The Threat of Screenshot-Taking Malware: Analysis, Detection and Prevention

Hugo Sbai

Balliol College

University of Oxford

A thesis presented for the degree of

Doctor of Philosophy

February 2022

Abstract

Among the various types of spyware, screenloggers are distinguished by their ability to capture screenshots. This gives them considerable nuisance capacity, giving rise to theft of sensitive data or, failing that, to serious invasions of the privacy of users. Several examples of attacks relying on this screen capture feature have been documented in recent years.

On desktop environments, screenshot APIs are widely used by legitimate applications that provide screen sharing, screen casting, remote control, employee control, and parental control. This makes malicious use of the screenshot functionality particularly stealthy. Existing malware detection approaches are not adapted to screenlogger detection due to the composition of their datasets and the way samples are executed.

Moreover, the available countermeasures either suffer from a lack of usability that prevents their large-scale use or have a limited effectiveness.

In this thesis, I propose a defence-in-depth approach combining prevention and detection against screenshot-taking spyware. This approach was developed after an extensive analysis of this spyware category.

Our detection model achieves an accuracy of 97.4% versus 94.3% for a standard state of the art detection model. This model was trained and tested on the first complete and representative dataset dedicated to malicious and legitimate screenshot-taking applications.

Our prevention mechanism is based on the retinal persistence property of the human visual system. Its usability was tested with a panel of 119 users.

Keywords— Spyware - Screenlogger - Screenshot - Malware Analysis - Malware Dataset - Malware Detection - Retinal Persistence - Flicker Fusion - Defence-In-Depth

Statement of Originality

This thesis is written in accordance with the regulations for the degree of Doctor of Philosophy. The thesis has been composed by myself and has not been submitted in any previous application for any degree. The work presented in this thesis is my own.

Acknowledgements

This research was supervised by Michael Goldsmith and Jassim Happa, and I would like to thank them for their continued support, guidance and advice throughout the project. Aside from these direct supervisors, I am very grateful to the rest of our research group, who have all provided advice throughout.

I would also like to thank Professors David De Roure, Kurt Debattista, Ivan Martinovic and Kasper Rasmussen, who, through their comments at my Transfer of Status, Confirmation of Status and final viva, provided me with invaluable feedback and direction, with which this thesis was much improved.

Thank you especially to all the friends who have made this process both more manageable and more enjoyable. Finally, I would like to thank my family for everything they have done.

Table of Contents

1 Introduction **2**

1.1 Context and motivation 2

1.2 Research questions and Contributions 6

1.3 Detailed design 8

1.4 Thesis structure 11

1.5 List of publications 13

2 Literature Review **14**

2.1 Screenlogger behaviour analysis and construction of a dataset . . . 14

2.1.1 Screenlogger behaviour analysis 14

2.1.2 Existing datasets 14

2.2 Malware detection 16

2.2.1 Signature-based detection 16

2.2.2 Anomaly-based detection 17

2.2.3 Behaviour-based detection 18

2.2.4 Synthesis 30

2.3 Screenlogging prevention 32

2.3.1 Screenlogger prevention: the authentication case 32

2.3.2 Screenlogger prevention: the general case 45

2.3.3 Synthesis 62

3 Adversary Model **67**

3.1 Screenloggers' capabilities and comparison with other attacks . . 67

3.1.1 Credential theft on virtual keyboards 68

3.1.2 Sensitive data breach 74

3.1.3 Spying on the victim's activity 76

3.1.4 Blackmail 78
3.1.5 Reconnaissance 80
3.1.6 Synthesis 82
3.2 Attack scenarios 83
3.2.1 Online banking customer attack (capability 1 + capability 2) 83
3.2.2 Real-time monitoring of the victim’s activity (capability 3) 85
3.2.3 Blackmail (capability 4) 87
3.3 System Model 88
3.3.1 Targeted systems 88
3.3.2 Targeted victims 88
3.4 Threat Model 89
3.4.1 General Description 89
3.4.2 Operating process 89
3.4.3 Scope 90

4 Threat Analysis 93

4.1 Methodology 93
4.1.1 Extensive study of security reports 93
4.1.2 Proposed taxonomy 94
4.2 Screen capturing 99
4.2.1 Used API 99
4.2.2 Screenshot-triggering 99
4.2.3 Captured area 102
4.3 Screenshots storage 103
4.3.1 Image compression 103
4.3.2 Storage media 104
4.4 Screenshots exfiltration 105
4.4.1 Communication protocol 105

4.4.2 Encryption 106
4.4.3 Screenshots sending triggering 107
4.5 Synthesis: Completeness requirements 108
4.5.1 Behavioural completeness 109
4.5.2 Proportional completeness 111
4.5.3 Discussion 112

5 Dataset Construction 114
5.1 Malicious dataset 114
5.1.1 Data collection and analysis 114
5.1.2 Challenge 1: Ensuring that screenshots are taken 115
5.1.3 Challenge 2: Representativeness of the dataset 121
5.2 Benign dataset 133
5.2.1 Screenshot-taking application categories 134
5.2.2 Data collection and analysis 136

6 Behavioural Comparison 138
6.1 Screen capturing 138
6.1.1 Used API 138
6.1.2 Screenshot triggering 139
6.1.3 Captured area 139
6.2 Screenshots storage 140
6.2.1 Image compression 140
6.2.2 Storage media 141
6.3 Screenshots exfiltration 142
6.3.1 Communication protocol 142
6.3.2 Screenshot sending triggering 143
6.3.3 Encryption 143

6.4 Hypotheses for detection 144
6.4.1 Screen capturing 144
6.4.2 Screenshots storage 146
6.4.3 Screenshots exfiltration 146
6.5 Discussion 148

7 Behavioural Screenlogger Detection 151
7.1 Experimental Setup 151
7.2 Performance measurements 152
7.3 Basic detection approach 153
7.3.1 Feature extraction 153
7.3.2 Detection algorithm 157
7.3.3 Model training and testing 158
7.3.4 Feature selection 159
7.3.5 Evaluation 160
7.4 Optimised detection approach 164
7.4.1 New features 164
7.4.2 Evaluation 170
7.5 Discussion 171

8 Leveraging Retinal Persistence for a Usable Countermeasure to Malicious Screenshot Exploitation 173
8.1 Proposed model: On-the-fly screenshot alteration 175
8.2 Developing several retinal persistence based algorithms 178
8.2.1 Pattern used inside hidden areas 181
8.2.2 Determining hidden areas 184
8.2.3 Frequency of display 192
8.3 Hypotheses 192

9 Evaluation and Security Analysis of the Proposed Countermeasures 195

9.1 Evaluation criteria 195

9.1.1 Security analysis 195

9.1.2 Usability 196

9.1.3 Real-time 197

9.1.4 Network bandwidth 198

9.2 Methodology 198

9.2.1 Security analysis 199

9.2.2 Usability 205

9.2.3 Real-time 215

9.2.4 Network bandwidth 216

9.3 Results 217

9.3.1 Security analysis 217

9.3.2 Usability 221

9.3.3 Real-time 224

9.3.4 Network bandwidth 226

9.4 Discussion 227

10 Conclusion 230

10.1 Contributions 230

10.1.1 Threat analysis (Chapters 4 and 6) 230

10.1.2 Dataset construction (Chapter 5) 231

10.1.3 Screenlogger detection (Chapter 7) 232

10.1.4 Retinal persistence-based mitigation technique (Chapters 8, 9 and 10) 233

10.2 Limitations 234

10.3 Final remarks and future work 236

List of Figures

1.1 Screenloggers problem. 3

1.2 Carbanak cybergang attack using screenshots [1] 5

1.3 Design of the proposed solution. 9

1.4 Example of altered screenshot. 10

2.1 Drawbacks of existing detection methods for screenloggers 32

2.2 Examples of anti-shoulder surfing authentication techniques using obfuscation and confusion. 34

2.3 Examples of anti-shoulder surfing authentication techniques using cognitive tasks. 36

2.4 Examples of anti-shoulder surfing authentication techniques using graphical passwords. 39

2.5 Example of gesture-based authentication [2]. 40

2.6 Secure Haptic Keypad [3]. 42

2.7 XSide system [4]. 42

2.8 Formation of the number '4' using visual persistence [5]. 43

2.9 Authentication system proposed in [6]. 45

2.10 Shuffling text method [7]. 47

2.11 Sample mappings of sensitive private data elements to their corresponding Cashtag alias [8]. 48

2.12 Filters applied to photos against shoulder surfing [9]. 49

2.13 Samples of results from the visual cryptography approach proposed by Hou et al. [10]. 58

2.14 Samples of result from the visual cryptography approach proposed by Grange et al. [11]. 59

2.15 System overview and distortion planes used by Chia et al. [12]. . . 60

2.16 Images projection using moving privacy black bars [13]. 62

3.1 LCL bank's virtual keyboard. 72
3.2 Oney bank's virtual keyboard. 72
3.3 Screenshot attack scenario 1. 84
3.4 Screenshot attack scenario 2. 86
3.5 Screenshot attack scenario 3. 87
3.6 Threat model message sequence chart with different settings. . . . 91

