This is approximate, as both Capy and KeyCAPTCHA increase the user-friendliness by restricting the user to select from a grid of possible coordinates. This though has been accounted for in the previous calculation of the answer space.

In particular, if we want to pass the CAPTCHAs with a 10% success rate, then for Capy we would need a classifier that has an accuracy of 0.9969125, and of 0.9999999918 for KeyCAPTCHA using $n = 3$ puzzle pieces.

We then created a promising metric, the file-size of the solution candidate image once compressed with a lossy algorithm (JPEG). This metric cannot be used directly per-se, or be fed directly to a ML algorithm expecting good results. The reason why is that maybe 124Kb is a small size for an image, where it is a big size for another one with less detail. This metric has sense only when compared for the same background image *and* puzzle piece.

That is why we decided to create another dependent metric, that was the position if ordered among all the possible answers, in ascending file-size order. Thus, one would always be the possible solution that, when compressed with JPEG, results into a smaller file-size, for that particular challenge . 2916 would always represent the bigger one.

When we used this metric with Capy, we saw that it was able to correctly solve 65% of the challenges just by always choosing the image with this metric equal to 1 (smallest file size after being compressed using JPEG).

This same metric of order was used for KeyCAPTCHA with slight variations. KeyCAPTCHA randomly presents puzzles with up to 3 puzzle pieces. Even in this case, this metric was able to correctly select the solution images (up to three) and break KeyCAPTCHA on 20% of occasions.

During a presentation of this work at the Cambridge Computer Laboratory, Prof. Markus Kuhn made a remark regarding the possibility that our attack was so successful thanks to the images having been previously compressed loosely (as if using JPEG), although the image itself is served by Capy using a lossless compression format. We performed an experiment to test this hypothesis, briefly described in section 3.9.

6.6.3.4 New tailored metrics (optional)

In some cases, especially if the CAPTCHA being analysed is of a new type or presents some new characteristic, it would be convenient to spend some time devising tailored metrics that can be based on readily available algorithms. To do so, the practitioner can try to find one or more qualities that somehow

characterize the correct solution, even if they are not always necessary nor sufficient to correctly identify a solution.

In this step, it is better to err on selecting too many possible metrics than the opposite. It would be the S/ML tools later the ones that would pick up the useful metrics for the classification. It is correct to argue that some ML algorithms present problems when variables are correlated, or when using too many dimensions with sparse data. But this is a problem that many modern ML algorithms can handle well. In any case, if the practitioner does not get promising results, she can optionally apply a dimensionality reduction algorithm as PCA.

The creation of tailored metrics is in any case an optional step. Nevertheless, in some circumstances we recommend to devote time to this idea, as the results of those metrics can sometimes be surprisingly good.

There are some questions that can guide the practitioner into finding possibly useful metrics for a new type of CAPTCHA:

- What is the high-level difference/s that characterizes the solution to the challenge? What qualities does it have?
- Is there readily available software that is somehow related to any of those qualities? If so, in which way is it related?
- What protection mechanisms are in place?
- Is there a simple mathematical formula that is more or less affected by those qualities or protection mechanisms?

If the practitioner finds a possible metric that might be related to some quality of the correct solutions, or might be less affected by a protection mechanism of the CAPTCHA, it would be advisable to include it in the metric pool to test whether her idea is correct.

Next, we will present two examples in which a new metric is devised for three CAPTCHAs with good results.

Example 1. Civil Rights CAPTCHA The CRC (discussed in chapter 4) *protects* the text images that contain the possible empathic reactions. To

do so, it uses Securimage to convert the words describing these emotional reactions into images (Figure 4.2).

When we were considering the metrics to use with these challenges, we noticed the lines that Securimage adds to the challenge images. They tend to be lines of almost constant width, and that follow a more or less straight direction. We realized that the words were always presented centred in the image. Thus, we tried to use regular black pixel-counting metrics by columns.

We avoided these two lines, as they would affect our pixel-counting metrics. We investigated whether the derivative of pixel count per column would be a better metric than just the pixel count per column. Thus, we were able to create a new metric, a derivative of a previous, well-known one, that was able to circumvent the effects of one of the noisy transforms applied to the challenges.

Example 2. Capy, KeyCAPTCHA and Garb These are *puzzle CAPTCHAs*, discussed in chapter 3.

When we were analysing possible metrics to use, we thought about a property that the correct solution has, and that wrong solutions *have in lesser extent*. This property can be called *regularity*. The correct solution is an original image. Thus, it has colours and textures that are somehow uniform: appear in adjacent pixels, and possibly other regions of the image. Wrong solutions instead show at least a portion of the puzzle void, that in Capy is filled with parts typically from another image. That means that the wrong solution will have *less regularity*, more colors and more textures than the correct one.

Typically, lossy compression algorithms take advantage of that redundancy. In particular, JPEG is able to transform it into space-frequency coordinates that latter become quantized and compressed. Thus, JPEG is able to compress more these images that are more regular.

In the previous sections we have introduced the metric of file size after lossy compression of the image as a go-to alternative in many cases for very simple yet useful image analysis. This metric was first used for the security analysis of CAPTCHAs in our previous work ((Hernández-Castro et al., 2015)).

We created a derived metric, that was the order in which the image

would be classified if ordered among all the possible answers, in ascending file-size order. Thus, 1 would always be the possible solution that, when compressed with JPEG, results into a smaller file-size, for that particular challenge (that particular background and puzzle piece). When we used this metric with Capy, we saw that it was able to correctly solve 65% of the challenges. This same metric of order was used for KeyCAPTCHA with slightly variations, and it was able to correctly select the solution images (up to 3) and break KeyCAPTCHA on 20% of occasions. It was also able to break the Garb CAPTCHA on 98% of the occasions.

6.6.3.5 Test of metrics (optional)

This step is optional, as it is not needed and a practitioner can proceed directly to the next ones to begin using the metrics just selected and defined. But it can be convenient, in certain scenarios, to try our metrics with some challenges *manually*. Some reasons to do so are:

- Check whether the metric is applicable to all the challenges. For example, a metric that analyses the histogram of colours might be less relevant, or even not work well, if some of the challenges are in grey-scale.
- Check whether the metric runs on all challenges. There can be the case, for example, that a metric that is derived from a library might not be able to analyse all challenges, if some of them are not compatible because of format, size, etc.
- Check if two metric results are highly correlated. This means that they are both providing basically the same information, and we might well choose just one of them. Highly correlated metrics can also pose a problem to some ML algorithms that assume independent input variables, and make them slower to converge to a solution. We discuss this further in section 6.6.4.
- Check whether the metric gives an apparent *too good* result. If this is the case, check this result by implementing an attack. In some cases, it might not be necessary to go further with an S/ML analysis if a metric is good enough to create a successful attack. In following this path, we are mimicking the decisions of a real attacker, that will look no further if she finds a successful path of attack. We will also focus on the most

vulnerable point of the CAPTCHA design, possibly giving valuable feedback to the CAPTCHA designer.

Note that the apparent lack of correlations among challenges and metric values is not and cannot be a decisive point in order to remove a metric from our set. The reason for this is that, even though a metric by itself might not seem to be adding information to the characterization of the challenge, it can still be useful when combined with the rest of metrics.

6.6.4 Phase IV: Statistical and ML analysis

Once we have a promising set of metrics, we will need a minimum set of correctly (and wrongly) solved challenges. Figure 6.1 shows a generic flow chart that can be followed to download and grade a number of challenges. This will provide us with labelled examples, that will now be useful. Note that the challenge answers should never be graded by ourselves just following the CAPTCHA description. Instead, it has to be checked always with the CAPTCHA server, as some CAPTCHAs accept close or plausible solutions as correct ones, and this greatly influences the CAPTCHA difficulty and security.

The number of solved examples we need depends mainly on the techniques we will use. The number will vary between a few tens for some statistical and ML algorithms to a few hundreds for others. In our particular case of application of BASECASS, this number has varied between 50 and a few hundreds. This labour-intensive part of the framework can be done by a third party, like Amazon Turk or even CAPTCHA solving services.

Note that DL techniques usually require much larger numbers of examples, but we will typically not use them when looking for low-cost attacks. In any case, it is not too difficult to gather large numbers of challenges and labelled answers, even though typically the answers will form an unbalanced set.

Once we have created a set of metrics, we will apply them to each challenge solution labelling it as correct or not, using the information from the previously downloaded challenges. This creates a categorized, training set that we can use for both statistical analysis and the training of ML algorithms.

The reader might wonder why to do an statistical analysis if we will

later apply ML algorithms to try to extract data relations. From an attacker point of view, there is no interest in this, as we can say that ML algorithms are more powerful and will find relations in data in some cases when statistical tools alone cannot, when the opposite does not hold. Yet, from the point of view of a CAPTCHA designer, it might be interesting to also do an statistical analysis, as if it is successful, it will render results typically easier to interpret than the ones found by ML. Thus, the statistical analysis is an optional part of BASECASS, and of interest only in certain scenarios.

This training data can be of very different nature. In some cases, it might be extremely unbalanced, with only one correct solution for each hundreds or thousands of possible solutions.

We will apply different ML algorithms to it to test which one is able to handle it better. For this reason, we will typically choose ML frameworks that incorporate a series of different ML algorithms, as Orange (Demšar et al., 2013) or Weka (Hall et al., 2009). Both are open-source ML frameworks that incorporate several different algorithm families and implementations of different varieties.

We also will need to decide how to measure the effectiveness of each different ML algorithm. One typical proposal is to measure the accuracy, but this is typically not very significant when confronted with heavily unbalanced training sets. In this case, some other statistics as the f_1 or the κ statistic will be much more relevant.

The next example shows how different ML algorithms cope with different data scenarios, in particular, with more or less skewed data distributions.

Example: FunCAPTCHA vs CRC-OCR: how different algorithms cope with different kinds of data We have used somehow related metrics both in our analysis of FunCAPTCHA (chapter 5) and CRC-OCR (chapter 4). With FunCAPTCHA, we used a histogram of gray-scales, as well as total pixel count. With the CRC, we used pixel count per column/s (both the raw data and its derivative) as well as total pixel count.

Even though the metrics we used are related, the training sets are very different in each case. The one for FunCAPTCHA is very skewed (one woman per seven men) compared to the CRC which, although not being uniform, is much better distributed among its 133 categories.

Table 6.7: Best classifiers for off-line gender recognition with FunCAPTCHA and OCR-recognition with CRC.

FunCAPTCHA			CRC-OCR		
Algorithm	Correct (%)	κ statistic	Algorithm	Correct (%)	κ statistic
Multilayer Perceptron	99, 19	0, 96	LibLINEAR	59, 35	0, 58
KStar	98, 94	0, 95	Random Forest	51, 30	0, 50
IB1	98, 91	0, 95	LogitBoost	47, 73	0, 47
IBk	98, 91	0, 95	VFI	45, 82	0, 45
LMT	97, 73	0, 89	NNge	42, 80	0, 42
Logistic	97, 59	0, 89	Naive Bayes	40, 59	0, 39
FT	97, 36	0, 88	Multi Class Classifier	38, 48	0, 37
SPegasos	97, 43	0, 88	IB1	36, 51	0, 35
Decorate	96, 85	0, 84	J48 graft	33, 45	0, 32
SMO	96, 83	0, 84	Random Sub Space	32, 59	0, 31

We used Weka in both cases, and try all the compatible classifiers. Table 6.7 lists the best results for each case. We can see that the different ML algorithms cope very differently with this kind of training data populations, and very few of them are able to cope with both. One such example is IB1, a nearest-neighbour classifier that uses normalized Euclidean distance, which is apparently a good strategy for relating colour histograms or pixel counting histograms.

It is interesting that there is no NN algorithm that gets a good result with CRC-OCR, but the best result with FunCAPTCHA is achieved by the multilayer perceptron. We can see that in both cases, different tree-classification algorithms can achieve good results (LMT, FT, Random Forest, J48). Also some SVM-derived algorithms do well with FunCAPTCHA, and none do well with the CRC-OCR. Vice-versa happens for Bayes classification algorithms.

In the next two subsections we are going to coment in further detail the kind of statistical and ML analysis that we can perform, as well as comment on some tools that can be used for the ML analysis. We will also comment on the use of DL in BASECASS.

6.6.4.1 Statistical analysis

The statistical analysis phase is completely optional. Statistical methods might give some insight into correlations of different metrics, measure the information of each, and thus provide possible valuable insight into some weaknesses. Yet, statistical methods are mostly limited to linear relations, and are less powerful than some ML algorithms.

During our statistical analysis, we can perform the following multi-variate analysis:

- Correlation of the different metrics among themselves.
- Correlation of the different metrics with the classification.
- PCA (Pearson, 1901) or Factor Analysis (Cattell, 1952), to discover where most variance resides, or to reduce dimensionality.
- Discriminant Analysis (Cohen et al., 2013), that can be used to check if some variables are useful as predictors, can also be used as a classifier, and can be slightly more powerful than logistic regression in some cases.

Some of these techniques assume *independence*, that is, assume that examples are randomly sampled. This is a limitation with unbalanced training sets.

6.6.4.2 ML analysis

The greatest advantage of ML algorithms over typical statistical analysis is that there are many ML algorithms that can cope well with non-linearly separable data, that is, classes that cannot be differentiated based on a linear division of its values. Some of them also cope well with unbalanced training sets. Finally, we can select among the different ML algorithms using different statistics to measure their success: accuracy, κ , f_1 , etc., depending on the problem.

As we are trying to mimic the path of attack that would follow a low cost attacker, we will seek to try as many ML methods as possible with the least possible effort, to search for the ML algorithm that performs best

with the data. This can be done with the use of ML frameworks that provide the following benefits:

- Offer a single data format that can be used with all the ML algorithms.
- Provide a set of default parameters for each ML algorithm.
- Provide a series of ML algorithms implementations that can be tested automatically.
- (Some of them) provide a grid-search method to search for the best parameters for each ML algorithm.

ML frameworks There are currently several ML frameworks that offer different ML algorithms, yet two of them are more notorious for their continued development, support of several algorithms, and additional options like automatic grid search of parameters: Orange and Weka. Recently Weka included Autoweka (Thornton et al., 2013), a wrapper that allows to solve simultaneously the problem of selecting a learning algorithm and setting its hyper-parameters for best performance.

There are other popular ML options that can be used in different scenarios. For example, there are ML libraries that provide implementation of several ML algorithms. One of the most notorious, because of its support and the many ML algorithms that implements, is Scikit-Learn (Pedregosa et al., 2011), a ML library that uses the Python programming language.

Deep Learning A plethora of new Open SW and libraries have appeared that simplify the creation and training of Convolutional NNs, which are especially usefull for image recognition, and also of DNNs in general, useful for many different tasks. Among these SW and libraries, we can cite Caffe (Jia et al., 2014), Theano (Bergstra et al., 2010), TensorFlow (Abadi et al., 2016), and Keras (Chollet, 2015). Supervised DL typically requires a very large sizes of the training set. BASECASS proposes a way to check the basic security of a CAPTCHA, trying to prevent it from leaking basic side-channel information that, once gathered with a few metrics, could be used to bypass the CAPTCHA. This can constitute a low-cost, side-channel attack. We will typically not have access to large labelled datasets that we can use with DL tools.

There are exceptions. In some cases when we might be able to automatically classify with a certain accuracy, it might be possible to use a DNN to improve on it. More interestingly, in some other cases, we will be able to use a DNN to learn high-level *features* in an unsupervised way (Larsen et al., 2015). The activation of these features can later be fed to a NN layer or other ML algorithm for further classification. This opens exciting new possibilities for automatic extraction of CAPTCHA parameter creation attributes, and side-channel attacks. This offers some very interesting possibilities that we have not analysed yet, but leave as future work.

Even though it lies out of the scope of BASECASS, image-based CAPTCHA developers can use these tools to check that their CAPTCHAs are strong enough against the current state-of-the-art in ML.

6.7 Step 3.- Parameter-based S/ML Analysis

CAPTCHA challenges are not generated just purely randomly, that is, they are not just random noise. Instead, some random values within a certain range (and with a certain distribution) are taken as parameters to generate a particular challenge. These parameters can be of various types and each one influences one particular aspect of the CAPTCHA challenge being generated.

These parameters remain nevertheless private during the CAPTCHA challenge creation process. We can only see their results in the particular challenge created. Nevertheless, it is possible to infer some information about these parameters and their values when:

- The original CAPTCHA creator is collaborating towards its security analysis, or
- The CAPTCHA design is public and/or its implementation has been published as Open Source, or
- At least some of the main parameters affecting the creation of a particular CAPTCHA challenge are obvious and can be inferred easily from the particular challenge.

If our previous analysis has not found important weaknesses, but has hinted at some possible cases in which the CAPTCHA can be solved, we

can examine them in closer detail. This might be the case when the rate of success of the best ML algorithm is overall low, but is higher and consistent in a particular subset of challenges. If this is the case, we might find that some parameter sets of values render particularly weak challenges, but these are not enough to render the CAPTCHA broken due to their low frequency of appearance.

Examples of question that this analysis can answer could be:

- For a CAPTCHA that uses images as backgrounds: does using only one background affect? Which background renders the easiest challenges for a bot? What happens if no background is used?
- For a CAPTCHA that uses colors: does the number of colors used affect the difficulty for a bot? Are certain colours easier than others for a bot? What happens if we use only one color? And if we use the maximum number of colors?
- For a CAPTCHA that uses sprites: if the sprite is related to other challenge elements (same color as text, same filling as background, etc.) does this affect its difficulty for a bot?
- For a CAPTCHA that uses puzzle pieces: how does the shape affect its security? And their size? Can they overlap? If not, how far away from each other can they be? how does the filling of the puzzle piece affect the CAPTCHA?

We can use these parameters and their values to divide the training sets created, possibly gathering a larger number of examples, and retrain and test again the ML algorithms tested (and optionally do some statistical analysis, like checking for correlations). In this sense, we might be able to find particular sets of parameter values that are less secure.

This can be turned into a successful CAPTCHA attack if it is possible for a computer program to detect these weak parameter sets and ask the CAPTCHA for different challenges until these parameter values appear. This can be very useful information for the creator of the CAPTCHA, to avoid weaker parameter sets.

This step is then a second, more detailed, iteration of the previous statistical analysis and ML analysis steps.

6.8 BASECASS summary table

The procedures used in each application of BASECASS can be summarized in a table, along with the results found. If, during the distinct phases of the analysis, BASECASS finds vulnerabilities that might be sufficient enough for a side-channel attack, and if such attack is feasible and within the ethics of each particular case, then we can also include the results of such attack. Depending on them, it might not be necessary to continue with the application of BASECASS, if the CAPTCHA is considered broken beyond a reasonable effort of correction.

The findings that result out of the different BASECASS steps can be summarized in a template table. This table is divided in different types of analysis, and at the end of each one we present the main findings. Each section of the table represents one analysis type of BASECASS. Some sections of the table are optional and dependent on the result of the previous sections. A template of such table can be seen in Table 6.8.

The different analysis done in BASECASS, which results are presented in Table 6.8, are linked to the different steps of BASECASS in the following ways:

1. The first step of BASECASS is a black-box basic security analysis (Section 6.5). After this step, the practitioner should be able to complete the parts of the BASECASS table corresponding to: “challenge space”, “Answer domain” and “Domain and range conclusions”.
2. The second step of BASECASS is also a black-box analysis, but using metrics on the challenges and answers, and statistical analysis and/or ML to find correlations among them. Thus, after completing this step, the practitioner should be able to complete the parts of the BASECASS table corresponding to: “Metrics”, “Test of metrics”, “Data preparation”, “Statistical Analysis” and “ML analysis”. If an attack is possible, its results should be shown in the “S/ML attack & results” sub-table.
3. The third part of BASECASS is similar to the second one, but taking into account the values of the parameters used to create the different challenges. If is necessary and possible to perform this analysis, its results should be shown in the “ML vs. parameter analysis” sub-table.

If in any of these steps we have performed an attack to check a

vulnerability, its description and results should be shown in the “Attack & results” sub-table. Finally, the “Conclusion” sub-table summarizes the findings.

Table 6.8: BASECASS summary table.

BASECASS <CAPTCHA >Analysis		
Name: Descrip- tion:	<Captcha name and challenge subtype, if many > <more detailed description >	
Challenge space		
Base problem:	Type:	<Basic category of the problem presented>
	Size:	<Estimation of base problem size>
CAPTCHA problem:	Domain:	<Detailed description of the specific problem presented by the CAPTCHA>
	Size:	<Estimation of size, compared to the base problem, and/or based on possible parameters that influence on the creation of each challenge>
	Distribu- tion:	<Distribution in which each challenge parameter value appears, whether it is uniform or not, and additional data. A Pearson's χ^2 test might be applied if enough data is available>
Answer space		
Maximum Range:	<Theoretical size of the set of possible values to answers>	
Range:	<Real size of set of possible answers>	
Ratio:	<Ratio >	<Ratio (if finite)>
Distribu- tion:	<Distribution in which each answer value appears. A Pearson's χ^2 test might be applied if enough data is available>	

Challenge space & answer space conclusions		
Is attack possible:	Yes/No	<Whether an attack might be possible or not based on the previous findings>
Description:	<How the attack works>	
Success:	<Real success rate with which the attack bypasses the CAPTCHA>	
Metrics		
Denoising:	<Whether any denoising technique is used. If so, comment which>	
Pre-processing:	<Whether any pre-processing technique is used. If so, describe it>	
Generic	<General purpose metric used # 1 > <General purpose metric used # 2 > ...	
Order	<Order metric used # 1 > <Order metric used # 2 > ...	
Specific/ Tailored	<Special metric used # 1 > <Special metric used # 2 > ...	

Test of metrics		
$metric_1$: <Check metric applies to challenges and direct information gain >		
$metric_2$: <Check metric applies to challenges and direct information gain >		
...		
$metric_i$ vs. $metric_j$: Check if both present highly correlated results >		
...		
...		
Is attack possible:	<Whether an attack might be possible using one of the tested metrics >, ..., ...	
Description:	<Which metric the attack uses and how >	
Success:	<Real success rate with which the attack bypasses the CAPTCHA>, ..., ...	
Data preparation		
Training set	Size:	<Number of training examples>
	Balance:	<How many of them are of each class>
	Notes:	<Optional notes about data cleaning, transformations, data distribution, etc. >
Statistical analysis		
Correlations	<Most correlated variables with answers and R-factors>	
Regressions	<Variables that are used in best linear regression, and error>	
ML analysis		
Selection:	<Selection criteria for fitness of ML algorithm>	
Best algorithms:	<List of N best performing ML algorithms>	
Accuracy:	<Accuracy of the N best algorithms>	
κ -statistic :	< κ -statistic of the N best algorithms>	

S/ML attack & Results	
If previous phase leads to an attack	
Possible?:	<Whether an attack based on the previous findings seems possible or not>
Description:	<How the attack works>
Success rate:	<If so, with which success rate it bypasses the CAPTCHA>
Observations:	<Any additional observations>
ML vs. parameter analysis	
Optional: if and only if phases before not lead to a successful attack and there is enough data on challenge production parameters	
For each combination of parameter, value(s), and interesting ML result:	
<Parameter name ^	Value/s: <Description of set of values for the parameter that lead to an interesting result > Best algorithm: <Best performing ML algorithm> Accuracy: <Accuracy of the best ML algorithm> κ -statistic : < κ -statistic of the best algorithm> ...
Attack & Results	
If previous phase leads to an attack	
Possible?:	<Whether an attack seems possible based on previous findings>
Description:	<How the attack works>
Success rate:	<Real success rate of attack>
Observations:	<Possible observations>

Conclusion	
Weak- nesses:	<Possible list of weaknesses found, in decreasing order of importance >
Broken?:	<If the CAPTCHA can be considered bypassed, and if so, the success ratio of the attack >
Work- arounds:	<If any plausible work-arounds would prevent this and similar attacks >

Appendix B presents an empty table that can be used as a template when applying BASECASS to a new CAPTCHA. A template can also be found online at <https://github.com/carlos-havier/BASECASS-template>.