4.1 Need for a screenshot command to start screen capturing (in the malware of Mitre [14]). 100
4.2 Screenshot-triggering (in the malware of Mitre [14].) 101
4.3 Captured area. 103
4.4 Image files format. 104
4.5 Files storage. 105
4.6 Communication protocol. 106
4.7 Image files encryption. 107

5.1 Windows screen-capturing functions. 118
5.2 Screenshot-triggering. 123
5.3 Malware infiltrating the default browser process (Navegador padrao). 124
5.4 Captured area (in the malware of the dataset). 125
5.5 Image files format (in the malware of the dataset). 126
5.6 Image files storage (in the malware of the dataset). 127
5.7 Screenlogger generator. 130
5.8 Screenlogger generator process. 131

5.9 Screenlogger generator during execution: (a) When the option 'with command' is chosen (b) When the option 'without command' is chosen. 132

6.1 Screen capture size (legitimate applications). 140
6.2 Screenshots files format (legitimate applications). 141
6.3 Screenshots storage (legitimate applications). 142
6.4 Communication protocol (legitimate applications). 143
6.5 Image files encryption (legitimate applications). 144

8.1 Normal function-images vs. Microsoft Detours function-images ([15]). 176
8.2 RPASS 177
8.3 Punctual screenshot: solution 1. 180
8.4 Example of uniform pattern. 182
8.5 Example of Gaussian blur pattern. 182
8.6 Example of hierarchical pattern. 184
8.7 Examples of text screenshots altered with the periodical vertical sliding bars algorithm (captured at different times) 185
8.8 Examples of text screenshots altered with the periodical concentric circles algorithm (captured at different times) 185
8.9 Examples of text screenshots altered with the random vertical rectangles algorithm (captured at different times) 188
8.10 Examples of text screenshots altered with the random horizontal rectangles algorithm (captured at different times) 188
8.11 Examples of text screenshots altered with the random circles algorithm (captured at different times). 191

9.1 Step 3 instructions 201

9.2 Step 3 of the user test : determine the number of incomplete screenshots necessary for the user to be able to read the text 201
9.3 Step 4 instructions 202
9.4 Step 4 of the user test : determine the number of incomplete screenshots necessary for the user to read a specific information on the screen (example of a hotel reservation where the user must read the arrival date, departure date and city) 203
9.5 Step 1 instructions 209
9.6 Step 1 before clicking on ‘Start Reading’ button 210
9.7 Step 1 after clicking on ‘Start Reading’ button 210
9.8 Step 1 after clicking on ‘I finished reading’ button 211
9.9 Step 1 final screen (confirm usability scores) 212
9.10 Step 2 instructions 213
9.11 Step 2 before clicking on the ‘Show the Code’ button 214
9.12 Step 2 after clicking on the ‘Show the Code’ button 214
9.13 Step 2 after 3.5 seconds 215

List of Tables

2.1 Anti-segmentation and anti-recognition techniques and their weaknesses. 53

2.2 Limitations of existing approaches against screenshots. 65

4.1 Criteria of completeness. 95

4.2 Criteria of completeness. 110

4.3 Screenloggers behaviours (samples selected from [14]). 111

5.1 Screenlogger behaviours in our dataset vs those found in security reports selected from Mitre [14]. 129

6.1 Screenloggers behaviours in the malicious dataset vs those found in the legitimate dataset. 149

7.1 Detection results for the basic approach using features from the literature with the RF algorithm (k=10). 160

7.2 Detection results for the basic approach using features from the literature with the KNN algorithm (k=10). 161

7.3 Detection results for the basic approach using features from the literature with the SVM algorithm (k=10). 161

7.4 Detection results for the basic approach using features from the literature with the RF algorithm (k=3). 162

7.5 Detection results for the basic approach using features from the literature with the RF algorithm (k=5). 162

7.6 Detection results for the basic approach using features from the literature with the RF algorithm (k=7). 163

7.7 Detection results for the basic approach using features from the literature with the RF algorithm (k=12). 163
7.8 Detection results for the basic approach using features from the literature with the RF algorithm (k=15). 163
7.9 Detection results for the optimised approach using our specific features. 170

9.1 Number of successive screenshots needed by users. 218
9.2 Probability that a state of the art OCR will read the full text (350 characters) with different numbers of incomplete screenshots. . . . 220
9.3 Probability that a state of the art OCR will extract a specific piece of information (350 characters) with different numbers of incomplete screenshots. 220
9.4 Detection results when we simulate an increase in screenshot frequency. 221
9.5 Passive user usability evaluation. 221
9.6 Number of images generated per second. 225
9.7 Images compression rates. 226

Acronyms

ADB Android Debug Bridge. 6, 14, 67, 68, 70

ANN Artificial Neural Network. 27, 29

API Application Programming Interface. iv, v, 8, 9, 18–22, 28, 29, 31, 54, 67, 69, 82, 89, 92, 96, 99, 114–117, 122, 137, 138, 144, 151, 154, 159, 161–163, 165–171, 174–176, 207, 232–235

ARP Address Resolution Protocol. 74, 75

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart. 50–54, 63, 196, 197

DC Device Context. 23, 96, 97, 118, 169, 175, 176

DD Desktop Duplication. 67, 96, 99, 116, 117, 122, 131, 138, 144, 145

DLL Dynamic-link library. 18, 19, 29

DNS Domain Name System. 74, 75

DOM Document Object Model. 71, 76, 92

DT Decision Tree. 21, 22, 25, 157–159

FD Factor of Displacement. 37

FN False Negatives. 152, 160–163, 170, 221

FP False Positives. 152, 160–163, 170, 221

FPS frame per second. 179, 192, 198, 207, 208, 226, 228, 235

GDI Graphics Device Interface. 67, 96, 97, 99, 116, 117, 122, 131, 138, 144, 175

GDPR General Data Protection Regulation. 208

HVS Human Visual System. 9, 56, 59, 60, 64, 236

KNN K-nearest neighbour. xii, 21, 25, 27, 29, 53, 161

ML Machine learning. 7, 9, 17, 18, 20, 21, 23, 27–29, 50, 73, 112, 151, 153, 165, 172

NLP Natural Language Processing. 54, 63, 85, 87, 90

OCR Optical Character Recognition. xiii, 10, 26, 48, 50–53, 83, 85, 87, 90, 98, 107, 166, 173, 193, 203, 204, 219, 220, 228, 234

PSNR Peak signal-to-noise ratio. 196

RAT Remote Access Trojan. 15, 77, 96, 101, 120, 122–124

RF Random Forest. xii, xiii, 21, 25, 27, 157–163, 171, 232

RFE Recursive Feature Elimination. 159, 163, 171, 232

RPASS Retinal Persistence based Anti-Screenshot Solution. x, 9, 177

STM Shuffling Texts Method. 46

SVM Support Vector Machine. xii, 161

TN True Negatives. 152

TP True Positives. 152

1 | Introduction

1.1 Context and motivation

Spyware can be defined as software that gathers information about a person or organisation without their consent or knowledge and sends it to another entity [16]. The software is designed for secrecy and durability. A long-term connection to the victim's machine is established by the adversary, and once spyware is installed on the victim's device, it aims to steal information unnoticed. The most powerful spyware attackers targeting specific entities can launch an advanced persistent threat attack to access sensitive information while remaining undetected for the longest possible period [17].

Spyware is usually organised in multiple modules, each performing one or more malicious activities, with the ability to use them according to the attacker's purpose [18]. For example, AnubisSpy is an Android spyware that includes update and destruction modules, in addition to spying modules such as a screenshot module or a location module [19]. Typical spyware modules include keystroke logging, screen logging, URL monitoring, turning on the microphone or camera, intercepting sensitive documents and exfiltrating them and collecting location information [20].

Among the aforementioned spyware modules, screenloggers have one of the most dangerous functionalities in today's spyware as they greatly contribute to hackers achieving their goals, as illustrated in figure 1.1.

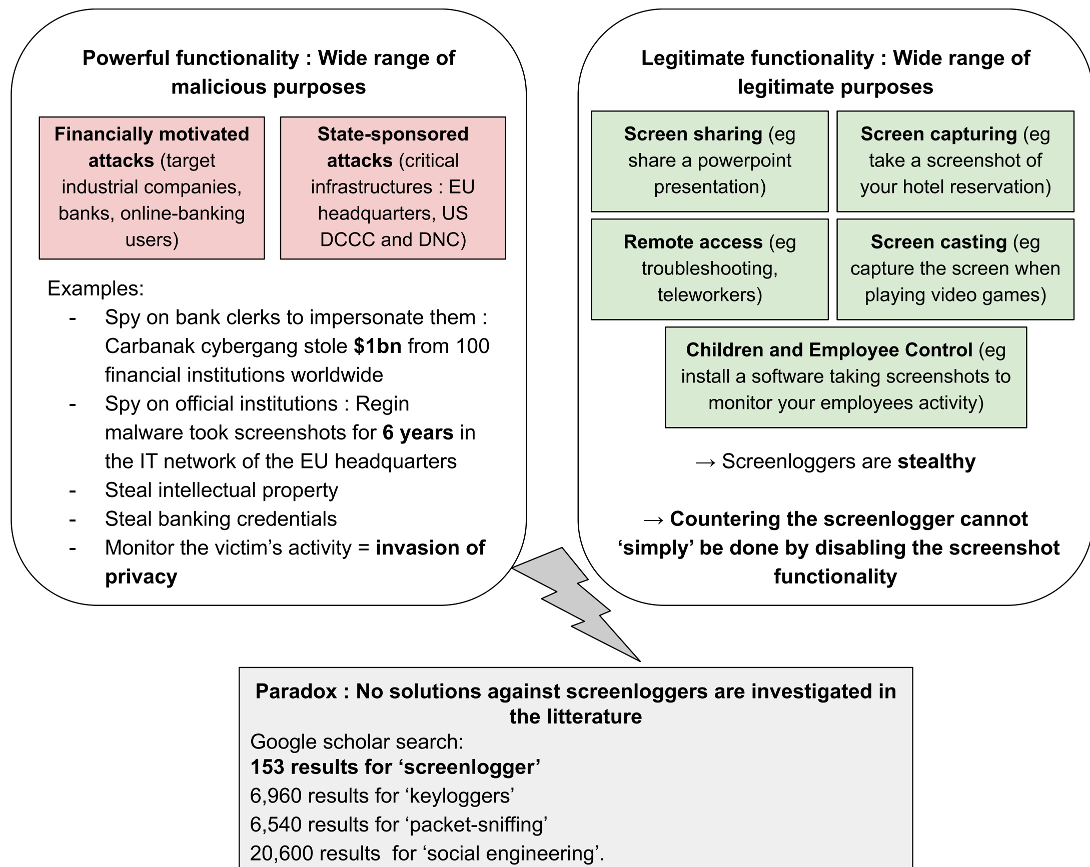


Figure 1.1: Screenloggers problem.

Screenlogger users can be divided into two categories: financially motivated actors and state-sponsored attackers. The first category targets important industrial companies (e.g., BRONZE BUTLER [19]), online banking users (e.g., RTM [21], FIN7 [22], Svpeng [19]), and even banks themselves (e.g., Carbanak [21], Silence [23]). The second category, which is even more problematic, targets critical infrastructure globally. For instance, the malware TinyZbot [24], a variant of Zeus [25], has targeted critical infrastructure in more than 16 countries. More precisely, the targets can be democratic institutions; for instance, XAgent targeted the US Democratic Congressional Campaign Committee and the Democratic National Committee [26]. In Europe, Regin [23], took screenshots for at least six years in

the IT network of the EU headquarters. Diplomatic agencies have also been compromised, for example, North Korean malware ScarCruft has targeted diplomatic agencies [27]. US defence contractors have also been hit by spyware, such as Iron Tiger.

Screenloggers have the advantage of being able to capture any information displayed on the screen, offering a large set of possibilities for the attacker compared to other spyware functionalities. Moreover, malware authors are inventive when maliciously using screen captures. Indeed, screen captures have a wide range of purposes. Some malware, such as Cannon and Zebrocy, only take one screenshot during their entire execution as reconnaissance to see if the victim is worth infecting [28, 29]. Others hide what is happening on the victim's screen by displaying a screenshot of their current desktop (FinFisher [29, 30]). Others take numerous screen captures for a close monitoring of the victim's activity. This can allow stealing of sensitive intellectual property data (Bronze Butler [4]), banking credentials (RTM [21], FIN7 [22], Xagent [26]), or to monitor the day to day activity of banking clerks to understand the banks' internal mechanisms (Carbanak [31], Silence [23]).

The screenshot capability is sometimes the unique functionality used in some phases of an attack to observe while remaining stealthy. For instance, in the Carbanak attack targeting banking employees [20, 32], attackers used the screengrabs to create a video recording of daily activity on employees' computers, as illustrated in figure 1.2 The hackers amassed knowledge of internal processes before stealing money by impersonating legitimate local users during the next phase of the attack.

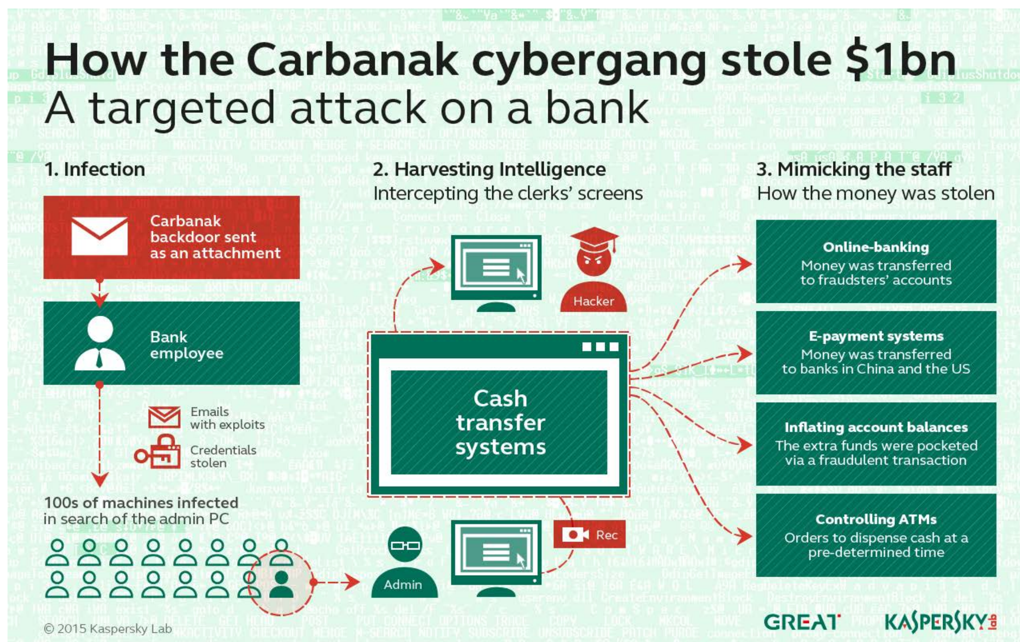


Figure 1.2: Carbanak cybergang attack using screenshots [1]

These examples show that the screenshot functionality is widely used today in modern malware programs and can be particularly stealthy, enabling powerful attacks. Even in the case where no specific attack is performed, the simple fact of monitoring and observing all the victims' activity on their device is a serious invasion of privacy. Moreover, screenshots are likely to contain personally identifiable information [33].