6.9 Examples of application of BASECASS

In this section we will discuss the ways to validate BASECASS, as well as provide examples of its application to different CAPTCHAs. Some of the examples will be complete, that is, based on the sequential application of most of the steps of BASECASS to a CAPTCHA until this is found either resistant or broken. Others will be examples of partial applications, that is, applying parts of BASECASS to a particular CAPTCHAs - this will be the case of the reviews of attacks from the literature.

In particular, we will present the application of BASECASS to the three previous case-studies analysed. This application will be sequential, and all the relevant steps would be applied sequentially in each case, till results are found or we determine the CAPTCHA to have a basic level of security. Next, we will review two cases from the literature. We will perform a limited application of BASECASS to them using almost exclusively the public information provided in each analysis. Finally, we will present its application to a new CAPTCHA proposal that appeared in 2017, after BASECASS was already designed.

To validate BASECASS, we should apply it to a number of different new CAPTCHA proposals and check whether it produces or not interesting results with some of them. Then, we should wait a certain amount of time to check whether other CAPTCHA researchers find similar flaws to the ones

found by BASECASS that allow for successful attacks. This scenario is not practical for a number of reasons:

- CAPTCHAs typically evolve with time, thus a version analysed by different researchers might not be sufficiently related to our version.
- Some CAPTCHAs disappear, making it impossible for other researchers to evaluate their security.
- There are many CAPTCHA proposals, from amateurs, the academic world, and commercial. Analysing the security of a significant number of them is very costly. Many of them remain without a security analysis.

Some of these problems would be solved if the industry would agree on implementing their proposals on some form of CAPTCHA general test-bed, for all versions implemented, that researchers could use to gain further insight in their security. Nevertheless, no one has proposed to create such test-bed yet, and some CAPTCHA companies are moving towards Security by Obscurity, making such proposals less and less possible.

In order to gain some insight into the applicability and interest of BASECASS, we will revise the Case Studies from which it was created in order to verify its correctness, that is, that as presented here, it would be able to find many of the flaws we identified in these security studies.

To gain broader knowledge for it, we will also check whether it would have produced results in other cases present in the literature.

In the next sections, we will review our case studies using the BASECASS framework. We are interested in seeing what results are obtained in each particular case if BASECASS had it been applied to them. We will compare these results to our previous findings.

6.9.1 BASECASS analysis of puzzle CAPTCHAs

Puzzle CAPTCHAs are image-based CAPTCHAs in which the user has to revert the image to its original format. The human user is able to do so because she *understands* the image, and thus can detect what is misplaced, lacking, or wrong with it. In this section we will focus on the three puzzle CAPTCHAs previously studied in Chapter 3.

The selection of these particular puzzle CAPTCHAs was due to a number of reasons. In particular, Gurb is open-source and can easily be tested. Capy has been presented as a carefully designed CAPTCHA by a PhD in Engineering and as incorporating measures against typical image analysis mechanisms. It has also been extensively praised in various summits and competitions, winning widespread recognition. KeyCAPTCHA has been able to get a small market share of the CAPTCHA market, and also presents the interesting idea of non-aliased borders.

One interesting aspect of puzzle CAPTCHAs is that the number of potential solutions to analyse can be much higher than in other CAPTCHAs. This presents a higher challenge to a classifier, requiring much greater accuracy in order to yield a significant attack success ratio.

In the following paragraphs we will go into further detail into these CAPTCHAs, and will present the result of the BASECASS analysis for each one of them.

Application of BASECASS to Capy Here we will briefly comment some of the aspects of applying BASECASS to the Capy CAPTCHA, and present the result summary table of the application of BASECASS.

BASECASS first step is a black-box basic security analysis of the CAPTCHA. Among others, we have to uncover the interaction of the CAPTCHA with its server, or create an alternative way to interact automatically with it. Thus, first we analyse the interaction of Capy with its server, which is performed in a straightforward way. At a point of it, a whole *PNG* image is transmitted that contains a sub-image of 400×267 pixels (the challenge image) and, in its right part, a puzzle piece of approximately 76×87 pixels, that is present to the user below the challenge image. This size might vary as the puzzle piece shape can change. The user answer is sent as a string containing the succession of drag & drop coordinates that the user's pointer crosses in order to put this puzzle piece in place, coded using base 32.

In order to gather enough data, we first created a program to automatically download the images containing both the puzzle background and the puzzle piece from Capy. We detected that the answer is sent as a string containing the successive positions travelled by the pointer (mouse or finger, in a mobile device) while performing the drag & drop, encoded in base 32.

Following with the application of BASECASS to analyse the challenge domain, we determined that Capy was using four background images, and that the puzzle piece can have different shapes and be from any of these images. Also, the puzzle piece void inside the background image is filled with a portion from any of the four backgrounds available. four background images is not a high enough number for a CAPTCHA, as it would be possible to detect the background image and, with that information, learn the correct placement of the puzzle piece. As we wanted to know whether the *base problem* Capy is based on could be good enough for a CAPTCHA, we assumed from now on that Capy authors could easily augment the number of background images to thousands or millions, and proceeded assuming this.

BASECASS encourages us to compare the base problem space with the challenge problem space to have a basic understanding of their relative difficulty. To measure the size of P , we assumed that we limit the image size to that used by Capy. In that case, there are $(400 \times 267)^{8^3}$ maximum images⁴. For each one, we can select up to $(76 \times 87)^{8^3} - 1$ fillings for its puzzle piece (the size varies, but it is around 76×87 pixels). Each one, we can position in $\frac{400}{10} \times \frac{267}{10}$ different positions⁵. This is so because Capy restricts the movements to a 10×10 grid, to make it easier for the human users to find the correct position. This makes a total of $(400 \times 267)^{8^3} \times (76 \times 87)^{8^3} - 1 \times 40 \times 26 \approx 10^{219}$ images.

To measure the size of H , we can perform a similar calculation, but now with the real number of images, four. The number of possible puzzle fillings is then $\approx 4 \times (400 - 76) \times (267 - 87) - 1 = 233279$. The number of positions to place the puzzle piece is 40×26 . Thus, the total Size: 970 millions⁶.

BASECASS encourages us to consider calculating the distribution of challenges. In this case, the only parameters we can consider are the background, the puzzle piece shape, its position, and the filling used for the puzzle piece void on the background image. Although the parameters can be reconstructed once all the backgrounds are known, the cost of this analysis is out of scope for a low cost attack, so it is not produced in this case.

BASECASS also compares the possible answer space with the real answer space used in the CAPTCHA. The answer space is easier to calculate.

⁴Theoretical maximum different images of 400×267 pixels in 8-bit RGB space.

⁵Note that this is a maximum estimation. It is clear that images differing in one value for a pixel will be indistinguishable to the human eye.

⁶For comparison, this size is $\approx 10^{210}$ smaller than the theoretical maximum.

If we restrict ourselves to an image of the size of Cappy, the maximum possible should be $(400 - 76) \times (267 - 87) = 58320$. As Cappy restricts movements to a 10×10 grid, this is instead 100 times smaller, that is, $(400 - 76) \times (267 - 87) \approx 583$. That means that a random brute-force attack has a success rate of 0.17%, slightly high, but possible for a CAPTCHA according to some authors.

BASECASS recommends to determine if the distribution of answers is uniform or is instead skewed. As the answer space is not small, for this test to be significant we should collect a very large number of examples and their solutions, at least in the order of 25,000. This test is again too costly, and in this particular case was not performed.

We have concluded with the first step of the BASECASS analysis. Now, we have some basic data about Cappy, and we can proceed with the second step of BASECASS and define the requirements and metrics for the S/ML analysis. In this case, there is no need for denoising or any transformation, and we are going to process the images as they are. We need to define which metrics could be of interest. Among them, we listed:

- General purpose metrics:
 - Histogram of colours used: as Cappy fills the space where the puzzle piece should go with a sub-image, sometimes taken from another image, we consider that that would sometimes add colors to the image and modify the colour histogram. As the colour histogram in RGB is a 3D space, dividing it in bins would render a very big space to analyse. Instead, what we will do is clusterize it using a ML algorithm (K-means), and check how good the clusterization is (mean and variance of the distance to centroids) using different numbers of clusters ($k = 3, 5, 7, 11$).
 - Number of pixels detected as borders: we will use different border-detection algorithms and count afterwards what percentage of the pixels are detected as borders. The idea is that for a correctly-reconstructed image, there will be *less borders* than for the image with a puzzle piece.
 - Results of the ENT test: a number of general metrics, including the entropy, serial correlation, lossless compression rate, Monte-Carlo estimation of π , etc.
- Ad-hoc metrics:

- Size after compression: The idea of using compression results to extract information from a CAPTCHA is not entirely novel, and has contributed to break a CAPTCHA before ((Hernandez-Castro, Ribagorda and Saez, 2010)). In this case it has a special relevance, as an original image will typically be *more regular* than the same image with a puzzle piece filled with some other image. This *regularity* can affect how some compression algorithms work, in particular those that transform the image into the frequency domain, for example using the DCT, like the JPEG lossy compression algorithm, thus affecting the size of the resulting compressed image.
- Comparative metrics:
 - Order in size after compression: if the size after compression is a measure of goodness of the solution, a ML algorithm would be interested in knowing which is the smallest/largest one (or n) of the set of possible solutions to a challenge.
 - Order in number of pixels detected as borders: in a similar fashion, this will possibly serve a ML algorithm to improve the accuracy while classifying among a set of possible solutions.

BASECASS includes a step to test the performance of the different metrics, that was performed in this analysis. In this particular case we found an unexpected result while testing the performance of the different metrics. In particular, we were surprised by the good results of the metric that compared the resulting file size after JPEG lossy compression. In our off-line tests, this metric alone seemed well able to break the Capy CAPTCHA. According to BASECASS, we performed an attack based on this result. Note that this metric of order based on the JPEG file size behaved so well, was so accurate that we did not need to use a ML classifier in order to completely break the Capy CAPTCHA. Table 6.9 summarizes the results obtained with the application of BASECASS to Capy.

Table 6.9: Summary table of the application of BASECASS to Capy.

BASECASS Analysis of the Capy CAPTCHA	
Name:	Capy CAPTCHA.
Description:	Image re-composition.

Challenge space		
Base problem:	Type:	Image re-composition.
	Size:	10^{219}
CAPTCHA problem:	Domain:	Position a puzzle piece of approx. 76×87 pixels in a 400×267 image, restricted to 10×10 pixel grid
	Size:	970 millions
	Distribution:	Distribution of parameters unknown and not studied.
Answer space		
Maximum Range:	58320	
Range:	583	
Ratio:	$\frac{1}{100}$	
Distribution:	Distribution of answer distribution not performed.	
Challenge space & answer space conclusions		
Is attack possible:	No	Attack is not possible with our knowledge of the challenge and answer space.
Description:		
Success:		

Metrics	
Denoising:	No de-noising technique used.
Pre-processing:	No pre-processing technique used
Generic	Histogram of colours used Number of pixels detected as borders Results of the ENT test: entropy; χ^2 test; arithmetic mean; interpretation as a sequence of 24-bit X and Y coordinates for estimating π using a Monte-Carlo algorithm; serial correlation coefficient
Order	Order in size after compression (JPEG). Order in number of pixels detected as borders.
Specific/ Tailored	Size after compression using the JPEG lossy compression algorithm.
Test of metrics	
<i>JPEG size order</i> : metric is able to guess correct answer on a large number of cases.	
Is attack possible:	Yes
Description:	JPEG size order for a single image.
Success:	While testing this metric off-line, it seems to perform well enough for a successful attack, possibly with over 20% success ratio.

Attack & Results	
If previous phase leads to an attack	
Possible?:	Yes
Description:	Given an image, we position its puzzle piece in the 40×26 possible positions. The resulting image is compressed using JPEG. We choose as correct the position that renders the image that, once compressed, requires a smaller file size.
Success rate:	65% on an attack for 1000 challenges.
Observations:	
Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Small set of possible background images (4). • Artificially reduced set of possible answers (10^2 smaller than the possible 400×267).
Broken?:	Yes, with a 65% success rate.
Work-arounds:	<ul style="list-style-type: none"> • Increased number of background images through larger database and image alterations. • Broader solution space (larger size of images, more possible puzzle positions, increased number of puzzle pieces). • Challenge pre-filtering to limit the usefulness of our metric.

Application of BASECASS to the Garb CAPTCHA Garb is somehow similar to Capy in that both transform an image and ask the user to reverse the transformation in order to pass the challenge. In the case of Garb, the transform is a reordering of its parts. For the black-box analysis of the first step of BASECASS, we can notice that Garb divides the images in four parts

and shuffles them, asking the user to shuffle them again in order to solve the challenges.

The first step of BASECASS also recommends to create a way to interact automatically with the CAPTCHA. In this case, it is straightforward to interact with Garb, as it is Open Source. BASECASS also recommends to estimate the sizes of P and H . The challenge space of Garb is quite small, as it consists on the permutations of four elements, $4! = 24$. This is not a good idea for a production CAPTCHA, as can be solved by brute force with enough success rate ($\frac{1}{4!} = 4,16\%$). We check on the code that the distribution is indeed uniform. BASECASS also recommends to check the size and distribution of the answer space, yet in this case, each answer is characterized by a permutation that undoes the permutation applied by Gurb, thus the answer space is symmetrical to the challenge space, and of the same size.

The second step of BASECASS allows us to define a way to perform a S/ML analysis. In this phase, we think it is interesting to see if using our previously defined metrics, that somehow try to measure how *natural* an image is, we would obtain a similar success also for this slightly different puzzle CAPTCHA.

There are a few metrics though that are not of application for Garb, as they are not or little altered by the transformations that Garb uses. In particular, the histogram of colours used is not altered by the reordering of the parts, and some of the results of the ENT test will not change while others will vary very slightly with the re-orderings. The only metrics that will vary depending on the image reordering will be the number of pixels detected as borders and the size after compression by JPEG, as well as their respective metrics of comparison. As can be seen, the JPEG file size order metric is able to determine the correct answer in 98% of the cases.

Table 6.10 summarizes the application of BASECASS using these metrics to Garb.

Table 6.10: BASECASS analysis for Garb CAPTCHA.

BASECASS analysis for Garb CAPTCHA.	
Name:	Garb CAPTCHA.
Description:	Image reordenation of 4 portions of an image.

Challenge space		
Base problem:	Type:	Image re-composition through reordering.
	Size:	If we limit the image size to that used by Garb, there are $(150 \times 150)^{8^3}$ maximum images ⁷ . Garb divides the image in only 4 parts, but this is clearly insufficient. Going for a bigger image (225×225) and dividing it 9 equal parts, we can order them in $9! - 1$ incorrect ways. This makes a total of $(225 \times 225)^{8^3} \times 9! - 1 \approx 18.000$ million images.
CAPTCHA problem:	Domain:	Fixed number of possile divisions of images, and their permutations.
	Size:	Number of images: 62. Number of possible puzzle positions: $4! - 1$. Total Size: 1426.
	Distribu-tion:	Even though an analysis has not been made, as the source code is available, the initial chal-lenge positions are known to be pseudoran-dom.
Answer space		
Maximum Range:	4!	
Range:	4!	
Ratio:	1 : 1	1 : 1
Distribu-tion:	We have not conducted an analysis over the distribution of positions for answers, yet as the source code is available, they seem to be pseudo-random.	
Challenge space & answer space conclusions		
Is attack possible:	Yes	
Descrip-tion:	A brute force attack on the permutation used to shuffle will have a $\frac{1}{4!} = 4,16\%$ success rate.	
Success:	We do not proceed to such an attack, as we want to learn if the idea in which Garb is based is strong enough if it had enough backgrounds and possible permutations of them.	

Metrics	
Denoising:	No de-noising technique is used.
Pre-processing:	No pre-processing technique is used.
Generic	Number of pixels detected as borders.
Order	Order in number of pixels detected as borders. Order in size after compression (JPEG).
Specific/ Tailored	Size after compression using the JPEG lossy compression algorithm.
Test of metrics	
<i>Ordering by JPEG size</i> : While testing this metric off-line it seems to perform extremely well.	
Is attack possible:	Yes
Description:	Ordering by JPEG size: given an image, we re-shuffle the following the 4! possible permutations. The resulting images are compressed using JPEG. We choose as correct the permutation that renders the image that, once compressed, requires a smaller file size.
Success:	98% on an attack for 1000 challenges

Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Small set of possible solutions (4! permutations). • Small set of possible background images (it is possible to <i>learn</i> them by trial-and-error).
Broken?:	Yes. 98% using a new metric.
Work-arounds:	<ul style="list-style-type: none"> • Increased solution space through bigger images & more puzzle portions, allowing for more permutations. • Recommended bigger than 9 (so a brute-force attack would have 0.00027% success). • As these would not prevent the JPEG-based attack, we will need challenge pre-filtering to avoid usefulness of our metric.

As can be seen, BASECASS is able to detect the weaknesses of Garb and point out the flaws we found during our previous security analysis.

Application of BASECASS to KeyCAPTCHA KeyCAPTCHA is conceptually very similar to the Capy CAPTCHA. When we analyse it following the recommendations of the first step of BASECASS, we notice that the main differences are related to design details and implementation. In particular, it uses a white background that will affect the lossless compression size, and the puzzle pieces have anti-aliased borders, that avoid a perfect match. The number of puzzle pieces used per challenge is variable, among one and three.

In order to apply BASECASS to KeyCAPTCHA, the methodology suggests to create a way to interact semi-automatically with the CAPTCHA. In this case, the typical analysis resulted difficult due to several obfuscation techniques used by KeyCAPTCHA, included the fact that the images are not transferred as-is, but mangled.

The analysis of KeyCAPTCHA is not a complete BASECASS analysis, as we are starting from what we know from similar CAPTCHAs like Capy and Garb. Thus, as we wanted to try first how our metrics could work, and for this we would not need a large labelled dataset in which to train ML algorithms, a very small dataset can suffice. Thus, we decided to download 50 challenges from KeyCAPTCHA.

In a similar way to what we did with Capy, to determine the size of P in this case, we limit the image size to that used by KeyCAPTCHA, there are $(449 \times 177)^{8^3}$ maximum images⁸. Yet as KeyCAPTCHA depicts single objects that only take a fraction of the image space, a big proportion of pixels are background ($\approx 70\%$). So the maximum is around $(449 \times 177 \times .7)^{8^3} \approx 10^{128}$ images.

To determine the size of H , we consider the different number of images seen, that using mark & recapture, we estimate in 50. The number of places to extract the puzzle pieces from is $\approx (\frac{449}{5} \times \frac{177}{5} \times .7)$. The number of possible puzzle pieces is 3, 2, 1, so the total size: $\approx 50 \times (\frac{449}{5} \times \frac{177}{5} \times .7) \times ((\binom{2225}{3}) + (\binom{2225}{2}) + (\binom{2225}{1})) = 50 \times 2225 \times 1835858625 \approx 2 * 10^{149}$.

Similarly to Capy, it is very costly to determine the distribution of challenges over H . The challenge is made using the following parameters: background image (≈ 50), position for puzzle piece(s) (2225), puzzle piece type and number (from three to one). Not all the parameters can be reconstructed automatically, unless an exhaustive search is done and all the backgrounds are known. In this case, the cost of such analysis is out of scope of a low cost attack.

The maximum answer space would be $449 \times 177 = 79473$ if the user was able to move the puzzle piece to any location. The answer space in KeyCAPTCHA is limited to a 5×5 pixel grid on top of the background image, thus $\frac{449 \times 177}{5 \times 5 \approx 3178}$. We have not conducted an analysis over the distribution of positions for answers, as we have a limited set of correct answers. In this case, the cost of such analysis is out of scope of a low cost attack.

At this point, BASECASS recommends to prepare the S/ML analysis. We already had an idea of using a particular successful metric, the order based on JPEG file size. In this case though, we would need some pre-processing: the white background problem in particular is clearly going to alter the

⁸Theoretical maximum different images of 449×117 pixels in 8-bit RGB space.

⁹This size is $\approx 10^{114}$ smaller than the theoretical maximum.

usefulness of our size compression metric. As BASECASS suggests to try the metrics in each case, we did some experiments and saw that the results were indeed poorer than with the other CAPTCHAs. In this case, BASECASS suggests trying to de-noise or transform the challenge in order to be able to still use the metrics. We decided to slightly alter this metric trying two possible modifications:

- Change the white pixels in the image for random noise. If the puzzle pieces are put in a place that covers more random noise (former background), this will diminish the image file size after compression. This is so as random noise is hard to compress, even if compressing in a lossy way.
- Consider only solutions as acceptable if the puzzle pieces were placed on top of mostly ($> 90\%$) white pixels.

After some small tries, the first solution was chosen, so the challenge image was pre-processed by altering its white pixels with random noise. This alteration was not done prior to process the image using the other metrics.

BASECASS suggests to try the metrics and so was done with this CAPTCHA, now with a similar result to the previous cases. The order based on JPEG file size was able to correctly solve 20% of the 50 challenges downloaded, including challenges with one, two and three puzzle pieces.

Table 6.11 summarizes the application of BASECASS to KeyCAPTCHA and the results found. As can be seen, many of the findings and conclusions are similar to the ones found with Capy CAPTCHA, as both puzzle CAPTCHAs are similar in many aspects.

Table 6.11: BASECASS analysis of KeyCAPTCHA.

BASECASS analysis for KeyCAPTCHA	
Name:	KeyCAPTCHA
Description:	Image depicting an object with one or more puzzle pieces

Challenge space		
Base problem:	Type:	Image re-composition through substitution.
	Size:	$\approx 10^{128}$
CAPTCHA problem:	Domain:	Image re-composition through substitution.
	Size:	$\approx 2 * 10^{14}$
	Distribution:	Unknown.
Answer space		
Maximum Range:	79473	
Range:	≈ 3178	
Ratio:	$\frac{1}{25}$	
Distribution:	Unknown	
Challenge space & answer space conclusions		
Is attack possible:	No	A brute-force attack on the position of the solution would not be possible, because even though it would have a $\frac{1}{2225} = 0,045\%$ success rate attack for the 1 puzzle piece challenge, it would have approx. 0,00002% for 2 challenges with 2 puzzle pieces and proportionally less for challenges with 3 puzzle pieces.
Description:	A brute-force attack would be possible given the small set of images used for the challenges (50). It would be possible to pre-learn their solutions by trial and error (waiting for challenges with 1 puzzle pieces) and then answer all challenges correctly.	
Success:	We do not proceed to such an attack, as we want to learn if the idea in which puzzle CAPTCHAs are based is strong enough.	

Metrics	
Denoising:	No denoising technique is used.
Pre-processing:	Random noise is added to the white (background) pixels prior to applying the JPEG size metrics.
Generic	Number of pixels detected as borders. Results of the ENT test: entropy; χ^2 test; arithmetic mean; interpretation as a sequence of 24-bit X and Y coordinates for estimating π using a Monte-Carlo algorithm; serial correlation coefficient.
Order	Order of the size of the possible solutions after compression (JPEG). Order in number of pixels detected as borders.
Specific/ Tailored	Size after compression using the JPEG lossy compression algorithm.
Test of metrics	
<i>JPEG file size order</i> : In off-line tests the metric seems to perform well enough for an attack	
Is attack possible:	Yes
Description:	Given an image, we add random RGB noise to its background white pixels. Then, we position its puzzle piece(s) in the $\frac{449}{5} \times \frac{177}{5}$ possible positions. The resulting images are compressed using JPEG. We choose as correct the position that renders the image that, once compressed, requires a smaller file size.
Success:	20% on an attack for 50 challenges, with varying number of puzzle pieces each.

Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Small set of possible background images (≈ 50). • Unnecessarily reduced set of possible answers (5^2 smaller than the possible).
Broken?:	Yes. 20% success rate using a new metric.
Work-arounds:	<ul style="list-style-type: none"> • Increased number of background images through larger database and image alterations. • Broader solution space (larger size of images, more possible puzzle positions, increased number of puzzle pieces). • Challenge pre-filtering to prevent the use of our metric.

As can be seen, BASECASS is able to detect the usefulness of our new metric and check the success rate of an attack using a metric of order derived from it. It is also able to find the weaknesses found in our security analysis.

6.9.2 BASECASS analysis of the Civil Rights CAPTCHA

The first step of BASECASS requires us to analyse it and create a way to interact automatically with it. In the case of the *CRC*, this will be useful in order to download enough data for this analysis. In order to do so, its communications protocol was analysed, and we developed a program that allowed to download the text of the challenge and the three PNG images containing the possible answers. This tool also allowed us to post an answer to the *CRC* server and get back its result (either the challenge was passed or not). This first step also recommends that, when possible, we analyse its challenge space. Even though in their web-page they mention that their database of news is going to be updated regularly, after downloading 1000

challenges, we only found 21 news items. This number is insufficient because as the set of 21 news is not further protected as they are just regular texts, each one can be easily identified by a bot, so it is easy to download them all and assign a subset of *correct* emotional answers to each one of them. This also allows us to do a brute-force attack in which a program will *learn* the possible correct answers just by trial and error. BASECASS also recommends that we analyse the of these 21 news: both how many times they are actually presented to the user, and in answer space (positive and negative news). We find them to be it strongly biased towards negative news. Their appearances distribution remains similarly biased.

At this point we find that this part of the challenge is solvable by a brute-force attack, if the answers to each news excerpt are coarsely divided into positive and negative. As we do not know whether this is the case, we proceed to do some analysis of how the CAPTCHA server validates the answers. Apparently, the answer has to actually come from the set of three answers presented to the user.

In any case, this part of the challenge can be considered broken, that is, it does not add security to the CAPTCHA, because if the emotional answers could be read and classified into positive or negative, it would be straightforward to solve the challenges.

The first step of BASECASS also recommends to analyse the answer space, both theoretical and the real one used in the CAPTCHA. Note that, if we restrict the answers to one word, the potential answer space of P is not very large: according to some word lists¹⁰, there are around 167 1-word emotions, so adding a few of the the modifiers "very", "a bit", "totally" as the CRC does, we can get to 668 words and two words combinations.

After some initial interactions, we start seeing repetitions on the set of possible answers. This is expected, given that the amount of possible emotions that can be described with one or two words is limited. We download a set of 1989 challenges and manually classify the possible answers, which are 133. Most of them appear with more than one repetition, so we consider this to be the total set of possible answers (or a good approximation to it) for our further analysis. The distribution of their appearance is not uniform, with a Pearson's χ^2_{132} value of 482, 12 (this distribution is shown in Figure 4.5). This allows for a potential brute-force attack in which we will repeat the most frequent five answer (to avoid possible detection by repetition), that

¹⁰For example, at <http://wire.wisc.edu/quizzesmore/Emotionwords.aspx>

can pass the CAPTCHA with a 1.2% success rate.

We can now proceed to the second step of BASECASS, the S/ML analysis of the *CRC*. The answers of the *CRC* are protected using Securimage, a general open-source OCR/text CAPTCHA widely used, that offers many configuration parameters. In this case, Securimage is used with a static configuration, that includes two or three lines over the text.

In order to proceed with the S/ML analysis, we want to define what metrics to use. Initially, we choose quite simple metrics: the total pixel count, as some characters use more pixels than others, can give us an idea of the size (in pixels) of the characters used; we measure in relatively to the maximum. To be more precise, we also use the pixel count per column, and per groups of three and five columns.

When we decided to use these metrics, we realized that the lines introduced by Securimage might influence their result. A way in which we can minimize their impact is if we consider instead the differential in pixels, because a line that has approximately the same width and an horizontal component (that is, is not purely vertical) will use approximately the same number of pixels per column during its length. Of course, this still will alter the results of our metrics when the lines start and end, and also when they occlude parts of a character. But still, this might be a good way to, in general, decrease the influence of the lines over our metrics. Thus, we decide to add these differential metrics.

In order to *read* the text of each of the three images in each challenge, we define these metrics to extract from every image, and proceed to train a set of classifiers on them. We obtained the best classification results with Linear Regression and Linear Support Vector Machines (LibLINEAR) (Fan et al., 2008), attaining 59.3% accuracy. This means that in a challenge composed of 3 possible answers, we have 28.8% of correctly *reading* the three possible answers and 35% of *reading* two of them.

Note that sometimes we will need to correctly *read* less than the three answers in order to choose the correct answer. If one of the one or two answers read are from the correct category given the news excerpt, then we can try that answer as the correct one, with an improved percentage of success.

The metrics to use for the classification of the news bits are taken from basic NLP techniques. In particular, after some data cleaning removing

country names, stop-words, etc., we transform the words to their WordNet synset representations and to TF-IDF normalised vectors with a cut-off of two. In order to train our classifiers, we use 622 manually downloaded and classified news bits from the *Civil Rights Defenders*. We test different classifiers and different syset representations. Finally, we choose SVM Lineal, translating the texts to chains of WordNet hypernyms, which obtained 1.00 precision during our tests.

Following BASECASS, and given our promising off-line classification results, we put together a program that automatically downloads and answers *CRC* challenges, testing if its answer is classified as correct or not by the *CRC* CAPTCHA server.

After 1000 challenges, we obtained a success rate of 16.5% challenges correctly solved. Using an slightly improved version that memorizes previous results, we soon obtain a success rate of 20.7%. This result is good enough to consider the *CRC* CAPTCHA bypassed.

In the two following tables (Tables 6.12 and 6.13) we summarize the results of the application of BASECASS to the two challenge subtypes of the CRC, that is, the OCR part of the challenge, and the empathy part of the challenge.

Table 6.12 summarizes the application of BASECASS for the Civil Rights CAPTCHA to its OCR/text sub-challenge. We can see that, in this case, BASECASS would have found the same flaws that we were able to find in our security analysis, in particular the weak answer distribution, and the possibility of approximating it well enough using simple metrics and ML.

Table 6.12: CRC-OCR BASECASS Analysis.

BASECASS analysis for the CRC - OCR.	
Name:	CRC - OCR
Description:	Three 1/2-words expressions of emotions protected with Securimage.

Challenge space		
Base problem:	Type:	Optical Character Recognition (OCR) in English.
	Size:	1022000 ²¹¹
CAPTCHA problem:	Domain:	OCR for words and 2-word expressions in English typically representing an emotion or subjective stand.
	Size:	Number of word and word combinations: 133. Number of possible images: $2^{width \times height}$. Parameters: black & white images, two overlapping lines.
	Distribution:	Parameters do not seem to vary through all the challenges: two semi-horizontal lines, black & white images, same font used, one or two words from the previously mentioned set.
Answer space		
Maximum Range:	668	
Range:	133 counting single words and combinations of two words.	
Ratio:	5 : 1	
Distribution:	Their appearances are not uniform, with a χ^2_{132} of 482, 12, giving a p-value of 0 (or more precisely $2.32e - 41$).	
Challenge space & answer space conclusions		
Is attack possible:	Yes	A brute-force attack would be possible, given the small answer domain and not uniform distribution of answer appearances.
Description:	We could just reply picking randomly one from the top n appearing answers.	
Success:	With $n = 5$ (to avoid possible detection of a single answer), we would pass the CAPTCHA aprox. 1, 2% of the times. We seek to improve this result through ML.	

Metrics		
Denoising:	No denoising technique is used.	
Pre-processing:	No pre-processing technique is used.	
Generic	Black pixel count by columns, grouped by 1, 3 and 5 columns. Total black pixel count.	
Order		
Specific/ Tailored	Differential of black pixel count by columns, grouped by 1, 3 and 5 columns. The differential helps to counteract the effect of the semi-horizontal black lines added to the images.	
Data preparation		
Training set	Size:	1989 training examples, used for training and testing using 10-fold CV.
	Balance:	There are 133 classes.
	Notes:	The distribution of their appearance seems to be uniform within the different categories, with 59% positive, 36% negative and 4% neutral. Appearances for each of the 133 classes vary extremely, with from 1 to 27 appearances per class, and a χ^2_{132} of 482, 12.
ML analysis		
Selection:	Classification accuracy and κ statistic.	
Best algorithms:	LibLINEAR, Random Forest.	
Accuracy:	59%, 51% accuracy.	
κ -statistic :	0.58, 0.5	

S/ML attack & Results	
If previous phase leads to an attack	
Possible?:	Given an off line classification accuracy of 59%, an attack seems plausible.
Description:	Classification of the answer images in one of the 133 possible words, using LibLINEAR with the previously described metrics.
Success rate:	Combined for both OCR and Empathy: 20%.
Observations:	
Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Small set of possible answer values (133). • Appearance of answer values is not uniform ($\chi^2_{132} = 482, 12$, p-value = 0). • Set parameters for challenge generation with Securimage. Securimage not intended for protecting such a small word set.
Broken?:	Yes. 20% with simple metrics.
Work-arounds:	<ul style="list-style-type: none"> • Increase drastically set of possible answers allowing combinations and expressions not describing emotions. • More uniform appearance of answers. Much more varied parameters for challenge generation with Securimage: fonts, number of lines, colors, distortion level, etc.

Table 6.13 summarizes the application of BASECASS for the Civil Rights CAPTCHA regarding the empathy sub-challenge. Again BASECASS would have found the same flaws that we were able to find in our security

analysis, mainly the fact that the empathy challenge has not an answer domain big enough, plus it is possible to approximate a correct answer using some variations from well-known NLP techniques.

Table 6.13: CRC-Empathy BASECASS Analysis.

BASECASS analysis for the CRC - Empathy		
Name: Description:	CRC - Empathy A short news excerpt typically related to Human Rights.	
Challenge space		
Base problem:	Type:	Empathy
	Size:	A human emotional reaction or subjective stand on a subject. Depending on the classification, there might be 8 basic emotions (not including weaker and stronger variants, and also complex emotions based on these) (Plutchik, 1991), or up to 42 different emotions ¹²
CAPTCHA problem:	Domain:	Unknown, but seems to categorize the news excerpts in two categories, <i>positive</i> vs. <i>negative</i> .
	Size:	The news excerpts are from a set of 21 elements.
	Distribution:	Parameters do not vary through all the challenges: the news excerpts are a fixed.
Answer space		
Maximum Range:	Apparently, there seems to be a coarse discrimination only among positive and negative reactions.	
Range:	2	
Ratio:	$\frac{42}{2=21:1}$	
Distribution:	They are imbalanced, as 66% of the news excerpts are negative.	

Challenge space & answer space conclusions		
Is attack possible:	Yes	A brute-force attack would be possible, given the small answer domain and not uniform distribution of answer appearances.
Description:	We could just reply picking randomly any of the possible negative answers, even better, any negative answer that appears.	
Success:	If we would be able to read the 133 possible answers, and always pick the negative one, we would pass this part of the challenge 71% of the time.	
Metrics		
Denoising:	No denoising technique is needed (content is text).	
Pre-processing:	Some pre-processing can be done using the text categories on WordNet.	
Generic	Appearance, using TF-IDF.	
Order		
Specific/ Tailored	Three possible transforms using WordNet: no transform, synonyms, hypernyms.	
Data preparation		
Training set	Size:	643 training news excerpts from the Civil Rights Association, used for training and testing using 10-fold CV.
	Balance:	167 positive, 290 negative, 165 neutral.
	Notes:	English stop-words removed. TF-IDF with a cut-off value of two.

ML analysis	
Selection:	f_1^{13} and classification accuracy.
Best algorithms:	SVM Linear ¹⁴ using synonyms.
Accuracy:	90%
κ -statistic :	0,85
S/ML attack & Results	
If previous phase leads to an attack	
Possible?:	Off line classification accuracy is 90%, so an attack seems plausible.
Description:	Classification of the news excerpts in either positive or negative, using previously trained classifier.
Success rate:	Combined for both OCR and Empathy: 20%
Observations:	
Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Small set of challenges (21). • Small set of possible answer values (positive or negative). • Appearance of answer values is not uniform (χ^2_{20} with a p-value = 0.336).
Broken?:	Yes. With a 20% success rate using simple metrics.
Work-arounds:	<ul style="list-style-type: none"> • Finer emotion classification. • More uniform distribution of emotions.

6.9.3 BASECASS analysis of FunCAPTCHA

The first step of BASECASS is a black-box analysis of the FunCAPTCHA. The FunCAPTCHA gender recognition CAPTCHA renders 3D head models into 2D in gray-scale. It uses the same model for the male and the female. It uses different rotations and fields of view, so size comparisons are not straightforward. The lightning seems to change slightly in each rendering too.

BASECASS also requires us to create a way to automatically interact with the CAPTCHA analysed. FunCAPTCHA uses JavaScript code obfuscation and private-key cyphered communications (AES) on top of HTTPS to try to protect/hide its client-server communications. In order to bypass these, we decided to use browser automation.

BASECASS recommends us to compare the theoretical size of the base problem being used, and the actual size of the challenges being proposed by the CAPTCHA, as a way to compare its strength to that of the base AI problem. The size of P is infinite: there are potentially infinite images of faces of men and women. In order to compare the size of H , we downloaded 500 images. We noticed that the appearances of the heads seem to repeat, but when we compare them they are all different at the pixel level. The parameters that seem to affect the final images rendered per challenge are:

- Model selected: either male or female, as there is a single one from each.
- Rotation in the vertical axis: the rotation is never as strong as to hide the nose or render a side-portrait, so the angle is always in the 0 to π range.
- Distance of the camera or field of view: the distance seems to be from the head is partially cropped (but the main elements as eyes, nose, mouth always appear) to further away so that the neck and shoulders can appear too.
- Lightning: the illumination seems to change among scenes, but it is harder to precise how it does so just by looking at a collection of challenge images.

As can be seen from the previous remarks, the challenge space H in FunCAPTCHA is quite restricted when compared to P . It is not trivial to

reconstruct the values used to create each challenge image. Thus, we cannot easily gather an amount of information that would allow us to perform a quantitative, statistical analysis on the distribution of these parameters.

From a qualitative point of view though, we can mention some facts that might affect the difficulty of the gender recognition problem created by FunCAPTCHA, and thus its security:

- Only one model is used for each genre. Naïvely, this seems to be an over-simplification of the genre recognition problem.
- The rotation of the model is only done in the Y axis. This also reduces variability of the renders produced.
- The size of the head of the male model *seems* to be bigger than the female, when compared with other attributes (eyes, hair). This though is not straightforward to use, as the distance of the camera from the head itself varies enough as to account for size variations.
- The rendering is performed in gray-scale. We do not know whether this is a good option or not. Maybe it is, if the colours/quantities of hair and skin type change drastically from male to female models.
- There is no distortion added to the images, not local nor global. The background is plain white. Given that the human vision system is very good at recognising human faces, to the point that it can recognise a face in under $100ms$. (Crouzet et al., 2010) (up to 50% faster than animals), and that we tend to recognise faces in almost-random noise, we think that the 2D renders could have been slightly protected with noise and distortions without affecting much the usability of the CAPTCHA.

The answer space is simple to analyse. There is only one answer, in the 3×3 matrix presented to the user, that portrays the face of the female. The drop target position is at the center. When we were studying FunCAPTCHA, we did not appreciate any deviation from randomness on the positions chosen to place the female. Also, the eight images are transferred to the client as independent images with associated numbers $n \dots n + 8$. We found no correlation between these numbers and the images containing the females.

Once finished with the first step of BASECASS, we can proceed to the second, the S/ML analysis. In order to do so, we have to determine which

metrics to use, and whether any de-noising, pre-processing or transformation would be beneficial. As the images were not altered in any way, it seemed that these would not be necessary, and we can process the images as they are. It does not seem useful to characterize the challenges at the pixel level. Not only we would have too many parameters to handle, they will probably be essentially meaningless.

Prior to the statistical and ML analysis, we needed to define which metrics could be of interest. We used some well-known metrics that gather some basic information from each image:

- General purpose metrics:
 - Histogram of colours (shades of grey) used, grouped in bins of different sizes: 5-values, 15-values and 25-values bins.
 - Number of non-white pixels (in % from the maximum).
 - Size after compression: gives an estimate of the amount of information contained.
- Ad-hoc metrics: we did not use ad-hoc metrics. We did not find any ad-hoc metric that we thought could be useful and relevant to this particular CAPTCHA.
- Comparative metrics: we did not use any comparative metric. The answer space is smaller than in other cases, as we have one correct answer in each eight cases per sub-challenge.

We decided to run the tests with these simple metrics and see whether they would allow for proper classification.

Even though FunCAPTCHA presents images that look similar to the eye, a pixel-level comparison always finds plenty of differences among them. Thus, a nearest neighbour comparison at pixel level does not seem appropriate. Yet the idea of nearest neighbour classification based in our metrics is appealing in FunCAPTCHA because:

- nNN classification is quite intuitive in this case. It can tell us which particular exemplar (class) is closest, up to n of them, and thus we can decide to influence our class choice by n results, weighted by distance of not. It also tells us whether our defined metrics are or not directly useful to compare the images.

- nNN does not require to choose many learning parameters.

The second phase of BASECASS recommends not to restrict ourselves to a single ML method, so we created a compatible ARFF data file and run all compatible algorithms available in Weka.

As there is a 8 to 1 imbalance in the training and test set, we choose to classify our classifiers according to their κ statistic value instead of the accuracy, less relevant in this scenario. We tested with different ML algorithms to determine those that were more successful. In particular, the MultilayerPerceptron, KStar, IB1/k, LMT, Logistic/SimpleLogistic and FT had all an accuracy over 97% and a κ statistic equal to or over 0.88. This implies than an attack using them might be feasible.

As per BASECASS, we proceed to the attack using the five best performing ML algorithms. We found that even such a basic attack is able to bypass FunCAPTCHA with a 90% success rate. Table 6.21 summarizes the application of BASECASS to FunCAPTCHA and the results found.

Table 6.14: FunCAPTCHA BASECASS Analysis.

BASECASS analysis for FunCAPTCHA		
Name:	FunCAPTCHA human gender recognition	
Description:	Select an image depicting a female out of 8 images	
Challenge space		
Base problem:	Type:	Image classification by gender.
	Size:	Unknown.
CAPTCHA problem:	Domain:	Gender classification of given 2D renders from two 3D models.
	Size:	Unknown, all 2D renders are different at pixel level.
	Distribution:	Cannot examine distribution given that the parameter creation values remain unknown.

Answer space		
Maximum Range:	8^n , where $n = \{1, 3, 5\}$	
Range:	8^n , where $n = \{1, 3, 5\}$	
Ratio:	1 : 1	
Distribution:	Their appearances seems uniform, no particular position or image number seems correlated with the female gender.	
Challenge space & answer space conclusions		
Is attack possible:	No	
Description:		
Success:		
Metrics		
Denoising:	No denoising technique is used.	
Pre-processing:	No pre-processing technique is used.	
Generic	Histogram of gray shades, grouped in bins containing 5, 15 and 25 values. Total non-background pixel count.	
Order		
Specific/ Tailored	Size after lossy compression (JPEG) using different quality settings.	
Data preparation		
Training set	Size:	4320 training images, of which 535 represent females, manually classified.
	Balance:	Approx. 1 in 8 are images depicting a female, as expected.
	Notes:	Test set made of 148×8 training images, of which exactly 1 in 8 represent females.

ML analysis	
Selection:	κ statistic.
Best algorithms:	Multilayer Perceptron, KStar.
Accuracy:	99%, 98% accuracy.
κ -statistic :	0, 96, 0, 95
S/ML attack & Results	
If previous phase leads to an attack	
Possible?:	Given an off line classification accuracy of 99%, which means $0,99^8 = 92\%$ per subchallenge, an attack seems plausible.
Description:	Classification of the challenge images as $\{male, female\}$, using Multilayer Perceptron trained with the previously described training set.
Success rate:	90% overall in all types of challenges served by FunCAPTCHA.
Observations:	

Conclusion	
Weak- nesses:	<ul style="list-style-type: none"> • Small set of possible challenge images, once basic metrics are extracted from the images: even though the set of parameters applied for challenge creation remains unknown, it is true that using just two 3D models seems too restrictive. • Lack of further protection mechanisms (distortions, noise, backgrounds, additional rotations, etc.)
Broken?:	Yes. 90% accuracy with simple metrics.
Work- arounds:	<ul style="list-style-type: none"> • Increase drastically set of possible parameter values: number of 3D models, rotations, lightning, maybe more models in the same render, etc. • Added distortions, noise, background, etc. • Remains unknown if with current ML state-of-the-art technology this would suffice.

The main problem seems to be in fact that the problem space of FunCAPTCHA is too small, much more than the base problem of gender recognition. BASECASS is able to find this using very simple metrics and well-known ML algorithms.

It is remarkable that using only general metrics, we are able to attain such good results both for off-line classification and during the corresponding attack.

6.9.4 BASECASS partial analysis of the QRBGS ‘Math’ CAPTCHA

In this section, we will present the application of BASECASS to another CAPTCHA proposal that has already been analysed from a security standpoint. A full application of BASECASS would be time-consuming and require a basic security analysis, which is out of place now that this CAPTCHA has been found flawed. Instead, we will apply partially our BASECASS framework, using only on the publicly available data of its published security analysis (Hernandez-Castro and Ribagorda, 2010). Using this data, we will check if BASECASS is able to find whichever weaknesses have been reported.

The first step of BASECASS requires us to create a mechanism to interact automatically with the CAPTCHA. As this is a partial application, and we will use the data already public, we do not need to create such tool.

The first step of BASECASS also recommends to relate the sizes of the theoretical base problem and the actual problem being generated by the CAPTCHA. Given the published data and accessing the QRBGS CAPTCHA on-line, we can estimate the sizes of both P and H . The QRBGS CAPTCHA offers four different challenge sub-types: an arithmetic expression, finding the smallest real root of polynomials (written in two different formats), and calculating a derivative on a certain point.

After interacting a number of times with the CAPTCHA, the numbers of elements in each subtype that we have seen are shown in Table 6.15:

Table 6.15: QRBGS challenge subtypes and space.

subtype	expression	example
arithmetic	$g_n(g_1(ar_1, ar_2) \dots, a_n)$	$-3 - 1 + (-5) - (-1) * 5 = ?$
smallest real zero of polynomial	$\prod_{n=0}^{i=0} p_i \times x^i$	$p(x) = x^3 + 8x^2 + 21x + 18$
smallest real zero of polynomial	$\sum_{n=0}^{i=0} (x - r_i)$	$p(x) = (x - 7)(x + 5)(x + 3)(x - 6)(x - 7)$
derivative	$\frac{\partial}{\partial x} a_1 \times f(a_2 \times x + a_3) + a_4 \times f'(a_5 \times x + a_6) _{x=a_7}$	$\frac{\partial}{\partial x} \left[5 \cdot \sin \left(5 \cdot x + \frac{\pi}{2} \right) + 6 \cdot \cos \left(\frac{\pi}{2} \right) \right] \Big _{x=0}$

In Table 6.15, g_i are binary functions, in particular addition or

multiplication, and ar_i are either single-digit integers or the result of an expression from g_i . In this subtype, it seems that $n < 9$. This leads to a size of $19^8 \times 2^7 = 2 \times 10^{12}$. For the polynomials, p_i and r_i are also single digit integers, and $n < 9$. So the polynomials expressed as powers of x have a size of $19^8 = 16,983,563,041$. The size of the set of polynomials expressed as factors is the same. For the derivatives, f and f' are the functions \sin or \cos , and a_i are either single-digit integers or rational multiples of π from the set $\frac{\pi}{2^i}$ where $i = 0..2$. This leads to a size of $(19 + 3)^7 * 2^2 = 9,977,431,552$ elements. The total is then $|H| \approx 2.2 \times 10^{12}$.