What makes the screenlogger threat even more problematic is that, on desktop environments, the screenshot functionality is legitimate and, as such, is used by many benign applications(e.g. screen sharing for work or troubleshooting, saving important information, creating figures, monitoring employees). The necessity of capturing the screen has for instance increased with telework, even on sensitive machines. Teleworkers, including bank employees or lawyers, may need to control their office computer remotely from home, or to share their screens during

online meetings. Therefore, countering the screenlogger threat cannot 'simply' be done by disabling the screenshot functionality on sensitive machines. Other natural approaches such as white-listing are prone to the ever more sophisticated strategies malware authors can deploy to inject malicious code into legitimate processes or bypass the user's consent.

Paradoxically, only a few works appear in the literature about screenloggers, leaving the threat relatively unknown. This gap is revealed by the low number of occurrences of the words 'screenlogger' and affiliated keywords on Google Scholar compared to other threats: 153 for 'screenlogger' vs 6,960 for 'keyloggers', 6,540 for 'packet-sniffing' or 20,600 for 'social engineering'.

To the best of our knowledge, the only studies focusing on screenloggers have been limited to specific questions, such as the Android Debug Bridge (ADB) vulnerability that allows screenshot-taking on Android smartphones [34–37], or screenshot protection during authentication using virtual keyboards [38–40]. No overall view is available of the threat represented by screenshot-taking malware and how it can be countered. Therefore, the objective of this thesis is to address the lack of emphasis in the literature on screenloggers by studying their behaviour and proposing a defence-in-depth approach to fight them.

1.2 Research questions and Contributions

The research questions we are addressing in this thesis are the following:

- What are the behaviours displayed by screenshot-taking malware in the wild?
- Can we build a complete and representative dataset dedicated to screenshot-taking malware and legitimate applications?

- What are the most effective features to detect screenshot-taking malware and distinguish them from legitimate applications?
- Is it possible to automatically prevent malicious screenshot exploitation on a large scale without requiring user intervention?

In order to propose a complete and coherent approach against spyware programs taking screenshots, this thesis brings five main contributions:

- Threat analysis: creation of novel criteria to classify the behaviours exhibited by screenshot-taking malware.
- Construction of the first dataset of malicious and benign screenshot-taking applications with two main constraints: ensuring that screenshots are taken during the analysis and ensuring the representativeness of the dataset using the aforementioned criteria.
- Use of the constructed dataset to identify the existing features which are the most adapted to the detection of screenloggers.
- Development of the first behavioural detection system adapted to the specifics of screenshot-taking malware. Instead of blindly relying on seemingly meaningless data given to a Machine learning (ML) model, we propose new features that reflect the behaviours shown by screenloggers, as opposed to legitimate screenshot-taking applications.
- Use of the retinal persistence property of the human eye to mitigate manual and automatic screenshot exploitation while trying to achieve the best possible usability. Forcing malware programs to take screenshots more frequently makes them more easily detectable by our specific behavioural detection system. Several experiments were conducted to ensure the method's robustness and the possibility to deploy it widely, on four main aspects: se-

curity (as measured against human users as well as automatic algorithms), usability (measured objectively and subjectively with actual users), real time (can the images be generated sufficiently fast so that retinal persistence can work?) and network bandwidth (in most cases, this approach reduces the screenshot size, which allows for sending screenshots more frequently).

1.3 Detailed design

As illustrated in Figure 1.3, the proposed defence-in-depth mechanism lies between the application level (running programs) and the OS. It is triggered whenever a program requests a screenshot to the OS. Identifying such an event is not straightforward as there is not a unique Application Programming Interface (API) call to take a screenshot. By executing and analysing hundreds of malware and legitimate applications, we identified the different sequences which characterise a screen capture. These sequences include the different mechanisms that screenloggers could use to evade detection: API calls out of sequence, spaced out in time, or interleaved with other API calls.

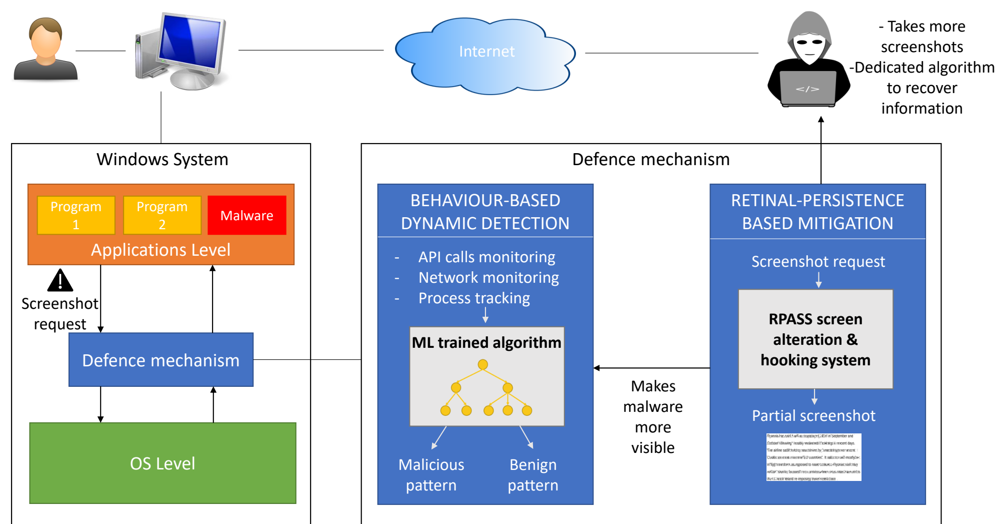


Figure 1.3: Design of the proposed solution.

Our defence mechanism is composed of two parts.

The first part is a behaviour based dynamic detection system. As soon as a program makes a screenshot request, this detection system starts monitoring its API calls and network traffic. It also tracks the different background and foreground processes which are linked to this event. All this data is given to a ML model that we trained in order to distinguish benign and malicious patterns.

The second part is a Retinal Persistence based Anti-Screenshot Solution (RPASS). This system intercepts the screenshot requests and returns an altered screenshot instead of returning the normal screenshot. An example of this is illustrated in figure 1.4. The goal is to force the attacker to take more screenshots while enabling the user to see the screen (screen sharing, remote control). This is possible thanks to the retinal persistence property of the Human Visual System (HVS). If the user needs to take a simple unique screenshot, the only way to do so is to

use the special “print screen" key on the keyboard, taking the necessary hardware precautions to avoid the keypress being simulated.

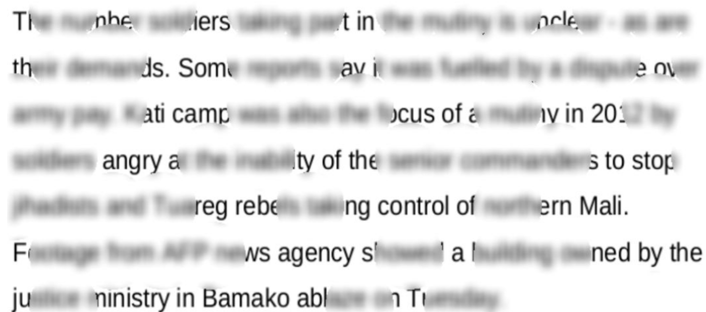


Figure 1.4: Example of altered screenshot.

The mitigation system has two major effects. The first one is to make it harder for an attacker to recover information from the screenshots. They need to take several screenshots in a limited time window and to use a specific algorithm to reconstruct the screen. Indeed, we designed the alteration mechanism in a way that a simple majority rule would not suffice; a pixel can be more time visible than hidden but it can also be the reverse. Therefore, an Optical Character Recognition (OCR) will be necessary to analyse the successive images and find what parts are hidden, followed by an algorithm which will combine the different parts.