Given these restrictions, it is easy to see that P , which is solving arithmetic expressions, finding roots of polynomials and calculating derivatives, is infinite, whereas H is finite. This should not be a problem given that $|H|$ is big enough: the restrictions imposed on the coefficients being integer or multiples of π might not be too big of a constraint. But when Hernandez-Castro and Ribagorda (2010) download more than 10,000 challenges, they see a majority of repeated ones. The number of different challenges served is slightly less than 750. This is clearly a mistake, as now a learning attack is much easier to perform.

BASECASS also recommends us to check the distribution of the challenge space. We do not have data regarding the challenge distribution. Gathering this data would require *reading* the formulas to analyse the appearance of the different factors, which is clearly beyond the scope of the analysis (and would break the CAPTCHA by itself).

BASECASS recommends that we check the answer space and distribution. After checking different challenges, it seems that all the solutions are integers. This is clearly a mistake, as it reduces the solution space greatly from its potential, \mathbb{R} . Checking some challenges, it seems that all solutions are also small integers: we do not see any value even close to 100 or -100 . Then, Hernandez-Castro and Ribagorda proceed to check the distribution of answers. They find that the distribution of correct answers to each type of sub-challenge and spreads over just a few integer values, which further limits the CAPTCHA and makes it potentially weak against a learning attack. More so, the distribution is also extremely skewed, which means that we can randomly answer the most probable answers and still be able to bypass the CAPTCHA a significant number of times. The answer distribution is shown in figure 6.5.

As recommended by BASECASS, it was launched an attack to

learn how relevant and exploitable are these flaws. They learn that their straightforward attack reaches an overall 44% success ratio over all subtypes.

Table 6.16 summarizes the partial application of BASECASS to the QRBGS CAPTCHA and the results found.

Table 6.16: BASECASS Analysis for the QRBGS CAPTCHA.

BASECASS analysis for the QRBGS CAPTCHA		
Name:	QRBGS Mathematical CAPTCHA	
Description:	Read, understand and solve a mathematical expression.	
Challenge space		
Base problem:	Type:	Mathematical expression.
	Size:	Infinite.
CAPTCHA problem:	Domain:	Restricted mathematical expression.
	Size:	Up to $\approx 2.2 \times 10^{12}$, but actually less than 750.
	Distribution:	Unknown.
Answer space		
Maximum Range:	$ \mathbb{I} $	
Range:	≈ 200	
Ratio:	Infinite.	
Distribution:	Their appearances are very non uniform. The exact distribution is not studied.	
Challenge space & answer space conclusions		
Is attack possible:	Yes	An attack is possible given the very non uniform distribution of correct answers, the reuse of challenges and the fact that the CAPTCHA can be used as an Oracle.
Description:	Answering 0 to every challenge. Learning attack for the wrong ones: those not solved get future answers as 1, -1 , 2, -2 ... until answer is found.	
Success:	44% overall for all subtypes of challenges.	

Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Small set of possible challenges (750). • Challenges are always presented the same (no distortions, noise, backgrounds, rotations, etc.), so a learning attack is feasible. • Small set of possible answers (< 200). • Answer distribution is severely non uniform.
Broken?:	Yes. 44% success rate.
Work-arounds:	<ul style="list-style-type: none"> • Make distribution of answers more uniform and in \mathbb{R}. • Create the challenges dynamically so their number is closer to the maximum (2×10^{12}). • Protect challenges so a learning attack is more difficult.

It is worth noting that BASECASS is able to find the weaknesses of the QRBGS CAPTCHA in its first step, while checking the challenge and answer domains. That is one of the reasons why this step is important, and should be applied prior to other more involved steps, as the ones involving S/ML learning.

6.9.5 BASECASS partial analysis of the HumanAuth CAPTCHA

In this section we will present the application of BASECASS to another CAPTCHA proposal that, as happened with the Math QRBGS CAPTCHA, has already been analysed from a security standpoint. Our application of BASECASS framework to it will similarly be partial, based on the publicly available data of its security analysis by Hernández-Castro et al. (2010). At

the end of our partial application, we will check if BASECASS is able to find whichever weaknesses have been reported.

The HumanAuth CAPTCHA is an Open Source CAPTCHA that asks users to distinguish between images with natural and non-natural contents. The HumanAuth application comes with a image repository consisting of 45 nature images and 68 non-nature ones in JPEG format.

The first step of BASECASS strongly recommends to create a way to interact automatically with the CAPTCHA being studied. In this case, we do not need to develop a way to interact with the HumanAuth, as all its details are available in its source code package.

We can analyse the HumanAuth CAPTCHA as either a text-based CAPTCHA or an image CAPTCHA. Hernández-Castro et al. decided to do the second, so we will follow this route.

BASECASS recommends to estimate the size of the base problem and the size of the real problem being posed by the CAPTCHA and then compare them, in a way to estimate its strength compared to the base problem. The size of the images is 100×75 pixels, using 3 RGB channels with 8-bits per channel. The set of all possible images of this size, P , has thus a size of $|P| = 100 \times 75 \times 2^{8 \times 3} = 125,829,120,000$ possible images, even though this includes all images that differentiate from another in just a pixel and a bit - that is, many will look the same to the human eye. H is much smaller though, as it includes 45 nature images and 68 non-nature images, that are protected with the addition of a watermark. The watermark does not change, it just changes the position in which it within the image. The original watermark has a size of 16×16 pixels. Thus, there are $(100 - 16) \times (75 - 16) = 4,956$ positions for it. Thus $|H| = (68 + 45) \times 4,956 = 56,0028$ total possible images different at pixel level, although their differences are typically less than $100 \times \frac{(100-16) \times (75-16)}{100 \times 75} = 81\%$ different from many others, and as little as 16 pixels different (or less) than the closest one.

The first step of BASECASS also recommends to estimate the answer space of the CAPTCHA and its distribution, in a way to estimate its strength against brute-force attacks. The answer space of the HumanAuth CAPTCHA is reduced: we need to pick a number of elements from a set of 9. Thus, theoretically the number of answers could be $\sum_{i=1}^9 \binom{9}{i} = 2^9 = 512$. Yet HumanAuth presents always just 3 images to select, thus the answer space is the smaller $\binom{9}{3} = 9!/3! \times 7! = 8 \times 9/3 \times 2 = 12$ only different answers.

According to the source code, their distributions should be uniform.

Given the small answer space, and the fact that many challenges can be identified as having a similar image, as they are quite similar at pixel level, it might be possible to perform a learning attack against HumanAuth. This is not the attack that Hernández-Castro et al. perform, as they want to know whether the idea behind HumanAuth is sound, even if their image database was bigger.

After completing the first step of BASECASS, we can proceed to the second step, BASECASS S/ML analysis. To do so, it is necessary to choose some metrics that we will use to extract information from the challenges. Hernández-Castro et al. decided to use the ENT test for this. This test provides several numerical values for each image: the numerical value of the entropy, as measured by ENT in bits per byte; the χ^2 value for the corresponding degrees of freedom (width x height in pixels); the mean value of each byte; the value of π obtained using a Monte-Carlo algorithm that is supplied with the image data instead of a random stream; and the correlation of one byte against the next one.

Hernández-Castro et al. apparently used the whole set of HumanAuth as training images, checking them using CV. They obtained a 78% accuracy using Random Forests. This indicates that an attack might be possible.

In this situation, BASECASS encourages us to test our findings performing an attack. In order to test an attack, they create a set of 20,000 images using the provided watermark. They do so using the public source code available. The accuracy of the same classifier drops to 72%, but attain 91% using J48. Although they do not implement an attack, it is expected that with such accuracy, an attack would be successful on $0.91^8 = 47\%$ of occasions.

Table 6.17 summarizes the partial application of BASECASS to the HumanAuth CAPTCHA and the results found.

Table 6.17: BASECASS Analysis for the HumanAuth CAPTCHA.

BASECASS analysis for the HumanAuth CAPTCHA		
Name: Description:	HumanAuth image classification: artificial/natural. Select 3 images depicting a natural item from 9 images.	
Challenge space		
CAPTCHA problem:	Type:	Image classification.
	Size:	Infinite.
	Domain:	Image classification.
	Size:	560,028 possible images, derived from only 113.
	Distribution:	Uniform.
Answer space		
Maximum Range:	$2^9 = 512$	
Range:	12	
Ratio:	$\approx 42 : 1$	
Distribution:	Uniform.	
Challenge space & answer space conclusions		
Is attack possible:	Yes	A learning attack might be possible. Not tested.
Description:		
Success:		

Metrics		
Denoising:	No denoising technique is used.	
Pre-processing:	No pre-processing technique is used.	
Generic	ENT test suite:	
	<ul style="list-style-type: none"> • Mean value 	
	<ul style="list-style-type: none"> • Entropy per byte 	
	<ul style="list-style-type: none"> • Monte-Carlo value of π 	
	<ul style="list-style-type: none"> • χ^2 	
	<ul style="list-style-type: none"> • Serial correlation 	
Order		
Specific/ Tailored		
Data preparation		
Training set	Size:	20,000 training images.
	Balance:	Approx. 50% corresponding to each of the two classes.
	Notes:	Test done using 10-fold CV.
ML analysis		
Selection:	Accuracy.	
Best algorithms:	J48	
Accuracy:	91%	
κ -statistic :	Not reported	

S/ML attack & Results	
If previous phase leads to an attack	
Possible?:	Yes, an attack seems possible given the off-line classification results.
Description:	Classification of the challenge images using pre-trained J48 tree.
Success rate:	47% success rate is expected.
Observations:	Not performed in Hernández-Castro et al. (2010).
Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • Very small set of possible answers • Not enough large set of images • It is possible to correlate challenges, even while using watermarks
Broken?:	Yes. 47% success rate using general metrics.
Workarounds:	<ul style="list-style-type: none"> • Increase drastically the set of images • Add distortions and other measures to increase the difficulty of relating challenges and thus performing a learning attack • Increase the answer space by allowing different number of images to select

6.9.6 BASECASS analysis of CaptchaStar

CaptchaStar is a recent CAPTCHA developed by researchers of the University of Padua (Conti et al., 2016). It is based on a novel idea and does not have

similarity with any other precedent CAPTCHAs. It is based on the problem of re-composition of an image or detection of an image. This re-composition is not done directly over the image moving parts of it as in puzzle CAPTCHAs. Instead, it is done indirectly through the exploration of a search space by moving a mouse or a pointer. For simplicity, this search space equals the image dimensions, although this is not necessary.

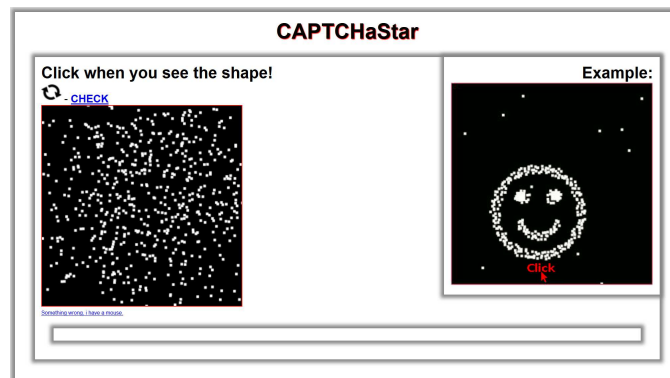


Figure 6.9: Example of a challenge produced by CaptchaStar.

CaptchaStar presents to the user a black & white image whose 4×4 -pixels have been reorganized, and move depending on the coordinates of the pointer - the mouse or a virtual cursor on a touch screen. An example can be seen in figure 6.9, where the image to the right shows the user how to solve the challenge, and the image to the left shows the current challenge. When the user moves the pointer, the pixels move. If the user moves the cursor in one coordinate, the pixels follow a different straight line, which varies per pixel and per coordinates. One mouse coordinate allows the user to see and understand the image. In this coordinate, the pixels appear as ordered as possible and represent some well-known item or icon, although with some noise. This coordinate is the solution to the challenge. Figure 6.10 shows an example of how the image transforms when the user moves the pointer over it.

BASECASS recommends to create a way to automatically interact with the CAPTCHA analysed. In this case, this was simply done through a program in Python that was able to download a challenge, send its answer to the CaptchaStar server and get the response of the CaptchaStar server,

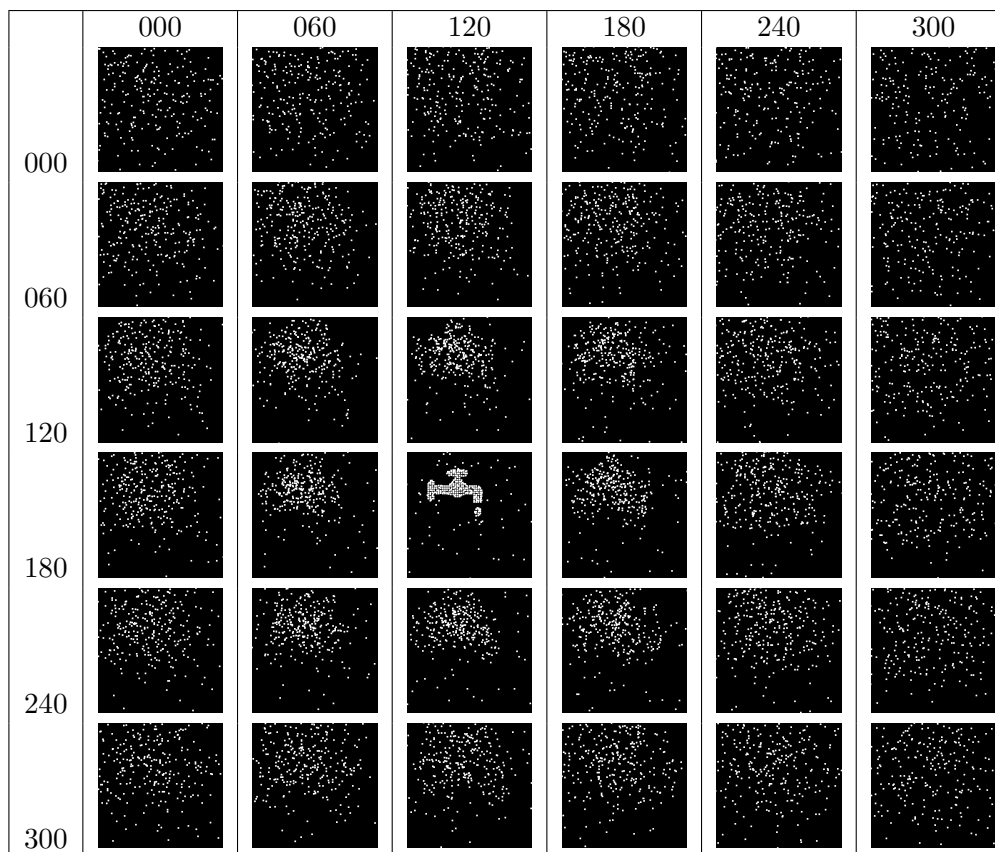


Figure 6.10: Renders of the same CaptchaStar challenge for different (x, y) cursor positions. The solutions can be seen when the cursor is over $(x = 120, y = 180)$ position.

all while recording a log of it. The correct answer was initially provided by humans through a replica of the interface of CaptchaStar. Later, it was found that CaptchaStar allows for requesting the validity of different answers for the same challenge, which allowed to use CaptchaStar as an oracle to find the corresponding solution.

BASECASS recommends us to compare the theoretical size of the base problem being used, and the actual size of the challenges being proposed by the CAPTCHA, as a way to compare its strength to that of the base AI problem, image recognition. In this case, the size of P can be very roughly estimated through how many different black & white images of 300×300 pixels can there be, if the pixel size for the image is indeed 4 pixels, and if we restrict ourselves to no more than 80% of the pixels in white. This would lead to $2^{\frac{0.8 \times 300 \times 300}{4 \times 4}} = 3.5 \times 10^{13}$. This is just an estimation, as many of these possible images would not represent a recognizable object or situation and could not be used as solutions. In order to compare the size of H , we downloaded 2000 images, and check that they were using 1631 different base images. Note that we have counted the number of images, but not the transformations performed on them, as they are unknown. When CaptchaStar presented the same image to the user, the transformation on its pixels was different, so there is theoretically no way for an attacker to reuse a previously-solved challenge to pass a new one. Even though the challenge space H in CaptchaStar is smaller when compared to P , thanks to the number of possible transformations, it is big enough to prevent brute-force attacks based on repeated challenges.

During our interactions with CaptchaStar, we were able to test that solutions that were not optimal were still accepted by CaptchaStar if they were up to 12 pixels from the optimal solution (see figure ??). This increases the user-friendliness, but reduces the search space. We determined that any solution in a 12×12 pixel square around the optimal solution would be accepted by CaptchaStar, reducing the answer space needed to explore to $\frac{300 \times 300}{12 \times 12 = 6250}$. This means that a brute-force attack would have a success rate of 0.016%.

The demo implementation of CaptchaStar allows to test several solutions for a single challenge. This facilitated to estimate the distribution of correct answers and compare it to an uniform distribution. After solving 5451 challenges, we produced a *heat map* (here plotted using Gaussian smoothing)

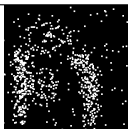
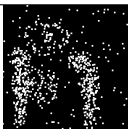
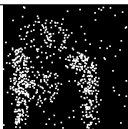
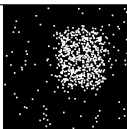
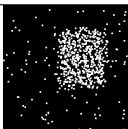
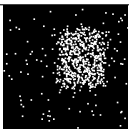
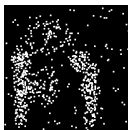

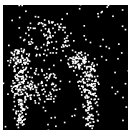

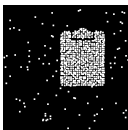
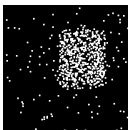
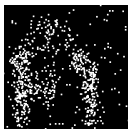

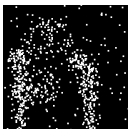
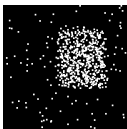
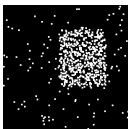
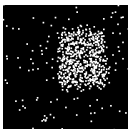
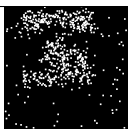
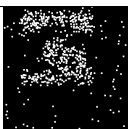
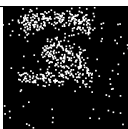









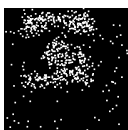





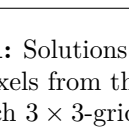
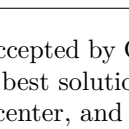
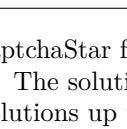
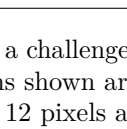
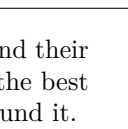













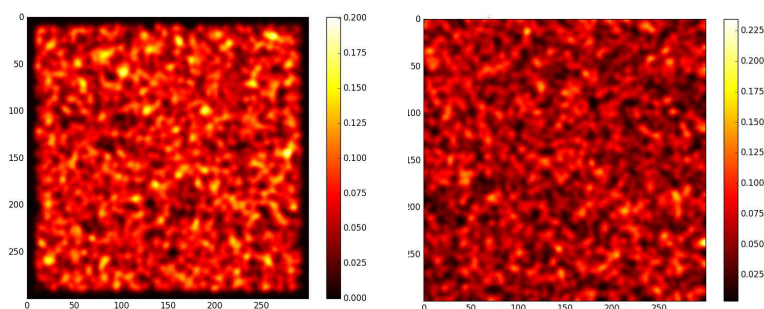
<div> <div>x offset</div> <div>y offset</div> </div>		-12	0	12	-12	0	12
-12	0						
							
							
-12	0						
							
							
-12	0						
							
							

Figure 6.11: Solutions accepted by CaptchaStar for a challenge and their offset in pixels from the best solution. The solutions shown are the best one at each 3×3 -grid center, and solutions up to 12 pixels around it.



(a) Frequencies of appearance of CaptchaStar centers of correct so- points for an uniform distribution. (b) Pseudo-random distribution of points for an uniform distribution.

Figure 6.12: Distribution of correct answers for CaptchaStar and for an uniform distribution, plotted using Gaussian smoothing. The correct answer distribution is close to an uniform, with exceptions around the borders.

of the centres of correct answers that can be seen in figure 6.12. As can be seen, CaptchaStar does seldom produce challenges which answers lie close to the borders. Interestingly, the distribution of peaks is more accentuated in the case of a pseudo-random uniform distribution than in CaptchaStar. The Pearson's χ^2 is 92474.15, indicating a $p - value < 0.00001$, which is a significant result that confirms that the answer distribution is not uniform (for a distribution with 89999 degrees of freedom at a significance level of 0,05).

Even when CaptchaStar allows for a margin of error of 12 pixels while answering, and even though the number of images used is limited, the transformations done on them and the semi-random choosing of the center of correct answers make it resilient enough to a brute-force attack. The fact that several answers can be tested using the CaptchaStar demo implementation allows for an Oracle attack, but this can be easily solved by the designers of CaptchaStar.

Once we completed the first step of BASECASS, we proceed to the second, the S/ML analysis. In order to do so, we have to determine which metrics to use, and whether any de-noising, pre-processing or transformation would be beneficial. The images were not altered in a way that keeps any visible hint of the original image nor that was easy to undo. It is precisely this alteration the one that embeds the problem on which the CAPTCHA is based, image recognition through a search space (or image re-composition).

Thus we decided to process the images as they are.

To define which metrics could be of interest, we picked-up some well-known metrics that gather some basic information from each image:

- General purpose metrics:
 - Results of the ENT test of randomness, as a measure of information and randomness on an image. The test is run in an un-compressed version of the answer images (uncompressed BMP format).
 - Size after compression: gives an estimate of the amount of information contained. We used the JPEG compression algorithm as implemented by the PILLOW Python library, with qualities of 1 and 95 (lowest and highest recommended).
- Ad-hoc metrics: we did not use any ad-hoc metric, as we did not find any ad-hoc metric that we thought could be relevant to CaptchaStar.
- Comparative metrics: the answer space is quite large, so we decided to follow BASECASS recommendation and create comparative metrics within the same challenge for all numerical results of the ENT test and for the size results. We did so normalizing all numeric answer ranges within the same challenge.

We decided to run the tests with these simple metrics and see whether they would allow for proper classification.

We had to determine the training and tests sets to use for ML. We used the 5451 challenges downloaded and answered in the previous step. In order to create a training/test set, we applied these metrics to the images resulting on placing the cursor on different positions. In particular, we created two sets:

1. The first one contained the images resulting when we divided the answer space in three parts, that is, when the possible coordinates are $(0, 0)$, $(150, 0)$, $(300, 0)$, $(150, 150)$, $(300, 150)$... $(300, 300)$. We included another 3×3 coordinates derived from positions at $(-10, 0, +10)$ offset of the coordinates from the center of the correct answers. This produced a maximum total of 18 images per challenge. In order to create the training/test files, we applied the mentioned metrics to these images. We will call this the *simple dataset*.

2. The second one contains similarly images resulting from dividing the answer space in five parts. Similarly, it contains another 5×5 coordinates derived from dividing the $[-10 \dots 10]$ offset range in five parts. Additionally, we added coordinates at $[-1, 1]$ from the center of correct coordinates. Note that, even though these coordinates are marked as correct by CaptchaStar, we marked them as wrong in order to see if some ML algorithms are able to differentiate which amongst almost-perfect and perfect solutions. In total, this produced a maximum of 59 images per challenge, to which we applied the mentioned metrics in order to produce the corresponding file. We will call this the *detailed dataset*.

We decided to use 3-fold CV for testing. We used the ML framework Weka, as it includes several classifiers that can be run using default parameters out-of-the-box. To determine which classifier performed best, we decided to use the κ metric, as it is more significant than the accuracy or others when dealing with unbalanced training and tests sets like the ones we have, with only 1/18 and 1/59 correct answers respectively.