The second effect of this mitigation mechanism is that, as the attacker needs to take more screenshots, it becomes harder to remain undetected. Indeed, it is not possible to capture the screen at spaced intervals of time to remain stealthy. Therefore, this mechanism is complementary with the detection module and makes its task easier.

1.4 Thesis structure

The rest of this thesis is organised as follows:

The main works partially responding to the issues raised in the thesis; namely, the detection and prevention of screenloggers, are presented and analysed in Chapter 2. The studied prevention approaches have been intentionally extended to anti-'shoulder surfing' countermeasures, even if this type of attack is outside the scope of the thesis. This choice was made because of the important similarities between shoulder-surfing attacks and those that are the subject of our work and the lack of countermeasures against screenloggers.

Chapter 3 presents the various uses of screenshot-taking spyware to highlight the dangers represented by this type of malware. These malicious capabilities are further demonstrated through attack scenarios of varying complexity, all of which correspond to plausible real-life situations. The presentation of the threat and system model completes the chapter by clearly defining target environments as well as the scope and limitations set for this work.

Chapter 4 presents our threat analysis methodology. This work aims to remedy the lack of behavioural studies specific to screenloggers in the literature. The behaviour of screenlogging malware is studied and analysed in detail at each stage of its operating mode using novel criteria specially created for the occasion. The chapter concludes with a definition of the completeness criteria for our screenlogger dataset.

The construction of a dataset dedicated to screenshot-taking malware and legitimate screenshot-taking applications is outlined in Chapter 5. The challenges faced and their solutions are thoroughly explained.

Chapter 6 offers a detailed comparative study of screenlogging spyware and legitimate applications that take screenshots. This comparison covers all the steps of the operating mode of the software considered. Differences and similarities between legitimate applications and malware are highlighted and analysed to identify promising criteria for screenlogger detection.

Chapter 7 outlines our detection methodology against screenloggers, which uses new categories of features. A detection model using only traditional features from the literature and another using our novel features are compared. This comparison allows for conclusions to be drawn on the effectiveness of the new detection methodology and discusses its extension to other categories of malware.

Chapter 8 presents our retinal persistence-based approach against malicious screenshot exploitation. The method is based on hiding some parts of the screenshot on the fly before its transmission to the application that requested it. The proposed algorithms vary in three main aspects: the way hidden areas are determined, the pattern inside the hidden areas and the frequency of the display. One of the most important characteristics of the proposed methodology is that it aims to be transparent to the user, contrary to existing approaches that are too intrusive to be widely deployed.

The approach proposed in Chapter 8 is evaluated in Chapter 9. Various parametric combinations for different algorithms are evaluated for four criteria. Experiments are conducted to show that the screenshot-altering method described in Chapter 8 improves the performances of the detection model presented in Chapter 7 by forcing the attacker to take more screenshots. The usability of the approach and its security against human attackers are tested with a panel of more than 100 users. The results obtained are extensively discussed and analysed.

Chapter 10 concludes this work.

1.5 List of publications

The contributions of this thesis have led to four conference papers:

- A Survey of Keylogger and Screenlogger Attacks in the Banking Sector and Countermeasures to them: paper published in the 10th International Symposium on Cyberspace Safety and Security (CSS 2018) [41].
- Dataset Construction and Analysis of Screenshot Malware: paper published in the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2020) [42].
- A Novel Behavioural Screenlogger Detection System: paper published in the 24th Information Security Conference (ISC 2021) [43].
- RPASS: Retinal Persistence based Anti-Screenshot Solution: paper submitted to the 20th International Conference on Applied Cryptography and Network Security (ACNS 2022).

2 | Literature Review

In this chapter, we present an overview of the state of the art regarding the five contributions listed in Section 1.2.

2.1 Screenlogger behaviour analysis and construction of a dataset

2.1.1 Screenlogger behaviour analysis

To the best of our knowledge, the only studies focusing on screenloggers are limited to specific questions, which are the ADB vulnerability that allows screenshot-taking on Android smartphones [44–47], and screenshot protection during authentication using virtual keyboards [5, 48, 49].

No general view is available of the operating mode of this kind of spyware and the behaviours they exhibit at different stages of their execution. Only MITRE dedicates a page to screenshot-taking malware [14]. The MITRE page cites 127 security reports about screenshot-taking malware. However, the reports are unfortunately not compiled or analysed to give a clear view of the threat represented by screenshot-taking malware and the capabilities they possess.

2.1.2 Existing datasets

Although there are many malware datasets available containing diverse categories of malware [50, 51], there is no existing dataset dedicated to gathering diverse forms of screenshot-taking malware. The only screenlogger samples can be found in general malware datasets in the middle of other types of malware. However,

such general datasets would not allow us to test our future detection approach nor to gather meaningful insights into screenloggers' behaviour. There are several reasons for that.

First, general dataset authors do not indicate whether their dataset contains screenshot-taking malware. They usually simply indicate that the dataset contains 'spyware' without giving information about their functionalities and particularly the screenshot functionality. To the best of our knowledge, only one dataset explicitly includes screenloggers' network traffic related to the well-known malware programs Zeus and Ares [52]. However, the collected network traffic of the aforementioned screenloggers is not representative of the real screenloggers' behaviour because authors only consider a periodic screen recording of 400 seconds, whereas recent screenloggers are highly diverse. Thus, existing malware datasets include few screenlogger samples, and they are limited because they only reflect a small number of characteristics describing a specific screenlogger behaviour.

Second, the screenshot functionality is often not executed immediately when a machine is infected. In particular, for a Remote Access Trojan (RAT), screenshot-taking is triggered by a command sent by the remote attacker. There are also some cases where screenshots are triggered by user events (such as mouse clicks) or by the execution of certain programs or specific web pages (online banking, etc.). Consequently, even if existing datasets may contain screenloggers, the screenshot functionality will most probably not be used anyway, preventing us from making observations.

Third, as mentioned before, legitimate applications are increasingly taking screenshots. These applications are diverse: screen sharing (e.g., Skype [53], Screen Leap [54], Join.me [55]), remote control (e.g., TeamViewer [56], Netviewer [57], GoToMyPC [58]), screencasting (e.g., Camtasia [59], CamStudio [60], Ezvid

[61]), parental or employees control (e.g., Verity [62], Kidlogger [63], Norton Online Family7 [64]), or screenshot-taking and editing (e.g., PicPick [65], Snipping Tool [66], FastStone Capture [67]). Thus, to effectively analyse spyware using the screenshot capture feature, it is critical to be able to understand and identify the similarities and differences between their behaviour and that of legitimate applications. However, there is currently no dataset of legitimate applications taking screenshots and again, even when some of the legitimate applications contained in the dataset can take screenshots, it is important to ensure that this functionality is triggered at runtime (for example by enabling screensharing during a Skype call).

Due to the small number of works on screenloggers in the literature, existing malware datasets are unsuitable for creating an effective detection method. They do not mention whether screenloggers are included or not, and when they do, only basic and naive behaviours are covered due to the specificities of screenshot-triggering. The same problems apply to legitimate screenshot-taking applications.

2.2 Malware detection

Malware detection methods can be classified into three categories depending on the technique they use: Signature-based detection, Anomaly-based detection and Behaviour-based detection.

2.2.1 Signature-based detection

The basic idea of signature-based methods is that the signature of the program is extracted from its files and matched to known signatures. Those methods work using two steps. First, known and already detected malware programs are analysed, and their signatures are stored in a database; second, the signatures are compared to detect possible malware. Bazrafshan et al. described those methods using three

components: (1) a data collector which performs static or dynamic analysis, (2) an interpreter in charge of converting the collected data into an intermediate format, (3) a matcher, which matches the extracted features with already known behaviour signatures [68].

Those methods perform well for known malware programs, with a low false-positive rate. However, they are vulnerable to obfuscation techniques by which the attacker can alter the code and avoid detection because the signature is changed [69]. Moreover, signature-based methods cannot detect malware employing polymorphism and metamorphism because the signature of the malware changes every time a machine is infected, nor zero-day attacks. Therefore, the signature database must be updated continuously and must hold a signature for each variant of every known malware program.

2.2.2 Anomaly-based detection

Anomaly-based detection techniques consist of modelling the normal behaviour of a system to detect abnormal activities. The definition of normal behaviour can be done manually using heuristic methods. Those methods use manually maintained and updated rules to differentiate between benign and malicious programs. It is also possible to use ML. For instance, Tang et al. created models of Internet Explorer and Adobe reader behaviours using unsupervised ML techniques [70]. Authors used hardware performance counters as features to detect the deviation from the normal behaviour of those programs; they argue that those counters are altered when a malicious code is injected.

The drawback of these methods for detecting malware in computer systems is that those systems contain many processes and many possible behaviours, making it difficult to define normal behaviour. This can result in a high false-positive rate.

2.2.3 Behaviour-based detection

These methods analyse the behaviour of the program to determine whether it is malicious or not. The main approach used for behaviour analysis is ML. Using the features extracted by analysing the behaviour of the programs, a dataset is formed and used to train an ML model. The latter can then classify a program by receiving as input its features vector and giving as an output a decision about its nature (malware or benign application). The ML model can be trained using supervised or unsupervised methods. Using unsupervised methods, the classes are automatically identified by the model, whereas with supervised methods, malware programs are labelled in the dataset. A review of studies using ML models for malware detection has been made by Souri et al. [71]. The features describing a program can be static or dynamic.

Static analysis

Features can be extracted in a static way, which means that the binary files of the malware are analysed without executing it. Different features can be extracted from the binary files. The most frequently used features in the literature are:

- Dynamic-link library (DLL) imports: to access a specific API, programs needs to import DLLs. Those imports can be detected and can reflect their behaviour;
- API calls and their number of occurrences;
- Control flow graph: after disassembling the binary file of a program, a control flow instruction graph can be created. Nodes represent blocks of instructions, and edges represent control flow instructions. A method using control flow graphs was presented by Eskandari et al. [72]. Another exam-

ple of recent work using control flow graphs is [73], which is an approach for detecting Java bytecode malware programs using static analysis with a deep Convolutional Neural Network classifier to determine maliciousness.

- Opcode (Operation code): the opcode is a part of machine language instructions, along with the operand [74]. Opcode can be used to detect malicious behaviours. For instance, Nar et al. used opcode frequency histograms [75].
- N-gram: n-grams are all possible substrings of length N that can be obtained from a larger string. Using this method, malware programs are analysed using text processing algorithms.

However, static analysis fails to overcome obfuscation techniques [76]. Malware designers can use those techniques to disturb the analysis process and hide the malicious behaviour of the program, for example, by using goto and jump instructions or by importing unused DLL libraries.Han et al. compared API calls extracted using static and dynamic analysis and showed that API calls extracted statically from malware and benign applications are less different compared to those extracted dynamically [77].

Moreover, malware can encrypt itself with a different key each time it infects a machine. It is impossible to extract its features statically without being able to decrypt it. More specifically, the packing technique is a method of using multiple layers of compression and encryption to hide the malicious code. Anezi et al. argue that 80% of current malware use this technique, and half of them are repacked versions of old malware programs [78].

Dynamic analysis

Dynamic analysis consists in of running a malware program and monitoring its activities, such as API calls or, or registry, file system and network use. The mal-

ware can be run in a real environment, taking the necessary precautions to avoid its spreading; , or it can be run in virtual environments [79, 80]. Megira et al. present a more detailed description of tools used to analyse malware [81]. Dynamic analysis can overcome obfuscation techniques, since it analyses the runtime behaviour of the malware. Therefore, this technique is widely used in recent malware detection works, which consider diverse types of dynamic features as described in the following subsections.

Malware detection based on API calls monitoring is considered as the most widely used method, because in order to perform their operations, most malware must use the system APIs to perform their operations. Therefore, the behaviour of a malware program can be identified using the sequence of API calls and their number of occurrences. Several recent works use dynamic API calls extraction to detect malware [77, 82–84].

The basic principle of these methods is illustrated by Han et al. [77]. The authors developed a detection system called MalDAE, which employs both static and dynamic API calls analysis. Their method is based on several steps. In the first step, the data is collected by analysing the portable executable section of the binary files to extract the sequences of API calls sequences statically. Then, the program is run in a sandbox, its dynamic API calls are extracted, , and a pruning is done in order to eliminate the redundancy of API calls and no-op instructions. This step helps to eliminate noisy API calls used by the malware designers. Then, the sequences of API calls sequences from both methods are merged, and the features are generated by taking the N API calls with the highest contribution. Finally, the vector combining the N API calls is filled with the number of occurrences of each one, and this vector is passed as an input to an ML model to classify the programs. This method was evaluated using the VirusShare dataset [85], and five types of

malware were investigated: backdoor, rootkit, hoax, constructor and email-worm. Four ML models were tested: Random Forest (RF), K-nearest neighbour (KNN), Decision Tree (DT) and XGBoost, each one using only static features, then dynamic features and, and finally merged features. This last setting showed the best performance.

Malware developers, however, use ever more sophisticated methods to evade detection, and, as argued by Weijie et al., traditional methods based on API calls sequences may be insufficient [82]. For instance, they can be cheated by malware with strong imitation ability, which can reproduce the same API calls as legitimate programs. As a result, detection methods based on API calls [86, 87] failed in detecting some malware. Another drawback is that the feature vector generation is based on the Top-N API calls, which were the most current in the training samples [77, 82]. Thus, the selected API calls strongly depend on the nature of the dataset. Hence, if the dataset does not contain any screenshot-taking malware or very few, screenshot calls would not be taken into account. A recent work aiming at solving these issues is presented in [88]. The authors propose a novel ensemble adversarial dynamic behavior detection method aiming at three features of malicious API sequence, namely Immediacy, Locality, and Adversary. There results shows that this technique provides more resiliency compared to existing ones.

Other methods proceed differently by focusing on specific API calls instead of extracting all the API calls made by a program. This process enables extracting more features about the API calls instead of extracting only the number of occurrences. It is the case of Javaheri et al., who focused on detecting spyware and, more specifically, keyloggers, screen recorders, and blockers [89]. The authors proposed a dynamic behavioural analysis through the hooking of kernel-level routines. More precisely, for the screenlogger case, the presented method is designed to detect screenloggers under the Windows operating system. To this end, it hooks

the GetWindowDC and BitBilt functions, and uses the system service dispatcher table, which is the table that contains pointers to each kernel function. Then, to classify a screenshot-taking program as spyware or benign, they used a DT (with the J48 algorithm) considering the following features: frequency of repetition, uniqueness of the applicant process, state of the applicant process (hidden or not) and values of parameters in the system calls. The results showed that the proposed method could detect screenloggers with an accuracy of 92% and an error rate of 7%.

However, this method suffers from several weaknesses. Relying exclusively on API calls may not be sufficient to distinguish screenloggers from legitimate screenshot-taking applications. Indeed, by investigating existing screenloggers, it is possible to notice that they may exhibit various behaviours, including the fact that the screenshots frequency may be different from one screenlogger to another. The frequency can be a few seconds, few minutes, or configurable by the adversary. Screenshots can also be taken irregularly at each user event. Legitimate screenshot-taking applications are also diverse. Some of them, such as screen-sharing applications, need to take screenshots at a high frequency, while others like parental controls make take screenshots at a lower frequency, and others like screenshots editing applications may take screenshots occasionally. Therefore, relying only on API calls may lead to high false positives and false negatives rates on an extensive dataset containing different types of screenloggers as well as benign screenshot-taking applications. It was not mentioned whether the dataset used to test the method proposed by Javaheri et al. contained benign screenshot-taking programs, and there is no information about the nature and diversity of screenloggers [89]. Moreover, the authors perform the hooking on only two functions, namely GetWindowsDC and BitBlt, whereas there are other ways of taking

screenshots. For example, it is possible to use the GetDC function instead of GetWindowsDC in order to obtain the Device Context (DC).