We tested both datasets with different ML algorithms to determine those that were more successful. Tables 6.18 and 6.19 show the top classifiers by their κ metric for both the *simple* and *detailed dataset* correspondingly. Of a total of 163 classifiers in Weka, only a 37 and 34 correspondingly were able to load the data and present a solution within the time-out (5 minutes).

Many ML algorithms are able to classify the *simple dataset* with a high κ value. In particular, the *meta.RandomCommittee* (an ensemble of random classifiers), the *functions.Logistic* (multinomial logistic regression model with a ridge estimator) and two tree-based classifiers (*trees.RandomTree* and *trees.J48*) obtained the best results. They all obtain a κ of 0,99 and a perfect accuracy. This implies that an attack using any of them might be feasible.

For the second training/test set, the *detailed dataset*, we got worse results, as is expected. At the top of the scale there is again a *meta* classifier (an ensemble). The first pure classifier that reaches a decent solution is *J48*, with a κ of 0.36. Even though it is not too high, it should be enough as to perform an attack.

As per BASECASS, we proceed to the attack using the best performing ML algorithms from each training/test set. We select the pre-trained models for the *J48* trees for both the *simple* and *detailed datasets*, as well as the

meta.RandomCommittee, the *functions.Logistic* and *trees.RandomTree*.

We design an attack that downloads a challenge and creates all possible images related to answers in a grid of 5×5 pixels. After applying the metrics to them, it runs one of these pre-trained Weka model to choose the most promising answer. It then sends this answer to the CaptchaStar server to test whether it is the correct one.

The creation of possible answers to a challenge and the extraction of metrics from them is very time consuming, as there are theoretically a total of 9×10^4 possible answers. As we have determined that CaptchaStar allows for imprecisions of up to 12 pixels, we divide the answer space in 10×10 -pixel grids and analyse only the 900 possible challenge answers. This is not adequate for the models that we have trained to be more accurate when discriminating answers closer to the solution center (*detailed datasets*). Because of this, for this reason we try to use a search grid of just 2×2 -pixels with these models, while substantially reducing the number of experiments due to the very long experiment time. The results of these attacks can be seen in table 6.20.

We found that even using metrics that have not been tailored to CaptchaStar, we can find attacks that bypass it with a 85% success rate.

Table 6.18: Results of different ML algorithms on the simple CaptchaStar dataset, ordered by κ statistic.

classifier	κ	accuracy
meta.MultiClassClassifier	0.99	1.00
functions.Logistic	0.99	1.00
trees.RandomTree	0.99	1.00
trees.J48	0.99	1.00
meta.Bagging	0.99	1.00
meta.WeightedInstancesHandlerWrapper	0.98	1.00
meta.RandomSubSpace	0.98	1.00
functions.SimpleLogistic	0.98	1.00
functions.SGD	0.98	1.00
functions.SMO	0.98	1.00
meta.FilteredClassifier	0.98	1.00
meta.AttributeSelectedClassifier	0.98	1.00
rules.DecisionTable	0.97	1.00
bayes.NaiveBayesUpdateable	0.96	1.00
bayes.NaiveBayes	0.96	1.00
meta.AdaBoostM1	0.95	0.99
trees.HoeffdingTree	0.93	0.99
bayes.BayesNet	0.90	0.99
trees.DecisionStump	0.90	0.99
rules.OneR	0.90	0.99
bayes.NaiveBayesMultinomialUpdateable	0.03	0.51
bayes.NaiveBayesMultinomial	0.03	0.51
misc.InputMappedClassifier	0.00	0.94
meta.MultiScheme	0.00	0.94
rules.ZeroR	0.00	0.94
bayes.NaiveBayesMultinomialText	0.00	0.94
functions.SGDText	0.00	0.94

Table 6.19: Results of different ML algorithms on the detailed Captcha-Star dataset, ordered by κ statistic.

classifier	κ	accuracy
meta.RandomCommittee	0.76	0.99
trees.J48	0.36	0.98
meta.AttributeSelectedClassifier	0.28	0.98
meta.FilteredClassifier	0.26	0.98
bayes.BayesNet	0.20	0.89
bayes.NaiveBayesUpdateable	0.18	0.88
bayes.NaiveBayes	0.18	0.88
meta.MultiClassClassifier	0.08	0.98
functions.Logistic	0.08	0.98
trees.HoeffdingTree	0.08	0.97
rules.OneR	0.05	0.98
meta.LogitBoost	0.05	0.98
meta.AdaBoostM1	0.03	0.98
bayes.NaiveBayesMultinomialUpdateable	0.01	0.50
bayes.NaiveBayesMultinomial	0.01	0.49
functions.SimpleLogistic	0.00	0.98
functions.SGD	0.00	0.98
meta.Vote	0.00	0.98
misc.InputMappedClassifier	0.00	0.98
rules.ZeroR	0.00	0.98
meta.MultiScheme	0.00	0.98
bayes.NaiveBayesMultinomialText	0.00	0.98
functions.SGDText	0.00	0.98
trees.DecisionStump	0.00	0.98

Table 6.20: Attack success rates and mean running times per challenge when using different ML algorithms for classification and different search grids. The ML algorithms were trained using both the simple and the detailed CaptchaStar datasets.

dataset	model name	grid steps	examples	correct	%	mean secs.
simple	meta-Bagging.model	10	200	151	75.5	51.30
simple	functions-Logistic.model	10	200	170	85.0	49.97
simple	trees-J48.model	10	200	124	62.0	50.01
simple	meta-MultiClassClassifier.model	10	200	170	85.0	48.39
detailed	meta-RandomCommittee.model	10	200	36	18.0	48.18
detailed	meta-RandomCommittee.model	2	10	4	40.0	1774
detailed	trees-J48.model	2	10	5	50.0	1859

Table 6.21 summarizes the application of BASECASS to FunCAPTCHA and the results found.

Table 6.21: CaptchaStar BASECASS Analysis.

BASECASS analysis for CaptchaStar		
Name: Description:	CaptchaStar re-composition of image by exploration Select a cursor coordinate where the image shows a recognisable item	
Challenge space		
Base problem:	Type:	Image re-composition by exploration.
	Size:	3.5×10^{13} , not including transformations.
CAPTCHA problem:	Domain:	Image re-composition by exploration.
	Size:	Unknown: based on a limited set of images, but unknown number of transforms.
	Distribution:	
Answer space		
Maximum Range:	90000	
Range:	6250	
Ratio:	1 : 14	
Distribution:	Their appearances seems not uniform.	
Challenge space & answer space conclusions		
Is attack possible:	No	
Description:	Challenge space is big enough. Answer space, while not uniform, does not show exploitable flaws.	
Success:	-	

Metrics		
Denoising:	No denoising technique is used.	
Pre-processing:	No pre-processing technique is used.	
Generic	ENT test of randomness. Size after lossy compression (JPEG) using different quality settings.	
Order	All metrics are normalized within the same challenge.	
Specific/ Tailored		
Data preparation		
Training set	Size:	5451 training images, classified using Captcha-Star as an oracle.
	Balance:	Approx. 1 in 18 are right solutions (simple dataset), 1 in 59 in the second training set (detailed dataset).
	Notes:	
ML analysis		
Selection:	κ statistic.	
Best algorithms:	J48, Logistic.	
Accuracy:	1, 00.	
κ -statistic :	0, 98.	

S/ML attack & Results	
If previous phase leads to an attack	
Possible?:	Given an off line classification accuracy of 100%, with a κ of 0, 99, an attack seems plausible.
Description:	Classification of the answer challenge images as right or wrong, using different classifiers trained with the previously described training sets.
Success rate:	85% while restricting the analysis to increments of 10×10 -pixels.
Observations:	The models trained in the detailed dataset do not increase the success rate.
Conclusion	
Weaknesses:	<ul style="list-style-type: none"> • It is possible to use the demo site as an oracle. • The challenge gives away too much statistical information, making it possible to learn to determine the correct answer. • The base problem might not be strong enough.
Broken?:	Yes. 85% accuracy with simple metrics.
Work-arounds:	<ul style="list-style-type: none"> • Allow just one answer per challenge. • Allow the challenge answer space to have several areas in which a similarly-appearing sets of points gather trying to fool the metrics used and other possible metrics. • Increase answer space, shrink correct answer area, timeout answers. • Remains unknown if with current ML state-of-the-art DNNs this would suffice. It might be possible to train a Reinforcement Learning (RL) agent or a DCNN classifier to solve the problem.

CaptchaStar presents a novel idea for a Captcha, and apart from an easy-to-correct implementation mistake, it is quite well designed and has no other major design flaws. Unfortunately, BASECASS reveals that its base problem is not strong enough. It suffers from presenting *too much* information to the user, being the correct answer easy to characterize even through the simplest, non-tailored metrics.

It is important to note that using only general metrics, we are able to attain a very good success rate both for off-line classification (100%) and during the test attack (85%).

6.10 Summary of BASECASS

In this chapter, we have presented BASECASS, a methodology that guides a practitioner in testing a basic security level for many new CAPTCHA proposals. We have presented an overview of it, and next we have explained it in detail, including examples of some of its sub-steps. It is out of the scope of this work to test if our proposed methodology is in fact useful and efficient at finding possible weaknesses. But, when applied to the three case-studies presented on this dissertation, it has been able to find the weaknesses reported in them. More so, it has done the same for two more cases in the literature. Additionally, it has been successful in finding weaknesses and exploiting them in an attack against a novel and recent CAPTCHA design that was published after the framework was already created.

BASECASS is susceptible to be implemented as a tool in which plugins can solve the parts of it that need to be tailored to each CAPTCHA: the interaction with the CAPTCHA, the manual classification of a few examples, and most importantly, the selection of metrics. Even though the full analysis of BASECASS cannot be done automatically (for example, defining P and its size), most of it can.

It remains to be seen whether this methodology, or part of it, is put to use to test new CAPTCHA designs. Even if not, we hope to provide here valuable insight and ideas on some possible, original ways in which to test a new CAPTCHA design for a basic level of security.

Chapter 7

Conclusions and future work

This chapter summarizes the conclusions of this dissertation. It also presents some ways in which the work introduced here may be extended.

7.1 Conclusions

The recent advances in ML imply that some of the typical problems considered *AI-hard* can no longer be used as a base for CAPTCHA design as-is. Even though these recent advances benefit from large labelled data-sets, there is an increasing research into unsupervised training. If this is successful, there exists the possibility of integrating this new ML methods into BASECASS.

Most if not all CAPTCHAs in use today are susceptible to relay attacks. There are a few proposals that try to tackle this threat, but none have gained widespread use, and their resistance to relay attacks remains unknown. Also learning attacks, using CAPTCHAs as oracles, can be troublesome, and the solutions proposed so far do not work.

The main actors of the current CAPTCHA scenario offer solutions that are known to have vulnerabilities. They also increasingly follow the Security by Obscurity paradigm, that has a history of bad results in Cryptography and IT Security.

Given this situation, it is possible to think that the current security level of CAPTCHAs is not enough for protecting the services from automated

abuse. A possible reason of why such non-performing CAPTCHAs are in use is because in some scenarios, it might be better to have them and thus slightly increase the barriers to attacking, than to have no protection whatsoever.

In this dissertation we provide three case-studies of the security of commercial CAPTCHAs, in which we analyse five different CAPTCHAs that had never been analysed before. Interestingly, all of them present original challenges that have also never been tackled before: the use of empathy, gender recognition of synthetic faces, and restoring original images with puzzle pieces. We find weaknesses in them and confirm their exploit-ability through attacks.

The weaknesses found share some characteristics, suggesting the possibility of finding them following a semi-automatic procedure. This is the basis of the framework that we propose. BASECASS is a framework that suggests a series of checks on any new CAPTCHA proposal. These checks have to do with the challenge and answer space, and with unexpected leaks of information, that can be detected using ML.

We apply BASECASS to the three case-studies presented before and check that it actually finds the weaknesses reported. We also present to partial applications of BASECASS to two additional CAPTCHAs, and check that it also finds their vulnerabilities.

The problem of transmitting the hardness of the base problem to the CAPTCHA that uses it remains to be solved. Our methodology, BASECASS, provides a basic framework for at least comparing the sizes of H and P as a very rough estimation of the relative hardness of a base problem and a CAPTCHA that derives from it.

The problem of measuring the hardness of new CAPTCHA remains unsolved. We provided nevertheless a new framework, BASECASS, that is able to detect common weaknesses in a number of cases. More so, it is able to do so in a methodological way. The heavyweight lifting of the analysis is left to ML algorithms. We find BASECASS surprisingly successful, even using generic metrics. This result is quite unexpected given that many ML algorithms are designed with the expectation to receive *relevant* information to the task, that is, relevant *features*. This result is even more relevant in IT Security, as the cost of an attack is a very important measure, and this can also present a generic low-cost attack. That is the aim of this work, to increase the security of new CAPTCHA designs.

7.2 Future work

BASECASS proposes a step in which it is possible to link the S/ML analysis to the different values of the parameters used during the creation of a challenge. This has the potential to find weak values of parameters and avoid them in the production environment. In our case-studies, we have not been able to perform this analysis, as in most cases, the values of such parameters remained inaccessible to us: the CAPTCHAs were proprietary and it was difficult to extract such values from the challenges. In the case of the Garb CAPTCHA, it had only two parameters (image and permutation). Thus, this part of BASECASS remains untested, and we leave this verification as future work.

BASECASS is a framework than can be implemented as a software tool. The parts of it that depend on the specific CAPTCHA can be implemented as plug-ins. The part of it that needs a few labelled examples can be implemented through third-party CAPTCHA solving services. If implemented as Open Source, we hope that the research community would find it useful and that new CAPTCHA designers would use it to assess a basic security level for their designs.

New ML methods related to DL are gaining increasing efficiency at their tasks. Plenty of research is being done in unsupervised learning using these methods. It is foreseeable that in a near future, DL-related methods will be able to construct a high-level representation of almost any type of element (audio, video, images, text, etc.). Once we have such high-level representation, we can use it either with a DNN or with more typical ML algorithms. The activation of these features can later be fed to a NN layer or other ML algorithm for further classification. This opens exciting new possibilities for automatic extraction of CAPTCHA parameter creation attributes, and side-channel attacks. Their integration with BASECASS offers some very interesting possibilities that we have not analysed yet, but leave as future work.

Appendix A

Alternatives to CAPTCHAs

The different alternatives to CAPTCHAs can typically be applied to a subset of the problems that CAPTCHAs try to prevent. They also work at different parts of the threat model: threat prevention, attack prevention, attack detection and countermeasures.

Next, we will describe these protection measures and discuss their benefits and drawbacks.

A.1 Threat prevention

Threat prevention tries to minimize the threat. These mechanisms do not affect the vulnerability, the asset nor its value. They instead minimize the overall risk by reducing the threat, trying to avoid it taking place. This is typically done by discouraging the attacker, for example, increasing fines and other legal consequences, or decreasing the benefit extracted from the attack, which in turn minimizes the risk of the attack taking place. In our particular case the threat is that an asset, typically an on-line service that has been designed for people, is abused in an automatic or semi-automatic way, typically hundreds or thousands of times. The semi-automated attack is the one in which a third-party bypasses the protection mechanism in order for the attack to proceed.

A.1.1 Cost increase

The idea behind this proposal is to lower the economic incentives of *spam*, or in general, of repetitive automatic abuse of a service, by assigning to each automatic petition a small cost that would not alter the economics for regular users, but would for abusers. The cost can be monetary, but it is typically proposed to be a proof-of-work that requires some computing effort.

It is estimated that spammers worldwide and their associated move a market of 200US\$ million per year (Rao and Reiley, 2012). The idea of associating a monetary cost to every email sent was introduced by Dwork and Naor (1992), Back (2002). *Spammers* and *phishing* attacks typically rely on great numbers of emails sent to which a very small percentage of answers are received. Imposing a cost to every email sent would de-incentive these attacks, unless the expected revenue ROI (Return Of Investment) was positive, that is, would result in more revenue than the costs of such emails. Microsoft started the Penny Black Project to try to create such a proposal (Birrell et al., 2004).

As mentioned, some proposals suggest a PoW (Proof of Work) that is CPU-intensive and would require some processing time (Dwork and Naor, 1992). As CPU speed keeps improving, while memory available improves slower, there are also proposals for PoW that require a minimum amount of memory (Dwork et al., 2003, 2005). Some of these proposals also set puzzle difficulties based on a client's reputation, issuing "harder" puzzles to potential spammers (Le et al., 2012).

There are some critics of the PoW idea (Laurie and Clayton, 2004), as they claim that the difficulty level required for them would also affect regular users.

A.1.2 Spam bombarding

There is a history of retaliation actions against spammers. In 2005, Lycos created the campaign "Make Love, Not Spam" in which users downloaded a screensaver that connected back to spam web pages, slowing them down. Lycos was accused of performing a DDoS attack and its traffic was blocked by some ISPs. Some spammers retaliated forwarding back the requests to Lycos, launching a DDoS on it.

Similarly, Blue Security Inc. “organized their clients to bombard the spammers simultaneously with over half a million requests to stop spamming”. It then received a counter-attack from an spammer that completely block the company’s servers, not allowing it to do bussiness. The company had to shut down its web-site temporalily (Security, 2005).

There are solutions available that allow bombarding spam e-mail accounts, like SpamItBack . Another possible option is to bombard spam accounts or web-sites with millions of fake orders. The idea is to drive their profit margins down to a point where spamming is no longer economical (Rao and Reiley, 2012).

These ideas have not been successfull so far, either because of retaliation or adaptation of the attackers.

A.1.3 Money blockade

According to research, spam and other attacks use a number of botnets, web-sites, etc., yet 95% of it uses just a few banks (Levchenko et al., 2011) in St Kitts & Nevis, Azerbaijan and a Norwegian bank in Latvia. “The Latvian bank’s Norwegian owners say that the spam customers were inherited when they bought the bank, and claim that they have terminated their relationship with the spam affiliate programs” (Bright, 2011).

As the main bottleneck in spam seems to be the payment processors, some recommend that other banks refuse to settle credit card transactions with them, an approach already used in the US to block on-line gambling sites. It might not be easy though, with other payment systems available on-line, and it might involve retaliation.

A.2 Attack prevention

Attack prevention prevents a threat from materializing. As an example, updating some vulnerable software will prevent the threat of it being attacked using one of the vulnerabilities patched. Another attack prevention mechanism, this time for SW developers, is to use code analysis tools and/or protection libraries to make their SW less vulnerable to known attacks.

A.2.1 Alternate-channel validation

This mechanism consists in checking that the client is associated to some token that has a bigger chance of being in possession of a real human. A typical example is a mobile phone. Some companies offer telephone validation, either by SMS or by an automated phone call. An example is Ringcaptcha. The main drawbacks with this approach are both the lack of anonymity and the price, as for example, Ringcaptcha charges US\$49 per month if you are calling US numbers, although these prices increase for overseas.

A.2.2 Third-party identification

This mechanism consists in relaying the identification to a trusted third-party, the Identity Providers (IdPs). This is typically done using a SSO-like protocol (Single Sign-On), such as OAuth 2 and OpenID. These solutions are being sponsored by important IT companies such as Google, Microsoft, Twitter and Facebook.

OAuth/2 and OpenID Connect OAuth 2 supports OpenID Connect (OIDC), an authentication layer on top of OAuth (not to be confused with OpenID). OpenID Connect allows clients of different types (browser-based JavaScript apps, mobile apps, etc.) to launch sign-in flows and receive verifiable assertions about the identity of signed-in users, as well as additional identity information.

Although these solutions have not been developed with the intention to replace CAPTCHAs, we can naïvely think that their widespread use could decrease the need for CAPTCHA challenges being presented to the users. As can be seen, typically web-sites that rely on OpenID Connect/OAuth/2 allow bypassing the CAPTCHA mechanism (Figure A.1), understanding that if the user already has an account with an OAuth/2 provider that is *reliable* to them (one that implements better *bot* detection mechanisms), the CAPTCHA is no longer going to provide an increased level of security. In this fashion, we can think of third-party identity providers as a way to avoid using CAPTCHAs (as in Figure A.1).

¹Image modified from the web article “Secure your REST API with OAuth2 Implicit Grant”, at <https://www.ibuildings.nl/blog/2013/03/secure-your-rest-api-oauth2-implicit-grant>

OAuth/2 is an authorization protocol that allows third-parties to request access to some parts of the user's information in the provider account (i.e. Twitter posts, Google Mail contacts, etc.) and also perform some actions using such accounts (i.e. posting messages, sending e-mails, etc.). Much in the way Android applications requests permissions, OAuth third-parties can do the same regarding the access to the ID provider data and actions. For the user, the option is typically to grant all the permits, or not to use the third-party application or web-site.

Figure A.2 shows a typical OAuth authorization sequence. In this figure, Twitter is both the resource server (*owns/provides* the account) and the authentication server (is used to authenticate the user), but this is not necessarily the case. The user wants to use her browser (user agent) to access a web-page or start an on-line application (the client). This client allows for authentication through Twitter, so it asks the browser to get an authentication token from Twitter. Then the browser starts exchanging OAuth messages with Twitter in order to authenticate the user. This all happens behind the scenes, the user experience can be quite simple, as in Figure A.1, even though the first time the user has to authorize the access (Figure A.3).

Unfortunately, when a web-site or application uses OpenID Connect to verify an identity, it can also request additional information from the OAuth/2 provider (Figure A.3).

Security of OAuth2 There are currently more than 30 OAuth/2 providers, including well-known ones as *Google, Amazon, Facebook, Microsoft, Twitter, Yahoo!, Yandex*, etc. ².

The important number of applications and services that are clients of OAuth prevents from properly testing them. As an example, just for Twitter, its ecosystem of applications and clients had one million registered applications as of 2011, built by more than 750.000 developers around the World, with a new app registered every 1,5 seconds³. It is the users who have to decide if they trust a particular third-party, that is only known to them through the Internet, to access *all* or most of their private data ⁴. This is potentially a very important privacy risk.

²For a more detailed list of notable OAuth/2 providers, the Wikipedia maintains a page at https://en.wikipedia.org/wiki/List_of_OAuth_providers

³Figures from Twitter comment on their blog on 07/2011, at <https://blog.twitter.com/2011/one-million-registered-twitter-apps>

⁴For a discussion of possible privacy and security issues, a starting point can be "The

OAuth 2 does not support signature, encryption or client verification. It relies in TLS (SSL v3.0) for confidentiality and integrity. OAuth 2.0 is more of a framework than a defined protocol, thus interoperability is not guaranteed. It has been seen that its implementations has potential for many security flaws (Homakov, 2013a, Wang, 2014), so the IETF (Internet Engineering Task Force) has published a paper informing of its “Threat Model and Security Considerations” (Lodderstedt et al., 2013). Some experts consider it inherently insecure (Homakov, 2013b).

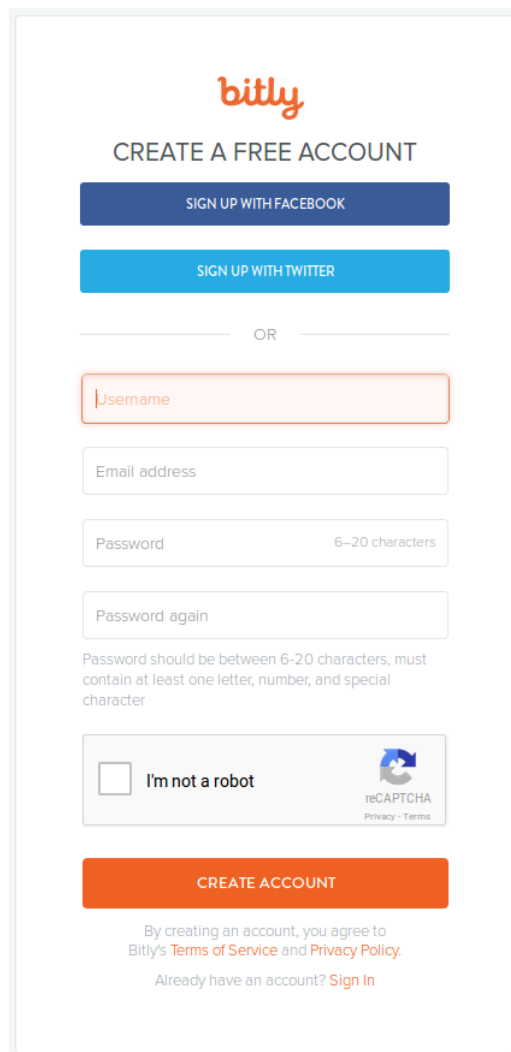
OpenID OpenID is similar to a SSO (Single-Sign-On) solution in that it allows using an existing account to sign on different web-sites and services. OpenID is also supported by well-known players as *Google*, *AOL*, *WordPress* or *Yahoo!*.