Malware detection based on network monitoring Previous works on malware detection have shown that network traffic can be used for this purpose. Several ML approaches have been developed to characterise benign and malicious traffic based on a set of network traffic features. Initially, network-based detection approaches have been proposed for botnet detection [90, 91]. Subsequently, the approaches were extended to other malware types as most recent malware programs implement a communication module allowing the hacker to retrieve data and maintain remote control of the malware through a command and control server (C&C) [92].

The RFC 3697 (standard for IPv6 Flow Label Specification) defines a traffic flow as 'a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that the source desires to label as a flow'. Lashkari et al. defined a network flow as a sequence of packets with the same values for five attributes: Source IP, Destination IP, Source Port, Destination Port and Protocol [93]. TCP flows are usually terminated upon connection teardown (by FIN packets), while UDP flows are terminated by a flow timeout.

The flow-based network features from previous works in this domain have been used to detect different malware types such as botnet, adware or ransomware. The features are classified using the four categories defined by Lashkari et al.: byte-based (features based on byte counts), packet-based (features based on packets statistics), time-based (features depending on time) and behaviour-based (features representing specific flow behaviour). To these, we can add transport layer features and host-based features. Numerous features may not be used by the classification algorithm because they do not carry useful information (depending on the malware type).

Some of the existing research has developed models for general malware detection based on network traffic [92–94]. To characterise general malware traffic, Lashkari et al. proposed an Android malware detection model based on nine network features: flow packet length (min, max), backward variance data byte, number of packets with FIN, flow forward and backward bytes, maximum idle time, initial window forward and minimum segment size forward [93]. These features have been selected from an initial set of 17 by applying three feature selection algorithms. The proposed approach showed an average accuracy of 91.41% and a false-positive rate of 8.5%.

Boukhtouta et al. used two approaches: Deep packet inspection and IP packet headers classification [92]. Deep packet inspection is an advanced method for examining the content of packets and identifying the specific applications or services it comes from. It is a form of packet filtering that locates, identifies, classifies, reroutes or blocks packets with specific data that conventional packet filtering, which examines only packet headers, cannot detect. The evaluation results showed that it is possible to detect malicious traffic using J48 and Boosted J48 algorithms with 99% precision and less than 1% of false positives rate.

Nari and Ghorbani proposed a framework for the automated classification of malware samples based on network behaviour [94]. They considered different malware families, such as Bifrose, Ardamax, Mydoom or LDPinch, and identified network flows from traffic traces gathered during the execution of malware samples. Then they created a behaviour graph for each sample to represent the malware activity. The behavioural profile is built using dependencies between network flows in addition to the flow information. The classification features are extracted from the behaviour graphs. This includes, for example, the graph size (the number of nodes in the graph, which also shows the number of flows in the network trace of the malware), the root out-degree (which represents the number

of independent flows in the network trace of the malware), the maximum out-degree (which shows the maximum number of dependent flows in the network trace of the malware).

Other approaches focus on detecting specific malware types by analysing their network trace.

In another work Lashkari et al. proposed an initial set of 80 network-flow features to detect four Android malware categories: adware, ransomware, scareware and SMS malware [95]. After applying two feature selection algorithms, they kept the following features: forward packet length std, initial window forward and backward, segment size min, backward packet length min, backward packet length std, total backward packets and maximum and minimum inter-arrival time. The proposed approach showed an average precision of 85% for three classifiers, namely RF, KNN and DT.

The Canadian Institute for Cybersecurity offers two datasets intended for intrusion detection: CICIDS2017 [96] and CSE-CIC-IDS2018 [97]. The datasets cover six attack profiles: DoS/DDoS, SSH/FTP, Web attack, infiltration, bot and port scan. Note that the CSE-CIC-IDS2018 includes a botnet attack with a screenshot-taking and keylogging scenario [97]. For both datasets, 80 network traffic features have been extracted. Then, RandomForestRegressor was used to select the best feature set for each attack [96]. For instance, the best set of features for botnet detection consisted in: subflow forward bytes, total length forward packets, forward packet length mean and backward packets per second.

Saad et al. and Beigi et al. used features characterising the length of the flow along with other general features to detect peer-to-peer botnet traffic [98, 99]. To select the features, they assumed that botnet traffic generated by bots is more uniform than traffic generated by legitimate users showing highly diverse behaviour.

Note that screenloggers have received less attention from the researchers in comparison with other malware types, despite their impact on personal user data. Although malware programs share some common characteristics, they have different communication patterns with their C&C server. For instance, in P2P botnets, the bots frequently communicate with each other and send ‘keep alive’ messages. To detect screenloggers based on their network traffic, it may be necessary to use specific features related to their characteristics, such as the data type or the fact that their traffic is asymmetric. To the best of our knowledge, the only available screenloggers’ network traffic is related to the Zeus and Ares malware programs [96, 97] and are not representative of existing screenloggers, as they only consider periodic sending of 400 s. Screenloggers may exhibit various behaviours to remain stealthy, such as irregular sending or reducing the packet size by taking only part of the screen, lowering the resolution or using compression. Our experiments showed that a screenshot of a virtual keyboard may reach a size of 33KB by using low resolution and compression while still being exploitable by OCR tools.

Malware detection based on resource use Some works use the monitoring of resources consumption to detect malware, such as CPU or memory use.

Many works in the literature are based on the number and sequences of the file system or registry operations. Indeed, to perform their tasks, malicious programs need to make some operations on the file system and the registries, such as creating, deleting or altering files or registers.

An example of a method using those parameters was proposed by Mao et al. who used directed graphs composed of edges representing the dependency between the processes, the files and the registry entries [100]. Once the dependency graph is created, a PageRank-inspired algorithm is used to quantify the importance of each node. Then heuristic rules are used to detect the malware. The authors compared

their approach with three classifiers: KNN, linear regression and RF. Other works consider memory and registry operations, along with other features [82, 83].

A work using memory access is presented by Banin et al. [101]. In this work, the authors use an automated virtualised environment and Intel PIN - a dynamic binary instrumentation tool - to record memory access sequences produced by malicious and benign executables. The dynamic binary instrumentation tool can be used for live analysis of binary executables and facilitates analysis of different properties of the execution, such as memory activity or addressing space. The authors focused on the analysis of basic memory access operations (read and write) and their n-grams. They used several ML models to test their methods, such as KNN and Artificial Neural Network (ANN).

However, these works only focus on the number or the sequence of memory or file operations and do not consider other parameters, such as the size of the written files, for example, which may be necessary for screenlogger detection and differentiation from legitimate screenshot-taking applications.

Other works include CPU usage as a parameter to detect malicious programs. Some are anomaly-based approaches [102, 103]. Other detection methods are behaviour-based and monitor the resource use of each program separately [104–106]. For instance, Massarelli et al. propose to detect Android malware by monitoring a set of system-wide and application-specific metrics, including CPU, memory and network usage [105]. The proposed architecture automatically executes Android applications in a sandboxed environment. During each run, it collects resource consumption metrics from the /proc file and processes them through detrended fluctuation analysis and Pearson's correlation.

All these methods target mobile environments, and to the best of our knowledge, CPU usage was not employed to detect malware in desktop environments. Al-

though it may not be effective alone, in the screenlogger case, this could be useful along with other parameters such as screenshot API calls to help distinguish between screen-recording malware and legitimate screenshot-taking applications.

Combination of different categories of features Some methods propose to detect malware by combining different categories of features.

Weijie et al. proposed a framework that allows for the detection and classification of malware programs using several parameters [82]. The proposed framework is called MalInsight and consists of four stages: (1) data collection, (2) profiling based on the programs' API calls, file and registry access, and network use, (3) feature generation (number of occurrences of API calls, network operations and I/O operations) and (4) training of four ML models. The dataset used contains benign applications and five malware types: backdoor, constructor, email-worm, net-worm and trojan-downloader. The performance of each model is compared by combining the different features. This is done to show the impact of each feature on the performance of the models, which depends on the malware type.