Figure A.4 shows a simple log-in example using OpenID, in which a user is requesting to log-in into a service (Service-now), which redirects her to log-in with her OpenID provider (if she is already not logged in). The OpenID provider sends back to the service provider the parameters containing the user’s credentials (typically her e-mail address, but can be others too).

Thus, there can be additional information linked to an OpenID account that can be shared with these third-parties: OpenID has an extension called Attribute Exchange that allows the transfer of user attributes from the OpenID identity provider to the relying party. These attributes can include the name, the gender, and many others required by the relying party. Additionally, the Identity Provider gets all the information from your OpenID logins, making it very easy to track an users’ activity on the Internet. These two facts also mean that OpenID represents a potential risk for the privacy of the users, as well as a single point of failure, as if the OpenID ID is compromised, an attacker will be able to impersonate another user in all the services secured by this OpenID provider.

Perpetual, Invisible Window Into Your Gmail Inbox”, at http://waxy.org/2012/02/the_perpetual_invisible_window_into_your_gmail_inbox/

⁵Figure taken from <http://wiki.servicenow.com/index.php?title=OpenID#gsc.tab=0>



The image shows a web form for creating a free Bitly account. At the top is the Bitly logo in orange. Below it is the heading "CREATE A FREE ACCOUNT". There are two buttons: "SIGN UP WITH FACEBOOK" (dark blue) and "SIGN UP WITH TWITTER" (light blue). Below these is a horizontal line with "OR" in the center. The form then has four input fields: "Username" (highlighted with an orange border), "Email address", "Password" (with a hint "6-20 characters"), and "Password again". Below the password fields is a text note: "Password should be between 6-20 characters, must contain at least one letter, number, and special character". This is followed by a reCAPTCHA widget with a checkbox and the text "I'm not a robot". At the bottom is an orange "CREATE ACCOUNT" button. Below the button is a line of text: "By creating an account, you agree to Bitly's [Terms of Service](#) and [Privacy Policy](#)." and another line: "Already have an account? [Sign In](#)".

Figure A.1: Logging-in with the possibility of using third-parties such as Facebook or Twitter, or alternatively registering using a CAPTCHA.

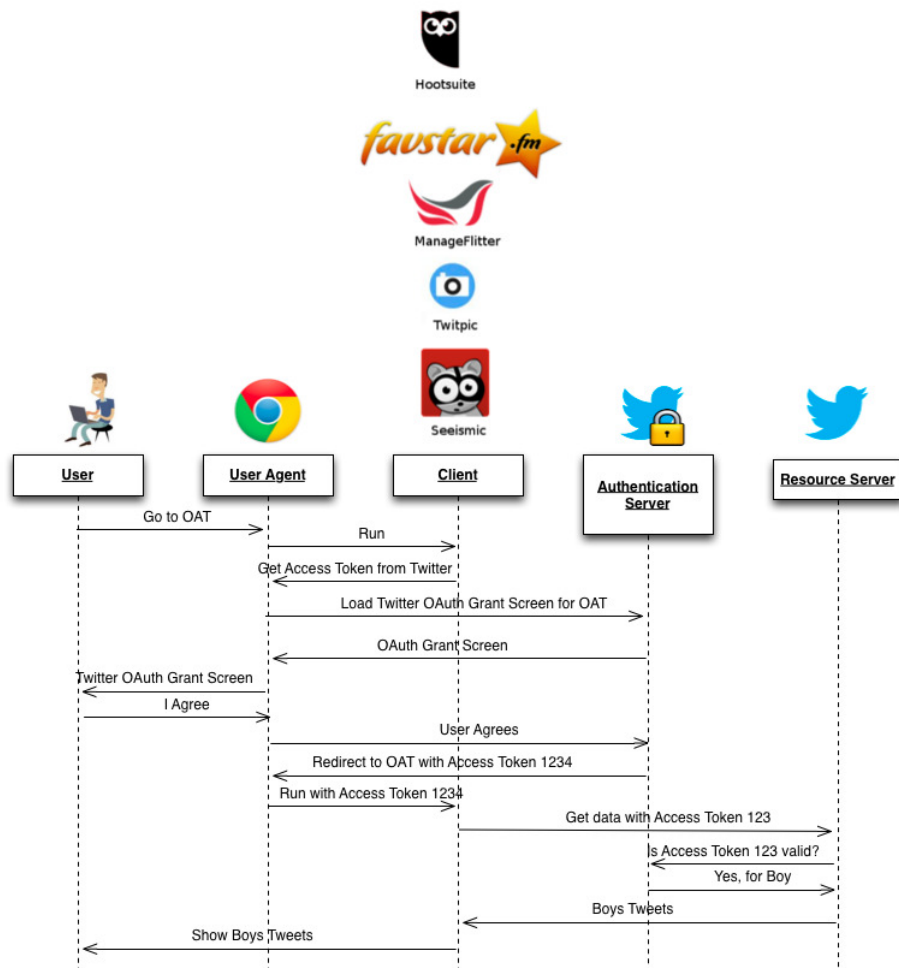


Figure A.2: Sequence of a third-party requesting access to a Twitter account ¹. The user, using her browser (user agent), wants to access one of many different clients (web-sites, applications, etc.) that accepts to log-in using her Twitter credentials. The client asks the browser to get a Twitter access token. The browser then requests such token from Twitter using the OAuth protocol.

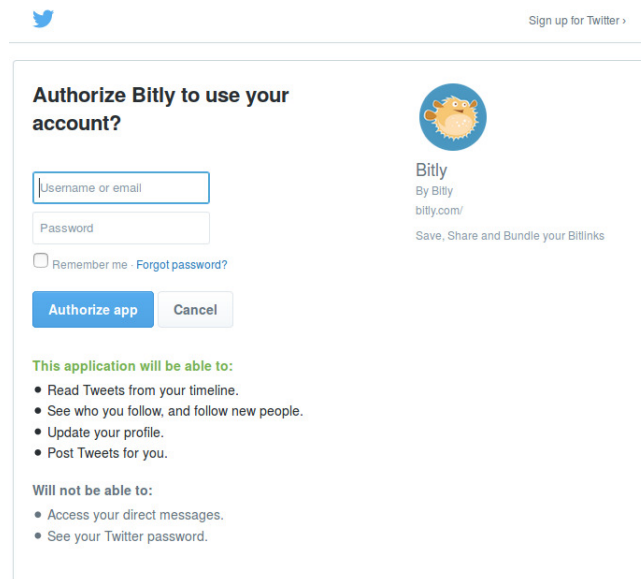


Figure A.3: Initial authorization to a third-party, showing the permissions that the application requests.

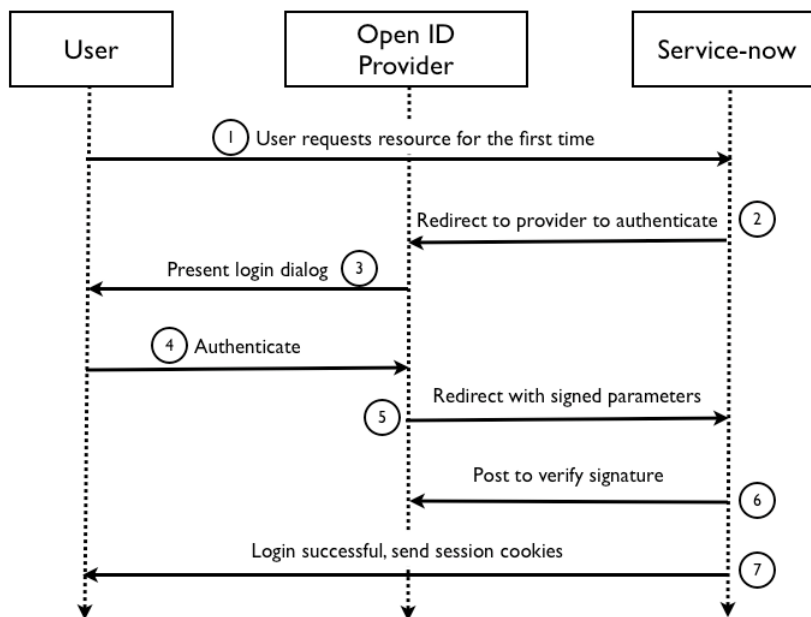


Figure A.4: OpenID login example ⁵. The user wants to access the Service, who in turn redirects her to the OpenID Provider for log-in. When the log-in is done, the OpenID provider redirects the user to the original Service being accessed, passing to it a signed authentication.

A.3 Attack detection

Attack detection does not prevent a threat or an attack, but allows us to detect that we are undergoing an attack. It is very important, as it will allow us to start other mechanisms to constrain the effects of the attack and possibly counteract it. The detection of an attack is closely related to the particular characteristics of the element to protect and the type of attack.

A.3.1 Form honey-pots

In IT Security, a honey-pot is a trap that is designed to be set off by an attacker only, and thus allow us to detect that an attack is taking place. Depending on the type and information that it can gather on the attacker, it might sometimes allow to identify the attacker.

Honey-pots have a long tradition in IT Security. There are network honey-pots, system honey-pots, and also data honey-pots, among others.

The honey-pot idea has also a variant to protect web forms, that is, any web page to which we can submit data. The idea behind it is that many automatic posting tools will try to fill-in all the fields of a web form. Thus, we can add to the web-form a field that is actually hidden from the user (using overlays or other mechanisms typically based on the style). If this field is not filled, we have a potential human client, whereas if it is filled we know that the filling agent is not an human.

Variants of this idea render the fields of the forms with random names, and only add the correct ones for the user to view using Java Script.

This is an example of an arms-race, as the defence focuses on a weaknesses common to some low-end attackers, and these will have to circumvent it.

Technologically it is not a sound alternative, as it is currently possible to create a simple web page reader/interpreter, for example using Computer Vision libraries as OpenCV, OCR Open Source SW as Tesseract and mechanization libraries such as Selenium. This way it would be possible to fill in the form correctly, even for a computer program.

A.3.2 Statistical and ML analysis of content

Here we will distinguish among comment and email spam, although both are closely related.

Statistical Analysis for Comment Spam The abuse of the ability to comment (review, etc.) on a web-site automatically is called *Comment Spam*. This involves the creation of automated comments that promote some element (as a web-site, for SEO), opinion or product. Among the possible detection mechanisms, the one mostly used is based on statistical analysis.

Spam detection is a case of *Text Classification*, a well-known AI problem. Statistical analysis relies on the analysis of the full content of the messages (email headers and content, or comment contents) and requires a training phase in which the users need to manually label each offending item as spam. With comment spam, the users will be the different blog owners. Typical anti comment-spam statistical tools create a database shared among their different clients.

This labelling allows the statistics to be recalculated, typically using Baye's Theorem (Equation A.1) or derivations.

$$Pr(S|W) = \frac{Pr(W|S) * Pr(S)}{Pr(W|S) * Pr(S) + Pr(W|H) * Pr(H)} \quad (A.1)$$

where:

- $Pr(S|W)$ is the probability that having the word (or feature) W in the message, this is *spam*. This is the probability that we want to calculate.
- $Pr(S)$ is the general probability of a message being *spam*. Some authors think this probability is 0,8 or higher.
- $Pr(W|S)$ is the probability of this word (or feature) appearing in spam messages.
- $Pr(H)$ is the probability of any message not being spam, so $Pr(H) = 1 - Pr(S)$. H comes from the word “ham”, as any message not being

spam is considered *ham*. The word *SPAM* appears to originally come from the canned meat sold by the company Hormel, meaning “spiced ham” (chopped pork shoulder meat with ham, salt, water, sugar, and sodium nitrite). Its association with repetitive, bothering messages seems to come from the Monthly Python’s “SPAM song”. Thus, real “ham” is the opposite to “spam”.

- $Pr(W|H)$ is the probability of this word (or feature) appearing in *ham* messages.

The statistics are calculated over the presence of *features* that typically are characters, pairs of characters (*digrams*), bags of characters, bags or words, pairs of words, groups of three or more words, etc. Naïve Bayes spam filtering is a very old spam classification technique (Pantel and Lin, 1998, Sahami et al., 1998), but with a big enough training set it can give a low false positive rate while still detecting most spam.

Words with very ambiguous meanings or little semantic content are typically dismissed. This is typically the case also for words that do not appear frequently enough. Other additional heuristics can be used to improve the results.

Statistical Analysis for Email Spam The abuse of email comes typically in the form of *spam*, that can also include *phishing* attacks. Several possible detection mechanisms exist. Among them, the most typically used are based on statistical analysis, shared with comment spam protection. The idea is to identify automatically or semi-automatically the offenders. Even though the basic mechanism is similar, the training and other specifics are different.

E-mails present a different scenario than blog posts and responses. For example, e-mails can contain HTML code that embeds or links to other resources. This allows for other means of attack.

When we want to apply Statistical Analysis to prevent email spam, we can consider that some of the classification of the examples (spam vs. not spam) can be shared amongst email users, but not all. Some part of this classification is done by each email user, and the statistics can be tailored to each user. Thus, the word “viagra” would have a distinctive chance of representing spam for a gymnast and for a pharmacy worker. The system will learn this automatically after the labelling of each one of them.

One of such attack techniques includes using images instead of words. Some e-mail providers apply OCR to them. Ironically, spammers started applying obfuscation techniques to spam images in a way similar to how some OCR-based CAPTCHAs work. With this, they prevented not only the success of OCR tools, but also try to avoid signature detection. In any case, image-based spam declined in the 2008 for a slow rebirth in 2011.

Evasion attacks Statistical Analysis has a potential drawback: it is possible to modify spam messages in an adversarial way -evasion attack- that will bypass statistical filtering, as well as some ML classification mechanisms. A typical evasion attack consists of adding good words to the message to increase their likeliness of being classified as *ham*. There are several versions of the the good-word attack (Wittel and Wu, 2004, Lowd and Meek, 2005, Bishop et al., 2010, Chan et al., 2011, Biggio et al., 2011, 2012, Zhou et al., 2012, Chan et al., 2015) that are able to bypass Statistical Analysis, even for short messages (Chan et al., 2015).

Database poisoning In the case of Comment Spam detection, each particular classification service typically has a shared database among its users, to increase its training base, as well as to make it easier to manage its learning. There are several potential problems with this approach, among them impersonating service users and poisoning the database. This attack consists on filling the database with incorrectly classified examples in order to affect its future classifications. This attack is also possible against ML classifiers.

Efficiency of Statistical Analysis It is difficult to find official statistics of the error rate of these comment spam detection services. Regarding the most well-known of them, there seems to be controversy about their error rate, and especially about their false positive rate. The only study that has analysed the two most used services in some detail (Ramilli and Prandini, 2009) has found that an extremely simple attack in which different sentences for different message parts are combined randomly is able to successfully bypass them. The authors devise a new filtering mechanism based on similarities among different comments, but its robustness remains to be checked.

There is also a lack of knowledge of how much of the current comment spam is done automatically and how much (if any) is done manually by third-party players. As CAPTCHAs are solved remotelly in low-wage countries,

there is also active seeking by spammers of human labour in order to run spam-related tasks (Ipeirotis et al., 2010). It is unknown though how much of current spam comes from this source.

Akismet Akismet is the most well-known service for comment spam detection. Akismet is a *home-grown* comment filtering from Automatic, the makers of WordPress. They have been in the business since 2005. It is now used by default in more than 50K new blogs that appear in WordPress every day.

It has been impossible for us to gather statistics of Akismet error rates, apart from stating that its accuracy is 99,9%, it is used in 12 million sites, and blocks 60 million spam comments per day ⁶, even though some sources report slightly smaller accuracy of 99,46% ⁷.

Although Akismet is broadly used, given its user base in WordPress blogs, there are critics that complain about its rate of false positives ⁸ and research like the one previously mentioned able to bypass it (Ramilli and Prandini, 2009).

Other detection proposals Typically, blog spam contains more or less “hidden” links to URLs to which the spammer wants to redirect the reader, yet not all URL references might be malicious. Some comment spam detection techniques rely on classifying these URLs, for example interpreting the link structure from the posted URL using SVMs, graph metrics and meta-data to detect spam detection (Shin et al., 2015).

Other studies that employ different ML techniques fed with both attributes extracted from the text messages and posting information have shown promising results (Alberto et al., 2015).

Other anti-spam proposals use alternative mechanisms in order to

⁶These statistics were published in an article by TechCrunch in 2011, at <https://techcrunch.com/2012/05/29/automattics-spam-fighter-akismet-just-filtered-its-50-billionth-piece-of-spam/>

⁷WordPress-Force opinion post from 2013, at <http://wpforce.com/huge-increase-spam-last-2-months/>

⁸Criticism with complaints like “Akismet has a reputation for flagging good comments as spam” can be found in blogs and forums. This one in particular is from “Why We Don’t Use Akismet” post at <http://www.web-development-blog.com/archives/why-we-dont-use-akismet/>

improve their detection ratios, i.e. CleanTalk uses fingerprinting techniques like: detection of JavaScript capabilities, IP source address, e-mail address, the content submit time, etc. In the case of CleanTalk, they do not analyse the comment content, so they can also reject any information posted to any web form and not just blog comments. These techniques are similar to the ones discussed in Section A.4.1. It remains to be seen the efficacy of simple measures like these against slightly more advanced attacks or targeted attacks.

A.4 Attack mitigation

Attack Containment, Mitigation and Countermeasure allow us to constrain, mitigate or even nullify the effects of an attack. These measures do not prevent the attack, but its effects in the protected systems can be stopped so no further damage is caused, or minimized so that the system can recover to a state previous to the attack.

A.4.1 Blacklists

Blacklists are lists of attackers. Their identification can be done through different possible mechanisms, as using their IP address, characteristics of the request strings, techniques for client & browser fingerprinting, new HTML5 APIs, etc.

Their detection is typically done through abuse detection, that normally is triggered when a server or servers detect a high number of unusual petitions from the same client.

A well-known example of this is used by Cloudflare, a U.S. company. Its aim is to protect, speeds up, and improve availability for a website or mobile application. They do this by imposing an intermediate server layer thanks to a change in DNS.

These mechanisms have their own drawbacks. For example, if a node in a private network that is behind a proxy is abusing a site, all nodes in that network will lose access to it. Even the traffic from a large proxied network might be taken for an attack, as happened recently for Hong-Kong Google users (Cheng, 2016).

Services that run these blacklists and filtering mechanisms also provide what is known as a *single point of failure*. I.E., the hacker group UGNazi attacked Cloudflare partially via flaws in Google's authentication systems in June 2012, gaining administrative access to Cloudflare and using it to deface 4chan. An October 2015 report found that Cloudflare provisioned 40% of SSL certificates used by phishing sites with deceptive domain names resembling those of banks and payment processors.

Fingerprinting In order to be able to create a black list, it is necessary to first distinguish among different clients and detect who is running an attack. Several techniques can be used for it, as client & browser fingerprinting, source IP detection, cookies and many others, even more so with the new HTML5 APIs, but each one has its limits. Because most of them are created at the client side, with enough motivation or dedication they can be faked.

In some cases, the attack might have a single clear origin: this means that we can trace the attack back to a certain entity. This entity can be a particular *browser* (possibly through the use of cookies, local JavaScript code, local storage (in HTML5), or browser and OS fingerprinting). It can also be an IP address that is generating much more traffic than typical. Or can be a bot that we can differentiate from normal traffic using some specific characteristic unique to it. The important part is that somehow we can differentiate the source/s of the attack. This will not always be possible. For example, it might happen that the IP address of the attacker is part of a network that does *NAT* to allocate internal addresses to several private entities. If we ban this IP, we will be banning the whole network.

If we can somehow figure out the origin of the attack, we can:

- **Origin banning:** ban that origin for a certain amount of time, so as to throttle down the attack. This will pose an inconvenience to legitimate users, but less than banning its source permanently. Note that a banning mechanism can sometimes be used as a DoS mechanism against a legitimate user, so we have to be careful on its implementation. Also, this mechanism might be easy to evade by an skilled attacker.
- **Increase security levels:** impose harder measures to protect the asset being accessed: if the asset has already some protection mechanism that allow for a parametrization of the security level, we can raise this parameter for all the traffic coming from the offender.

We might be able to detect that an attack is going on, but not able to detect its real origin. This can happen if we cannot find a differentiating characteristic of the attack that can always distinguish it, or if it comes from multiple and apparently unrelated sources. In this case we face a more difficult defence, as any extra security measure will affect all the users. Among the different possibilities, one is to require registration. This consists on forcing the users to give and validate an e-mail account. This adds a difficulty level based on how hard it is to obtain *any* valid e-mail account and programatically use it. It does not typically add much security, yet imposes an additional hurdle for valid users.

Blacklists have other inherent limitations. Their power is in the aggregation of information, but that also creates single points of failure. Plus, they allow running DoS attacks against clients or networks, preventing them access to a resource.

A.4.2 Client detection & filtering

We have introduced this idea when we discussed *blacklists* and *fingerprinting* in the sections before. Seen as a single mechanism of detection, it is related to the ability to classify without doubt those clients that are clearly bogus, or attackers.

The most common idea behind this mechanism is that many attackers do not use a regular browser, but some other SW that does not replicate the full functionality of a browser. As an example, many attackers do not run the Java Script code of a web-page, run it partially, or do not have full JS & DOM support.

This is just an arms-race. It can also be totally circumvented using a real browser, either doing that directly (for example, developing a browser plug-in) or through mechanization libraries such as Selenium.

Appendix B

BASECASS template

In this appendix we present an empty BASECASS table for reference. This table can be used as a template by any cybersecurity practitioner when applying BASECASS to a new CAPTCHA.

Table B.1: BASECASS template.

Name: Description:		
Challenge space		
Base problem:	Type:	
	Size:	
CAPTCHA problem:	Domain:	
	Size:	
	Distribution:	

Answer space		
Maximum Range:		
Range:		
Ratio:		
Distribution:		
Challenge space & answer space conclusions		
Is attack possible:		
Description:		
Success:		
Metrics		
Denoising:		
Pre-processing:		
Generic		
Order		
Specific/ Tailored		

Test of metrics		
Is attack possible:		
Description:		
Success:		
Data preparation		
Training set	Size:	
	Balance:	
	Notes:	
Statistical analysis		
Correlations		
Regressions		
ML analysis		
Selection:		
Best algorithms:		
Accuracy:		
κ -statistic :		
S/ML attack & Results		
If previous phase leads to an attack		
Possible?:		
Description:		
Success rate:		
Observations:		

ML vs. parameter analysis	
Optional: if and only if phases before not lead to a successful attack and there is enough data on challenge production parameters	
	For each combination of parameter, value(s), and interesting ML result:
Attack & Results	
If previous phase leads to an attack	
Possible?:	
Description:	
Success rate:	
Observations:	
Conclusion	
Weaknesses:	
Broken?:	
Work-arounds:	

A template can also be found online at <https://github.com/carlos-havier/BASECASS-template>.

Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2016), ‘Tensorflow: Large-scale Machine Learning on heterogeneous distributed systems’, *arXiv preprint* **abs/1603.04467**.
- Abokhodair, N., Yoo, D. and McDonald, D. W. (2016), ‘Dissecting a Social Botnet: Growth, Content and Influence in Twitter’, *ArXiv e-prints* **abs/1604.03627**.
- Ahn, L. V., Blum, M., Hopper, N. J. and Langford, J. (2003), CAPTCHA: using hard AI problems for security, in ‘Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques’, EUROCRYPT’03, Springer-Verlag, Berlin, Heidelberg, pp. 294–311.
- Alberto, T. C., Lochter, J. V. and Almeida, T. A. (2015), ‘Post or block? advances in automatically filtering undesired comments’, *Journal of Intelligent & Robotic Systems* **vol. 80**(1), 245–259.
- Anderson, R. (2002), Security in open versus closed systems - The dance of Boltzmann, Coase and Moore, Technical report, Cambridge University, Cambridge, England.
- Anonymous (2016), ‘Cloudflare recaptcha de-anonymizes tor users’, <https://cryptome.org/2016/07/cloudflare-de-anons-tor.htm>.
- Asghar, M. N., Hussain, F. and Manton, R. (2014), ‘Video indexing: a survey’, *International Journal of Computer and Information Technology* **vol. 03**(01).

- Athanasopoulos, E. and Antonatos, S. (2006), ‘Enhanced CAPTCHAs : using animation to tell humans and computers apart’, *Ifip International Federation For Information Processing* **vol. 4237**, 97–108.
- Atkeson, C., Moore, A. and Schaal, S. (1996), ‘Locally weighted learning’, *Artificial Intelligence Review* **vol. 11**(1), 11–73.
- Back, A. (2002), Hashcash - A denial of service counter-measure, Technical report.
- Baird, H. S. (2006), *Complex Image Recognition and Web Security*, Springer London, London, pp. 287–298.
- Baird, H. S. and Bentley, J. L. (2005), Implicit CAPTCHAs, Vol. 5676, Philadelphia, PA, USA, pp. 191–196.
- Baird, H. S., Coates, A. L. and Fateman, R. J. (2003), ‘PessimPrint: a reverse Turing test’, *International Journal on Document Analysis and Recognition* **vol. 5**(2-3), 158–163.
- Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D. and Bengio, Y. (2010), Theano: A CPU and GPU math compiler in Python, *in* ‘Proceedings of the 9th Python in Science Conference’, Austin, Texas, USA, pp. 1–7.
- Biggio, B., Corona, I., Fumera, G., Giacinto, G. and Roli, F. (2011), Bagging classifiers for fighting poisoning attacks in adversarial classification tasks, *in* ‘International Workshop on Multiple Classifier Systems’, Springer, Naples, Italy, pp. 350–359.
- Biggio, B., Nelson, B. and Laskov, P. (2012), ‘Poisoning attacks against support vector machines’, *arXiv preprint* **abs/1206.6389**.
- Bigham, J. P. and Cavender, A. C. (2009), Evaluating existing audio CAPTCHAs and an interface pttimized for non-visual users, *in* ‘Proceedings of the SIGCHI Conference on Human Factors in Computing Systems’, CHI ’09, ACM, New York, NY, USA, pp. 1829–1838.
- Bilton, N. (2014), ‘Social Media Bots Offer Phony Friends and Real Profit’. Accessed on 2017-08-16.
URL: <https://www.nytimes.com/2014/11/20/fashion/social-media-bots-offer-phony-friends-and-real-profit.html>
- Bird, S., Klein, E. and Loper, E. (2009), *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O’Reilly, Beijing.

- Birrell, A., Burrows, M., Dwork, C., Manasse, M. and Wobber, T. (2004), The Penny Black Project, Technical report, Microsoft Research.
- Bishop, M., Cummins, J., Peisert, S., Singh, A., Bhumiratana, B., Agarwal, D., Frincke, D. and Hogarth, M. (2010), Relationships and data sanitization: a study in scarlet, *in* 'Proceedings of the 2010 workshop on new security paradigms', ACM, Concord, MA, USA, pp. 151–164.
- Blei, D. M., Ng, A. Y. and Jordan, M. I. (2003), 'Latent dirichlet allocation', *Journal of machine Learning research* **vol. 3**(Jan), 993–1022.
- Boshmaf, Y., Muslukhov, I., Beznosov, K. and Ripeanu, M. (2013), 'Design and analysis of a social botnet', *Computer Networks: The International Journal of Computer and Telecommunications Networking* **vol. 57**(2), 556–578.
- Breiman, L. (2001), 'Random forests', *Machine Learning* **vol. 45**(1), 5–32.
- Bright, P. (2011), 'A way to take out spammers? 3 banks process 95% of spam transactions'. Accessed on 2017-08-16.
URL: <http://arstechnica.com/tech-policy/2011/05/a-way-to-take-out-spammers-3-banks-process-95-of-spam-transactions/>
- Bursztein, E. (2012), How we broke the NuCaptcha video scheme and what we propose to fix it, Technical report, Google Anti-abuse Research Team.
- Bursztein, E., Aigrain, J., Moscicki, A. and Mitchell, J. C. (2014), The end is nigh: generic solving of text-based CAPTCHAs, *in* '8th USENIX Workshop on Offensive Technologies (WOOT 14)', San Diego, CA, USA.
- Bursztein, E., Martin, M. and Mitchell, J. (2011), Text-based CAPTCHA Strengths and Weaknesses, *in* 'Proceedings of the 18th ACM Conference on Computer and Communications Security', CCS '11, ACM, New York, NY, USA, pp. 125–138.
- Cattell, R. B. (1952), *Factor analysis: an introduction and manual for the psychologist and social scientist*, Harper, New York, USA.
- Chan, P. P., Yang, C., Yeung, D. S. and Ng, W. W. (2015), 'Spam filtering for short messages in adversarial environment', *Neurocomputing* **vol. 155**, 167–176.
- Chan, P. P., Zhang, F., Ng, W. W., Yeung, D. S. and Jiang, J. (2011), A novel defend against good word attacks, *in* '2011 International Conference

- on Machine Learning and Cybernetics (ICMLC)', Vol. vol. 3, IEEE, Guilin, China, pp. 1088–1092.
- Chellapilla, K., Larson, K., Simard, P. and Czerwinski, M. (2005*a*), Computers beat humans at single character recognition in reading based human interaction proofs (hips), *in* 'Proceedings of the 2nd Conference on Email and Anti-Spam', Palo Alto, CA, USA.
- Chellapilla, K., Larson, K., Simard, P. Y. and Czerwinski, M. (2005*b*), Building segmentation based human-friendly human interaction proofs (hips), *in* 'Second International Workshop on Human Interactive Proofs (HIP 2005)', Springer, Bethlehem, PA, USA, pp. 1–26.
- Chellapilla, K. and Simard, P. Y. (2005), Using Machine Learning to Break Visual Human Interaction Proofs (HIPs), *in* 'Advances in Neural Information Processing Systems', Vancouver, Canada, pp. 265–272.
- Cheng, K. (2016), 'CAPTCHA search issue affecting Hongkongers has been resolved, says Google'. Accessed on 2017-08-16.
URL: <https://www.hongkongfp.com/2016/11/11/google-tackles-captcha-search-issue-affecting-hongkongers/>
- Chew, M. and Baird, H. S. (2003), Baffletext: a human interactive proof, *in* '10th IS&T/SPIE Document Recognition & Retrieval Conference', SPIE, San Jose, CA, USA, pp. 305–316.
- Chew, M. and Tygar, J. D. (2004), Image recognition captchas, *in* '7th International Conference on Information Security', ISC, Springer, Palo Alto, CA, USA, pp. 268–279.
- Chew, M. and Tygar, J. D. (2005), Collaborative filtering captchas, *in* 'Second International Workshop on Human Interactive Proofs', Springer, Bethlehem, PA, USA, pp. 66–81.
- Chollet, F. (2015), 'Keras: Deep learning library for theano and tensorflow'. Accessed on 2017-08-16.
URL: <https://keras.io/>
- Ciregan, D., Meier, U. and Schmidhuber, J. (2012), Multi-column deep neural networks for image classification, *in* '2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', IEEE, Providence, RI, USA, pp. 3642–3649.

- Cleary, J. G. and Trigg, L. E. (1995), K*: An instance-based learner using an entropic distance measure, *in* '12th International Conference on Machine Learning', Morgan Kaufmann, Tahoe City, California, USA, pp. 108–114.
- Cluley, G. (2007), 'Remember melissa the malware stripper? she's back', *Naked Security by Sophos* **2007**(11).
- Cohen, J., Cohen, P., West, S. G. and Aiken, L. S. (2013), *Applied multiple regression/correlation analysis for the behavioral sciences*, Routledge.
- Cohen, W. W. (1995), Fast effective rule induction, *in* 'Twelfth International Conference on Machine Learning', Morgan Kaufmann, Tahoe City, California, USA, pp. 115–123.
- Conti, M., Guarisco, C. and Spolaor, R. (2016), *CAPTCHaStar! A Novel CAPTCHA Based on Interactive Shape Discovery*, Springer International Publishing, Guildford, UK, pp. 611–628.
- Converse, T. (2005), Captcha generation as a web service, *in* 'Proceedings of the Second International Conference on Human Interactive Proofs', HIP'05, Springer-Verlag, Bethlehem, PA, USA, pp. 82–96.
- Crouzet, S. M., Kirchner, H. and Thorpe, S. J. (2010), 'Fast saccades toward faces: Face detection in just 100 ms', *Journal of Vision* **vol. 10**(4), 16.
- Cui, J.-S., Mei, J.-T., Zhang, W.-Z., Wang, X. and Zhang, D. (2010), A captcha implementation based on moving objects recognition problem, *in* '2010 International Conference on E-Business and E-Government (ICEE)', IEEE, Guangzhou, China, pp. 1277–1280.
- Danchev, D. (2008), 'Inside india's captcha solving economy', <http://www.zdnet.com/article/inside-indias-captcha-solving-economy/>.
- Dang-Nguyen, D.-T., Pasquini, C., Conotter, V. and Boato, G. (2015), Raise: A raw images dataset for digital image forensics, *in* 'Proceedings of the 6th ACM Multimedia Systems Conference', MMSys '15, ACM, Portland, Oregon, pp. 219–224.
- Datta, R., Li, J. and Wang, J. Z. (2005), Imagination: a robust image-based captcha generation system, *in* 'MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia', ACM, New York, NY, USA, pp. 331–334.

- De Marsico, M., Marchionni, L., Novelli, A. and Oertel, M. (2016), ‘FATCHA: biometrics lends tools for CAPTCHAs’, *Multimedia Tools and Applications* **vol. 76**(4), 5117–5140.
- Delaunay, B. (1934), ‘Sur la sphere vide’, *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* **vol. 7**(793-800), 1–2.
- Demiroz, G. and Guvenir, A. (1997), Classification by voting feature intervals, *in* ‘9th European Conference on Machine Learning’, Springer, Prague, Czech Republic, pp. 85–92.
- Demšar, J., Curk, T., Erjavec, A., Črt Gorup, Hočevár, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M. and Zupan, B. (2013), ‘Orange: Data mining toolbox in python’, *Journal of Machine Learning Research* **14**, 2349–2353.
- Deng, Y. and Manjunath, B. (2001), ‘Unsupervised segmentation of color-texture regions in images and video’, *IEEE transactions on pattern analysis and machine intelligence* **vol. 23**(8), 800–810.
- Dwork, C., Goldberg, A. and Naor, M. (2003), On memory-bound functions for fighting spam, *in* ‘23rd Annual International Cryptology Conference’, Springer, Santa Barbara, California, USA, pp. 426–444.
- Dwork, C. and Naor, M. (1992), Pricing via processing or combatting junk mail, *in* ‘12th Annual International Cryptology Conference’, Springer, Santa Barbara, California, USA, pp. 139–147.
- Dwork, C., Naor, M. and Wee, H. (2005), Pebbling and proofs of work, *in* ‘25th Annual International Cryptology Conference’, Springer, Santa Barbara, California, USA, pp. 37–54.
- Echeverría, J. and Zhou, S. (2017), ‘The ‘Star Wars’ botnet with over 350k Twitter bots’, *ArXiv e-prints* **abs/1701.02405**.
- El Ahmad, A. S., Yan, J. and Marshall, L. (2010), The robustness of a new captcha, *in* ‘Proceedings of the Third European Workshop on System Security’, EUROSEC ’10, ACM, Paris, France, pp. 36–41.
- Elson, J., Douceur, J. R., Howell, J. and Saul, J. (2007), Asirra: a captcha that exploits interest-aligned manual image categorization, *in* ‘CCS ’07: Proceedings of the 14th ACM conference on Computer and Communications Security’, New York, NY, USA, pp. 366–374.

- Facebook (2011), ‘How does facebook suggest tags?’, https://www.facebook.com/help/122175507864081?helpref=uf_permalink.
- Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R. and Lin, C.-J. (2008), ‘Liblinear: A library for large linear classification’, *Journal of Machine Learning Research* **vol. 9**, 1871–1874.
- Fenton, S. (2015), ‘TripAdvisor denies rating system is flawed, after fake restaurant tops rankings in Italy’, *The Independent* **Jun**.
- Ferrara, E., Varol, O., Davis, C., Menczer, F. and Flammini, A. (2014), ‘The Rise of Social Bots’, *ArXiv e-prints* **abs/1407.5225**.
- Fischer, I. and Herfet, T. (2006), Visual CAPTCHAs for document authentication, in ‘2006 IEEE Workshop on Multimedia Signal Processing’, IEEE, Victoria, BC, Canada, pp. 471–474.
- Friedman, J., Hastie, T. and Tibshirani, R. (1998), Additive logistic regression: a statistical view of boosting, Technical report, Stanford University, Stanford University.
- Fritsch, C., Netter, M., Reisser, A. and Pernul, G. (2010), Attacking image recognition captchas, in ‘International Conference on Trust, Privacy and Security in Digital Business’, Springer, Bilbao, Spain, pp. 13–25.
- FusionQuest (2009), ‘FusionQuest, Inc. Captcha2’, <http://www.captcha2.com>.
- Gao, H., Lei, L., Zhou, X., Li, J. and Liu, X. (2015), The Robustness of Face-Based CAPTCHAs, in ‘2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing’, Liverpool, UK, pp. 2248–2255.
- Gao, H., Yan, J., Cao, F., Zhang, Z., Lei, L., Tang, M., Zhang, P., Zhou, X., Wang, X. and Li, J. (2016), ‘A Simple Generic Attack on Text Captchas’, *Network and Distributed System Security Symposium (NDSS)* **1**(February), 21–24.
- Gigoit (2006), ‘Humanauth’, <https://sourceforge.net/projects/humanauth/>.
- Golle, P. (2009), Machine learning attacks against the asirra captcha, in ‘Proceedings of the 5th Symposium on Usable Privacy and Security, SOUPS

- 2009', ACM International Conference Proceeding Series, ACM, Mountain View, California, USA.
- Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S. and Shet, V. D. (2013), 'Multi-digit number recognition from street view imagery using deep convolutional neural networks', *arXiv preprint* **abs/1312.6082**.
- Goodfellow, I. J., Shlens, J. and Szegedy, C. (2014), 'Explaining and harnessing adversarial examples', *arXiv preprint* **abs/1412.6572**.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014), Generative Adversarial Nets, *in* Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence and K. Q. Weinberger, eds, 'Neural Information Processing Systems 2014', Curran Associates, Inc., Montreal, Canada, pp. 2672–2680.
- Gosschalk, K. and Ford, M. (2016), 'FunCAPTCHA', <https://www.funcaptcha.com/how-to-solve-funcaptcha/>.
- Gossweiler, R., Kamvar, M. and Baluja, S. (2009), What's up captcha?: A captcha based on image orientation, *in* 'Proceedings of the 18th International Conference on World Wide Web', WWW '09, ACM, Madrid, Spain, pp. 841–850.
- Goswami, G., Powell, B. M., Vatsa, M., Singh, R. and Noore, A. (2014a), 'FaceDCAPTCHA: Face detection based color image CAPTCHA', *Future Generation Computer Systems* **vol. 31**, 59–68.
- Goswami, G., Powell, B. M., Vatsa, M., Singh, R. and Noore, A. (2014b), 'FR-CAPTCHA: CAPTCHA Based on Recognizing Human Faces', *PloS one* **vol. 9**(4), e91708.
- Greco, S., Matarazzo, B. and Slowinski, R. (2001), 'Rough sets theory for multicriteria decision analysis', *European journal of operational research* **vol. 129**(1), 1–47.
- Greenblatt, M. and Lagares-Greenblatt, H. (2012), 'Webcam captcha'.
- Gross, J. (2015), 'Motion, orientation, and touch-based CAPTCHAs'.
- Group, C. (2016), 'HelloCAPTCHA vs Spambots', <http://www.hellocaptcha.com>.

- Guerara, M., Merlob, A. and Migliardi, M. (2017), ‘Completely Automated Public Physical test to tell Computers and Humans Apart: A usability study on mobile devices’, *Future Generation Computer Systems* **03/2017**.
- Gupta, S. (2015), ‘Article: Gender detection using machine learning techniques and delaunay triangulation’, *International Journal of Computer Applications* **vol. 124**(6), 27–32.
- Hall, M. and Frank, E. (2008), Combining naive bayes and decision tables, in ‘Proceedings of the 21st Florida Artificial Intelligence Society Conference (FLAIRS)’, AAAI press, Marco Island, Florida, US, pp. 318–319.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. (2009), ‘The weka data mining software: an update’, *ACM SIGKDD Explorations Newsletter* **vol. 11**.
- Halprin, R. (2007), Dependent captchas: Preventing the relay attack, Technical report, Computing and Information Systems.
- Hankins, P. (2004), ‘Minski’, <http://www.consciousentities.com/minsky.htm>.
- Hannun, A. Y., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A. and Ng, A. Y. (2014), ‘Deep speech: Scaling up end-to-end speech recognition’, *arXiv preprint abs/1412.5567*.
- Hartley, A. (2009), ‘WoW “gold farming” banned in China’, *Techradar* **Jun**.
- Hernandez-Castro, C. J., Hernandez-Castro, J. C., Stainton-Ellis, J. D. and Ribagorda, A. (2010), Shortcomings in captcha design and implementation: Captcha2, a commercial proposal, in ‘Eight International Network Conference (INC 2010)’, Heidelberg, Germany.
- Hernández-Castro, C. J., R-moreno, M. D. and Barrero, D. F. (2014), Side-channel attack against the Cappy HIP, in ‘Fifth International Conference on Emerging Security Technologies (EST 2014)’, IEEE, Alcalá de Henares, Spain, pp. 99–104.
- Hernández-Castro, C. J., R-Moreno, M. D. and Barrero, D. F. (2015), ‘Using JPEG to Measure Image Continuity and Break Cappy and Other Puzzle CAPTCHAs’, *IEEE Internet Computing* **vol. 19**(6), 46–53.

- Hernández-Castro, C. J., R-Moreno, M. D., Barrero, D. F. and Li, S. (2017), ‘An oracle-based attack on CAPTCHAs protected against oracle attacks’, *ArXiv e-prints* **abs/1702.03815**.
- Hernandez-Castro, C. J. and Ribagorda, A. (2009a), Remotely telling humans and computers apart: an unsolved problem, *in* ‘iNetSec 2009 - Open Research Problems in Network Security - IFIP WG 11.4’, Zurich, Switzerland.
- Hernandez-Castro, C. J. and Ribagorda, A. (2009b), Video captchas, *in* ‘IDET Security Conference - Security and Protection of Information (SPIE)’, Brno, Czech Republic.
- Hernandez-Castro, C. J. and Ribagorda, A. (2010), ‘Pitfalls in captcha design and implementation: the math captcha, a case study’, *Computers & Security* **vol. 29**(1), 141–157.
URL: <http://dx.doi.org/10.1016/j.cose.2009.06.006>
- Hernandez-Castro, C. J., Ribagorda, A. and Hernandez-Castro, J. C. (2011), On the strength of egglue and other logic CAPTCHAs, *in* ‘International Conference on Security and Cryptography (Secrypt 2011)’, Seville, Spain, pp. 157–167.
- Hernandez-Castro, C. J., Ribagorda, A. and Saez, Y. (2010), Side-channel attack on the humanauth captcha, *in* ‘International Conference on Security and Cryptography (Secrypt 2010)’, Athens, Greece.
- Hindle, A., Godfrey, M. W. and Holt, R. C. (2008), Reverse engineering captchas, *in* ‘2008 15th Working Conference on Reverse Engineering’, Antwerp, Belgium.
- Hoepman, J.-H. and Jacobs, B. (2007), ‘Increased security through open source’, *Communications of the ACM* **vol. 50**(1), 79–83.
- Hogan, P. (2016), ‘How ticket-scalping bots steal all those “hamilton” seats you desperately wanted’.
URL: <http://splinternews.com/how-ticket-scalping-bots-steal-all-those-hamilton-seats-1793861218>
- Holmes, G., Pfahringer, B., Kirkby, R., Frank, E. and Hall, M. (2002), Multiclass alternating decision trees, *in* ‘European Conference on Machine Learning (Joint European Conference on Machine Learning and Knowledge Discovery in Databases)’, Springer, Helsinki, Finland, pp. 161–172.

- Homakov, E. (2013a), ‘How we hacked Facebook with OAuth2 and Chrome bugs’, <http://homakov.blogspot.com.es/2013/02/hacking-facebook-with-oauth2-and-chrome.html>. Accessed on 2017-08-16.
URL: <http://homakov.blogspot.com.es/2013/02/hacking-facebook-with-oauth2-and-chrome.html>
- Homakov, E. (2013b), ‘OAuth1, OAuth2, OAuth...?’, <http://homakov.blogspot.com.es/2013/03/oauth1-oauth2-oauth.html>. Accessed on 2017-08-16.
URL: <http://homakov.blogspot.com.es/2013/03/oauth1-oauth2-oauth.html>
- Homakov, E. (2014), ‘The No CAPTCHA problem’. Accessed on 2017-08-16.
URL: <http://homakov.blogspot.com.es/2014/12/the-no-captcha-problem.html>
- Hu, W., Xie, N., Li, L., Zeng, X. and Maybank, S. (2011), ‘A survey on visual content-based video indexing and retrieval’, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **vol. 41**(6), 797–819.
- Huggins, J. and Hammant, P. (2014), ‘Selenium, browser automation framework’. Accessed on 2017-08-16.
URL: <http://code.google.com/p/selenium>
- Hupperich, T., Krombholz, K. and Holz, T. (2016), *Sensor Captchas: On the Usability of Instrumenting Hardware Sensors to Prove Liveliness*, Springer International Publishing, Cham, pp. 40–59.
- Inc., A. (2016), ‘Mollom CAPTCHA’.
URL: <https://www.mollom.com/how-mollom-works>
- Ipeirotis, P., Tamir, D. and Kanth, P. (2010), ‘Mechanical Turk: Now with 40.92% spam’. Accessed on 2017-08-16.
URL: <http://www.behind-the-enemy-lines.com/2010/12/mechanical-turk-now-with-4092-spam.html>
- Jia, R. and Liang, P. (2017), ‘Adversarial Examples for Evaluating Reading Comprehension Systems’, *ArXiv e-prints* **abs/1707.07328**.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. (2014), Caffe: Convolutional architecture for fast feature embedding, in ‘Proceedings of the 22nd ACM international conference on Multimedia’, ACM, Orlando, Florida, USA, pp. 675–678.

- Jiang, N. and Dogan, H. (2015), A gesture-based captcha design supporting mobile devices, *in* 'Proceedings of the 2015 British HCI Conference', British HCI '15, ACM, Lincoln, Lincolnshire, United Kingdom, pp. 202–207.
- Jiang, N. and Tian, F. (2013), A novel gesture-based captcha design for smart devices, *in* 'Proceedings of the 27th International BCS Human Computer Interaction Conference', BCS-HCI '13, British Computer Society, London, UK, pp. 49:1–49:5.
- Kang, L. and Xiang, J. (2010), Captcha phishing: A practical attack on human interaction proofing, *in* 'Information Security and Cryptology: 5th International Conference, Inscrypt 2009. Revised Selected Papers', Springer Berlin Heidelberg, Beijing, China, pp. 411–425.
- Katz, P. (1996), 'DEFLATE Compressed Data Format Specification version 1.3', RFC 1951 (Informational).
- Kerckhoffs, A. (1883), 'La cryptographie militaire', *Journal des Sciences Militaires* **vol. IX**(Janvier), 5–38.
- Khryashchev, V., Priorov, A., Shmaglit, L. and Golubev, M. (2012), Gender recognition via face area analysis, *in* 'World congress on engineering and computer science', San Francisco, USA, pp. 645–649.
- Kim, C. and Hwang, J.-N. (2002), 'Fast and automatic video object segmentation and tracking for content-based applications', *IEEE transactions on circuits and systems for video technology* **vol. 12**(2), 122–129.
- Kim, H., Tang, J. and Anderson, R. (2012), Social authentication: harder than it looks, *in* 'International Conference on Financial Cryptography and Data Security', Springer, Bonaire, Netherlands, pp. 1–15.
- Kim, J., Kim, S., Yang, J., Ryu, J.-H. and Wohn, K. (2014), 'Facecaptcha: A captcha that identifies the gender of face images unrecognized by existing gender classifiers', *Multimedia Tools and Applications* **vol. 72**(2), 1215–1237.
- Kim, J.-W., Chung, W.-K. and Cho, H.-G. (2010), 'A new image-based captcha using the orientation of the polygonally cropped sub-images', *The Visual Computer* **vol. 26**(6), 1135–1143.
- Kluever, K. A. (2008), Evaluating the Usability and Security of a Video CAPTCHA, Master's thesis, Rochester Institute of Technology.

- Kohavi, R. (1996), Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid, *in* ‘Second International Conference on Knowledge Discovery and Data Mining’, Association for the Advancement of Artificial Intelligence, Portland, Oregon, USA, pp. 202–207.
- Kolupaev, A. and Ogijenko, J. (2013), ‘Teabag 3D CAPTCHA v1.0.1’. Accessed on 2011-02-25.
URL: <http://ocr-research.org.ua/>
- Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, eds, ‘Advances in Neural Information Processing Systems 25 (NIPS 2012)’, Curran Associates, Inc., Lake Tahoe, USA, pp. 1097–1105.
- Kund, I. (2011), Non-Standard CAPTCHAS for the Web: A Motion Based Character Recognition HIP, Master’s thesis, University of Manchester.
- Kwon, S. and Cha, S. (2016), ‘A Paradigm Shift for the CAPTCHA Race: Adding Uncertainty to the Process’, *IEEE Software* **33**(6), 80–85.
- Landwehr, N., Hall, M. and Frank, E. (2005), ‘Logistic model trees’, *Machine Learning* **vol. 95**(1-2), 161–205.
- Larsen, A. B. L., Sønderby, S. K. and Winther, O. (2015), ‘Autoencoding beyond pixels using a learned similarity metric’, *arXiv preprint abs/1512.09300*.
- Laurie, B. and Clayton, R. (2004), Proof-of-Work proves not to work, *in* ‘IN WEAS 04’.
- Le, T., Dua, A. and Feng, W.-c. (2012), kapow plugins: Protecting web applications using reputation-based proof-of-work, *in* ‘Proceedings of the 2Nd Joint WICOW/AIRWeb Workshop on Web Quality’, WebQuality ’12, ACM, Lyon, France, pp. 60–63.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989), ‘Backpropagation applied to handwritten zip code recognition’, *Neural computation* **vol. 1**(4), 541–551.
- Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998), ‘Gradient-based learning applied to document recognition’, *Proceedings of the IEEE* **vol. 86**(11), 2278–2324.

- Leung, T. K., Burl, M. C. and Perona, P. (1998), Probabilistic affine invariants for recognition, *in* 'Proceedings of the 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE, Santa Barbara, California, USA, pp. 678–684.
- Levchenko, K., Pitsillidis, A., Chachra, N., Enright, B., Félegyházi, M., Grier, C., Halvorson, T., Kanich, C., Kreibich, C., Liu, H. and McCoy, D. (2011), Click trajectories: End-to-end analysis of the spam value chain, *in* '2011 IEEE Symposium on Security and Privacy', IEEE, Oakland, California, USA, pp. 431–446.
- Levi, G. and Hassner, T. (2015), Age and gender classification using convolutional neural networks, *in* 'IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)', IEEE, Boston, MA, USA.
- Lichterman, J. (2017), 'Norwegian news site readers pass a quiz before commenting'. Accessed on 2017-08-14.
URL: <http://www.niemanlab.org/2017/03/this-site-is-taking-the-edge-off-rant-mode-by-making-readers-pass-a-quiz-before-commenting/>
- Lillibridge, M., Abadi, M., Bharat, K. and Broder, A. (2001), 'Method for selectively restricting access to computer systems'.
- Liu, T., Yuan, Z., Sun, J., Wang, J., Zheng, N., Tang, X. and Shum, H.-Y. (2011), 'Learning to detect a salient object', *IEEE Transactions on Pattern analysis and machine intelligence* **vol.** **33**(2), 353–367.
- Lodderstedt, T., McGloin, M. and Hunt, P. (2013), 'OAuth 2.0 Threat Model and Security Considerations', RFC 6819 (Informational).
- Longe, O. B. (2010), 'Mitigating CAPTCHA relay attacks using multiple challenge-response mechanism', *Computing and Information Systems* **vol.** **14**(3), 36–42.
- Lowd, D. and Meek, C. (2005), Good Word Attacks on Statistical Spam Filters, *in* 'Proceedings of the Second Conference on Email and Anti-Spam (CEAS)', Stanford University, California, USA, pp. 161–172.
- Marshall, J. and Lin, G. (2006), 'HotCaptcha'. Accessed on 2006-09-01.
URL: <http://hotcaptcha.com/>
- Martin, B. (1995), Instance-based learning: Nearest neighbor with generalization, Master's thesis, University of Waikato, Hamilton, New Zealand.

- Martin, C. (2008), ‘Rapidshare CAPTCHA with Cats Cracked by Crypt-Load’.
URL: <http://www.aboutonlinetips.com/rapidshare-captcha-with-cats-cracked-by-cryptload/>
- McInerny, M., Brighton, M., Demirjian, S. and Hotchkies, B. (2017), ‘Turing test via reaction to test modifications’.
- Mehrnejad, M., Bafghi, A. G., Harati, A. and Toreini, E. (2011), Multiple SEIMCHA: Multiple semantic image CAPTCHA, *in* ‘2011 International Conference for Internet Technology and Secured Transactions’, IEEE, Abu Dhabi, United Arab Emirates, pp. 196–201.
- Mitra, N. J., Chu, H.-K., Lee, T.-Y., Wolf, L., Yeshurun, H. and Cohen-Or, D. (2009a), ‘Emerging images’, *ACM Trans. Graph.* **vol. 28**(5), 163:1–163:8.
- Mitra, N. J., Chu, H.-K., Lee, T.-Y., Wolf, L., Yeshurun, H. and Cohen-Or, D. (2009b), ‘Emerging images’, pp. 163:1–163:8.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015), ‘Human-level control through deep reinforcement learning’, *Nature* **518**(7540), 529–533.
- Mohamed, M., Gao, S., Saxena, N. and Zhang, C. (2014), Dynamic cognitive game CAPTCHA usability and detection of streaming-based farming, *in* ‘Usable Security (USEC 2014)’, Internet Society, San Diego, CA, USA.
- Mohamed, M., Sachdeva, N., Georgescu, M., Gao, S., Saxena, N., Zhang, C., Kumaraguru, P., van Oorschot, P. C. and bang Chen, W. (2013), ‘Three-way dissection of a game-captcha: Automated attacks, relay attacks, and usability’, *arXiv preprint* **abs/1310.1540**.
- Mori, G. and Malik, J. (2003), Recognizing objects in adversarial clutter: breaking a visual captcha, *in* ‘Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition’, Vol. vol.1, IEEE, Madison, Wisconsin, USA, pp. I–134–I–141.
- Naor, M. (1996), ‘Verification of a human in the loop or Identification via the Turing Test’, <http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human.ps>.

- Naumann, A. B., Franke, T. and Bauckhage, C. (2009), Investigating CAPTCHAs Based on Visual Phenomena, *in* 'IFIP Conference on Human-Computer Interaction', Springer, Uppsala, Sweden, pp. 745–748.
- Ng, C.-B., Tay, Y.-H. and Goi, B.-M. (2013), *A Convolutional Neural Network for Pedestrian Gender Recognition*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 558–564.
- Nguyen, D. V. (2014), Contributions to Text-based CAPTCHA Security, PhD thesis, University of Wollongong.
- Nguyen, V. D., Chow, Y.-W. and Susilo, W. (2011), Breaking a 3d-based captcha scheme, *in* 'Proceedings of the 14th International Conference on Information Security and Cryptology', ICISC'11, Springer-Verlag, Seoul, Korea, pp. 391–405.
URL: http://dx.doi.org/10.1007/978-3-642-31912-9_26
- Nguyen, V. D., Chow, Y.-W. and Susilo, W. (2012a), *Attacking Animated CAPTCHAs via Character Extraction*, Springer Berlin Heidelberg, Darmstadt, Germany, pp. 98–113.
URL: http://dx.doi.org/10.1007/978-3-642-35404-5_9
- Nguyen, V. D., Chow, Y.-W. and Susilo, W. (2012b), Breaking an animated CAPTCHA scheme, *in* 'International Conference on Applied Cryptography and Network Security', Springer, Singapore, Singapore, pp. 12–29.
- Nguyen, V. D., Chow, Y.-W. and Susilo, W. (2014a), A CAPTCHA scheme based on the identification of character locations, *in* 'International Conference on Information Security Practice and Experience', Springer, Fuzhou, China, pp. 60–74.
- Nguyen, V. D., Chow, Y.-W. and Susilo, W. (2014b), 'On the security of text-based 3D CAPTCHAs', *Computers & Security* **vol. 45**, 84–99.
- Nielsen, F. Å. (2011), 'A new anew: Evaluation of a word list for sentiment analysis in microblogs', *CoRR* **abs/1103.2903**.
- NuCaptcha (2016), 'NuCaptcha Security Features'. Accessed on 2014-11-20.
URL: <http://www.nucaptcha.com/security-features>
- Onwudebelu, U. and Ugwuoke, U. (2012), 'Employing response time constraints to mitigate CAPTCHA relay attacks', *African Journal of Computing & ICT* **vol. 5**(2), 11–16.

- Osadchy, M., Hernandez-Castro, J., Hernandez, J., Gibson, S., Dunkelman, O. and Pérez-Cabo, D. (2016), ‘No Bot Expects the DeepCAPTCHA! Introducing Immutable Adversarial Examples, With Applications to CAPTCHA Generation’, *IEEE Transactions on Information Forensics and Security* **vol. 12**(11), 2640 – 2653.
- Pantel, P. and Lin, D. (1998), Spamcop: A spam classification & organization program, *in* ‘Proceedings of AAAI-98 Workshop on Learning for Text Categorization’, AAAI Press, Madison, Wisconsin, USA, pp. 95–98.
- Papert, S. A. (1966), ‘The Summer Vision Project’. Accessed on 2014-14-02.
URL: <https://dspace.mit.edu/handle/1721.1/6125>
- Parsons, J. (2015), ‘Facebook’s War Continues Against Fake Profiles and Bots’. Accessed on 2014-14-02.
URL: http://www.huffingtonpost.com/james-parsons/facebooks-war-continues-against-fake-profiles-and-bots_b_6914282.html
- Paxton, T. and Tatoris, R. (2012), ‘How PlayThru makes CAPTCHA obsolete’. Accessed on 2012-09-26.
URL: <http://areyouahuman.com/benefits/>
- Pearson, K. (1901), ‘On lines and planes of closest fit to systems of points in space’, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **vol. 2**(11), 559–572.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011), ‘Scikit-learn: Machine Learning in Python’, *Journal of Machine Learning Research* **12**, 2825–2830.
- Plutchik, R. (1991), *The emotions*, University Press of America.
- Polakis, I., Lancini, M., Kontaxis, G., Maggi, F., Ioannidis, S., Keromytis, A. D. and Zanero, S. (2012), All your face are belong to us: breaking Facebook’s social authentication, *in* ‘Proceedings of the 28th Annual Computer Security Applications Conference’, ACM, Orlando, Florida, USA, pp. 399–408.
- Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

- Qvarfordt, P., Rieffel, E. G. and Hilbert, D. M. (2013), ‘Motion and interaction based captcha’.
- Ramilli, M. and Prandini, M. (2009), ‘Comment spam injection made easy’, *6th IEEE Consumer Communications and Networking Conference, CCNC 2009* pp. 1–5.
- Rao, J. M. and Reiley, D. H. (2012), ‘The economics of spam’, *Journal of Economic Perspectives* **26**(3), 87–110.
- Robinson, S. (2001), ‘Can Hard AI Problems Foil Internet Interlopers?’. Accessed on 2014-14-02.
- Ross, S. A., Halderman, J. A. and Finkelstein, A. (2010), Sketcha: a CAPTCHA based on Line Drawings of 3D Models, *in* ‘Proceedings of the 19th international conference on World wide web’, ACM, Raleigh, NC, USA, pp. 821–830.
- Sahami, M., Dumais, S., Heckerman, D. and Horvitz, E. (1998), A Bayesian approach to filtering junk e-mail, *in* ‘Learning for Text Categorization: Papers from the 1998 workshop (ICML/AAAI-98)’, Vol. 62, Madison, Wisconsin, USA, pp. 98–105.
- Sano, S., Otsuka, T. and Okuno, H. G. (2013), *Solving Google’s Continuous Audio CAPTCHA with HMM-Based Automatic Speech Recognition*, Springer Berlin Heidelberg, Okinawa, Japan, pp. 36–52.
- Santamarta, R. (2008), ‘Breaking gmail’s audio captcha’, <http://blog.wintercore.com/?p=11>. Accessed on 2010-13-02.
URL: <http://blog.wintercore.com/?p=11>
- Scarfone, K., Jansen, W. and Tracy, M. (2008), ‘Guide to General Server Security’. Accessed on 2017-14-08.
- Schmidt, C. (2017), ‘Remember that norwegian site that made readers take a quiz before commenting? here’s an update on it’. Accessed on 2017-08-14.
URL: <http://www.niemanlab.org/2017/08/remember-that-norwegian-site-that-makes-readers-take-a-quiz-before-commenting-heres-an-update-on-it/>
- Schryen, G., Wagner, G. and Schlegel, A. (2016), ‘Development of two novel face-recognition captchas’, *Comput. Secur.* **vol. 60**(C), 95–116.
- Seber, G. A. F. (1974), *The estimation of animal abundance*, Vol. vol. 16, Griffin London.

- Security, G. P. . O. (2005), ‘Spam Spammers... Here’s How To Succeed Without Retaliation’.
- SEO, D. (2008a), ‘Letter derotation’, <http://www.darkseoprogramming.com/2008/04/05/letter-derotation/>.
URL: <http://www.darkseoprogramming.com/2008/04/05/letter-derotation/>
- SEO, D. (2008b), ‘Phpbb3 captcha is super easy’, <http://www.darkseoprogramming.com/2008/05/12/phpbb3-captcha-is-super-easy/>.
URL: <http://www.darkseoprogramming.com/2008/05/12/phpbb3-captcha-is-super-easy/>
- Shao, C., Ciampaglia, G. L., Varol, O., Flammini, A. and Menczer, F. (2017), ‘The spread of fake news by social bots’, *ArXiv e-prints* **abs/1707.07592**.
- Sheffer, Y. (2015), ‘Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)’, RFC 7457 (Informational).
- Shen, D., Wong, W.-h. and Ip, H. H. (1999), ‘Affine-invariant image retrieval by correspondence matching of shapes’, *Image and Vision Computing* **vol. 17**(7), 489–499.
- Shet, V. (2014a), ‘Are you a robot? Introducing No CAPTCHA reCAPTCHA’, <https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html>. Accessed on 2017-08-14.
URL: <https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html>
- Shet, V. (2014b), ‘Street View and reCAPTCHA technology just got smarter’, <https://security.googleblog.com/2014/04/street-view-and-recaptcha-technology.html>. Accessed on 2017-08-14.
URL: <https://security.googleblog.com/2014/04/street-view-and-recaptcha-technology.html>
- Shin, Y., Myers, S., Gupta, M. and Radivojac, P. (2015), ‘A link graph-based approach to identify forum spam’, *Security and Communication Networks* **vol. 8**(2), 176–188.
- Sidorov, Z. (2017), ‘Rebreakcaptcha: Breaking google’s recaptcha v2 using google’, <https://east-ee.com/2017/02/28/rebreakcaptcha-breaking-googles-recaptcha-v2-using-google/>. Accessed on 2017-08-14.
URL: <https://east-ee.com/2017/02/28/rebreakcaptcha-breaking-googles-recaptcha-v2-using-google/>

- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016), ‘Mastering the game of Go with deep neural networks and tree search’, *Nature* **vol. 529**(7587), 484–489.
- Sim, T., Nejati, H. and Chua, J. (2014), Face recognition captcha made difficult, *in* ‘Proceedings of the 23rd International Conference on World Wide Web’, WWW ’14 Companion, ACM, Seoul, Korea, pp. 379–380.
URL: <http://doi.acm.org/10.1145/2567948.2577321>
- Sivakorn, S., Polakis, I. and Keromytis, A. D. (2016a), I am robot:(deep) learning to break semantic image CAPTCHAs, *in* ‘2016 IEEE European Symposium on Security and Privacy (EuroS&P)’, IEEE, Saarbrücken, Germany, pp. 388–403.
- Sivakorn, S., Polakis, J. and Keromytis, A. D. (2016b), I’m not a human: Breaking the Google reCAPTCHA, *in* ‘Black Hat 2016’, number i, Black Hat, Nevada, United States, pp. 1–12.
- Smith, C. (2016), ‘Brand new Pokemon Go feature may block you from cheating’. Accessed on 2017-08-14.
URL: <http://bgr.com/2016/08/25/pokemon-go-cheats-hacks-ban/>
- Stark, F., Hazirbas, C., Triebel, R. and Cremers, D. (2015), CAPTCHA Recognition with Active Deep Learning, *in* ‘Workshop New Challenges in Neural Computation 2015’, GI Fachgruppe Neuronale Netze and German Neural Networks Society, Aachen, Germany, p. 94.
- Sun, J., Yuan, L., Jia, J. and Shum, H.-Y. (2005), Image completion with structure propagation, Vol. vol. 24, ACM, Los Angeles, California, pp. 861–868.
- Susilo, W., Chow, Y.-W. and Zhou, H.-Y. (2010), Ste3d-cap: Stereoscopic 3d CAPTCHA, *in* ‘International Conference on Cryptology and Network Security’, Springer, Springer, Kuala Lumpur, Malaysia, pp. 221–240.
- Swire, P. (2004), ‘A model for when disclosure helps security: What is different about computer and network security?’, *Journal on Telecommunications and High Technology Law* **vol. 2**.

- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R. (2013), ‘Intriguing properties of neural networks’, *arXiv preprint* **abs/1312.6199**.
- Taigman, Y., Yang, M., Ranzato, M. and Wolf, L. (2014), Deepface: Closing the gap to human-level performance in face verification, *in* ‘The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)’, IEEE, Columbus, OH, USA, pp. 161–172.
- Tam, J., Simsa, J., Hyde, S. and von Ahn, L. (2008), Breaking audio captchas, Curran Associates, Inc., Vancouver, British Columbia, Canada, pp. 1625–1632.
- Tassi, P. (2011), ‘Chinese Prisoners Forced to Farm World of Warcraft Gold’. Accessed on 2017-08-14.
URL: <https://www.forbes.com/sites/insertcoin/2011/06/02/chinese-prisoners-forced-to-farm-world-of-warcraft-gold/>
- Thornton, C., Hutter, F., Hoos, H. H. and Leyton-Brown, K. (2013), AutoWEKA: Combined selection and hyperparameter optimization of classification algorithms, *in* ‘Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining’, ACM, Chicago, IL, USA, pp. 847–855.
- Vincent, D. (2011), ‘China used prisoners in lucrative internet gaming work’. Accessed on 2017-08-15.
URL: <https://www.theguardian.com/world/2011/may/25/china-prisoners-internet-gaming-scam>
- Viola, P. and Jones, M. (2001), Rapid object detection using a boosted cascade of simple features, *in* ‘Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001’, Vol. 1, IEEE, Kauai, Hawaii, USA, pp. I–I.
- Von Ahn, L. and Dabbish, L. (2004), Labeling images with a computer game, *in* ‘Proceedings of the SIGCHI conference on Human factors in computing systems’, ACM, pp. 319–326.
- Von Ahn, L., Maurer, B., McMillen, C., Abraham, D. and Blum, M. (2008), ‘reCAPTCHA: Human-based character recognition via web security measures’, *Science* **321**(5895), 1465–1468.
- Wallace, G. K. (1992), ‘The jpeg still picture compression standard’, *IEEE transactions on consumer electronics* **vol. 38**(1), xviii–xxxiv.

- Wang, J. (2014), ‘Secret signaling system’. Accessed on 2017-08-16.
URL: http://tetraph.com/covert_redirect/oauth2_openid_covert_redirect.html
- Warner, O. (2009), ‘Kittenauth’. Accessed on 2017-08-16.
URL: <http://www.thepcspy.com/kittenauth>
- Welch, T. A. (1984), ‘A technique for high-performance data compression.’, *IEEE Computer* **vol. 17**(6), 8–19.
URL: <http://dblp.uni-trier.de/db/journals/computer/computer17.html#Welch84>
- Wells, M. (2011), ‘Super captcha goes 3d’. Accessed on 2017-08-16.
URL: <https://goldsborrowwebdevelopment.com/article/2011/10/super-captcha-goes-3d/>
- Wittel, G. L. and Wu, S. F. (2004), On Attacking Statistical Spam Filters, *in* ‘Proceedings of the First Conference on EMail and Anti-Spam, CEAS’, Mountain View, Calif.:CEAS, Mountain View, CA, USA.
- Xu, Y., Reynaga, G., Chiasson, S., Frahm, J.-M., Monroe, F. and Van Oorschot, P. (2012), Security and usability challenges of moving-object CAPTCHAs: decoding codewords in motion, *in* ‘Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)’, USENIX, Bellevue, WA, USA, pp. 49–64.
- Yan, J. and Ahmad, a. E. (2007), Breaking Visual CAPTCHAs with Naive Pattern Recognition Algorithms, *in* ‘Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)’, IEEE, Miami Beach, Florida, USA, pp. 279–291.
- Yan, J. and Ahmad, A. S. E. (2008), A low-cost attack on a microsoft captcha, *in* ‘Proceedings of the 15th ACM conference on Computer and communications security’, ACM, Alexandria, VA, USA, pp. 543–554.
- Zetter, K. (2010), ‘Wiseguys plead guilty in ticketmaster captcha case’, *Wired.com* **November**(1).
- Zhou, X.-c., Shen, H.-b., Huang, Z.-y. and Li, G.-j. (2012), ‘Large margin classification for combating disguise attacks on spam filters’, *Journal of Zhejiang University SCIENCE C* **vol. 13**(3), 187–195.
- Zhu, B. B., Yan, J., Li, Q., Yang, C., Liu, J., Xu, N., Yi, M. and Cai, K. (2010a), Attacks and design of image recognition captchas, *in* ‘Proceedings of the 17th ACM conference on Computer and communications security’, CCS ’10, ACM, Chicago, Illinois, USA, pp. 187–200.

Zhu, B. B., Yan, J., Li, Q., Yang, C., Liu, J., Xu, N., Yi, M. and Cai, K. (2010*b*), Attacks and design of image recognition captchas, *in* ‘Proceedings of the 17th ACM conference on computer and communications security’, ACM, Chicago, IL, USA, pp. 187–200.