Another example is Mohaisen et al., who used a data-mining approach to cluster and classify malware programs [83]. The process was executed on malware samples received daily from partners, particularly banking institutions. The approach uses a classifier trained on a manually labelled dataset. Malware samples were executed in a VMware virtual environment and analysed using Yara [107]. The extracted features are grouped as follows: (1) File system, which includes the number of created, modified, deleted files, file extensions, and the number of file in specifics folders;(2) registry : number of created, modified or deleted registry keys; (3) Network : number of connections on specific ports, protocol type, count of http requests GET, POST and HEAD, size of reply packet and count of specific http response codes, such as 200, 300, 400 and 500. Four ML models were used:

SVM, classification tree, KNN and Perceptron. The SVM model showed the best performance. The authors argued that their method showed a precision of 99% in classification and 98% in clustering.

However, considering too many features may lead to overfitting. Indeed, depending on the malware type, some features may be more relevant than others. Thus, adding too many unnecessary features may mislead the classifier. Weijie et al. tested their method on completely unknown malware samples having no link with their training dataset, and the results showed that for this malware type, considering only high-level features related to file, registry and network access is better than relying on a complete set of features including API calls, DLLs and static features [82].

Indeed, this is an important problem in malware detection methods, and many methods need to know the malware type in advance to perform accurate detection [108, 109]. This overfitting problem was formalised by Grosse et al. [110]. This paper presents an adversarial algorithm for attacking an ML detection model. Their goal is to show how malware developers can adapt their strategies to evade detection. The proposed algorithm consists of altering the input feature vector to mislead the detector. This is done without causing the application to lose its malicious features. The authors designed an ANN model for malware detection, which was trained using a public malware dataset [111]. This model was used in an attack scenario, and it was misled by 63% of malware.

Urooj et al. [112] made a study about the machine learning techniques proposed in the literature during the last three years to detect ransomware. All these methods are based on behavioral detection with dynamic features. This study shows that many recent techniques use deep learning algorithms (ANN), alone or combined with traditional ML, to detect ransomware. However, to be efficient, this

kind of technique needs a huge amount of data (thousands or millions of samples). An example is the work of Kimmel et al. [113], which proposes recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure. They use 40,680 malicious and benign samples of general malware. However, there is only a little data about screenloggers with approximately a hundred of samples, making these techniques not applicable.

2.2.4 Synthesis

Signature-based methods give good results for known malware programs with a low false-positive rate. However, they are vulnerable to obfuscation techniques that are more and used by modern malware. Moreover, signature-based methods cannot detect malware integrating polymorphism and metamorphism mechanisms because the signature of the malware changes every time a machine is infected.

The drawback of anomaly-based detection techniques for detecting malware in computer systems is that those systems contain and execute many processes having many possible behaviours. This makes it difficult to define a normal behaviour to model in detection methods and can result in a high false-positive rate.

Static behaviour-based detection fails to overcome obfuscation techniques. Malware designers can use those techniques to disturb the analysis process and hide the malicious behaviour of the program.

Dynamic analysis behaviour-based detection can overcome obfuscation techniques, as it analyses the runtime behaviour of the malware. Therefore, this technique is widely used in recent malware detection works, which consider diverse types of dynamic features. However, it was shown that the performance of a detection model greatly depends on the dataset it was trained on. Indeed, the features selected as the most discriminative vary according to the malware type. Consid-

ering many features for general malware detection is not ideal either as it makes it easier for malware programs to pretend to be legitimate by taking advantage of overfitting.

Screenloggers can be a particularly evasive category of spyware due to the screenshot functionality being a legitimate one widely used by legitimate applications. However, only one work mentions screenloggers and several limitations to its methodology were discussed [89]. More precisely, the approach was limited to the API call feature, the API calls traced to detect screenshots were not exhaustive, the behaviours of screenlogger samples were not diverse, and they did not mention whether legitimate screenshot-taking applications were included. For all these reasons, it is necessary to propose a tailored model using features specific to screenloggers and able to distinguish them from legitimate screenshot-taking applications.

Figure 2.1 sums up why existing malware detection approaches are not adapted to the screenlogger case. There is a need to design a behaviour-based approach with dynamic features specific to screenloggers (API calls and network monitoring).

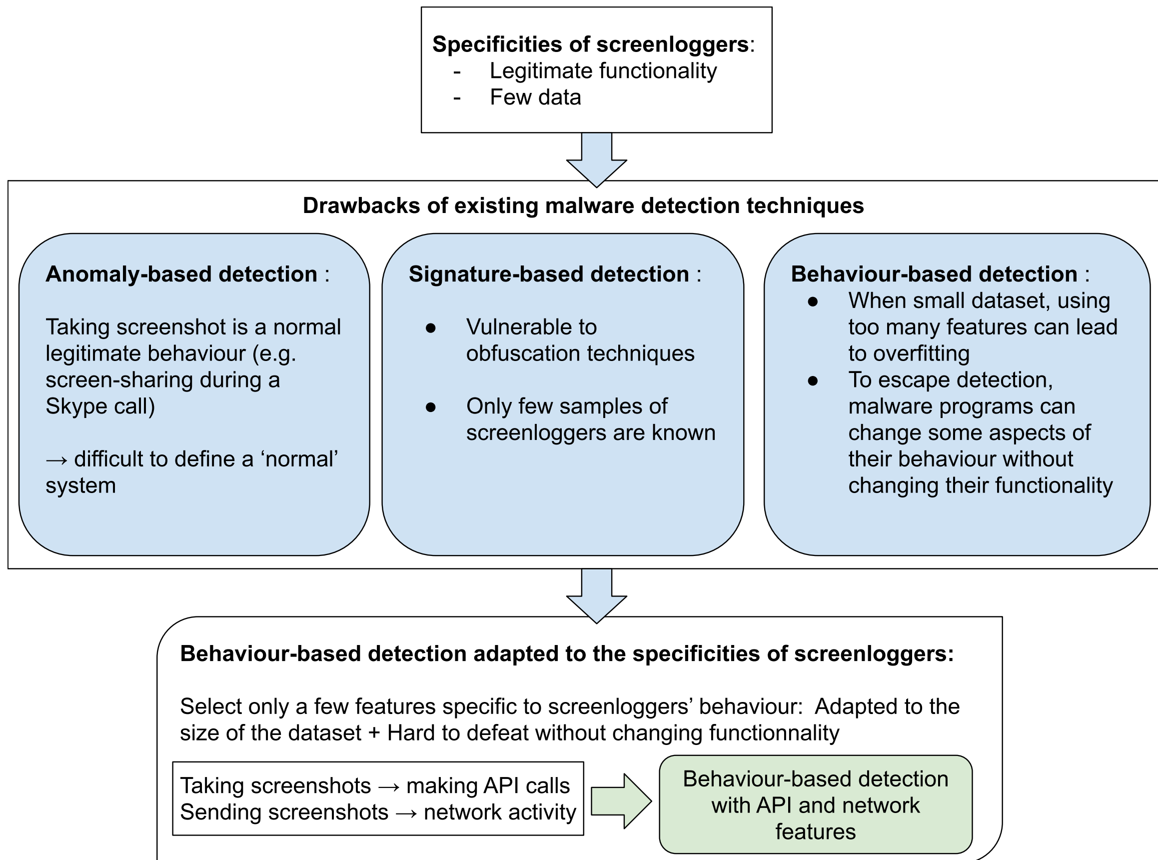


Figure 2.1: Drawbacks of existing detection methods for screenloggers

2.3 Screenlogging prevention

This section gives an overview of previous works aiming at protecting the content of the screen from attackers. These works are focused primarily on securing the act of authentication, but some of them also aim at general screen protection.

2.3.1 Screenlogger prevention: the authentication case

Most works aiming at protecting the screen content during authentication cover the case of shoulder-surfing attacks from an external observer. However, some of

these techniques are applicable to screen-recording attacks. A more limited number of works focus on tackling the screenlogger threat during the authentication process.

Protection against shoulder surfing

Shoulder surfing is an observation-based attack where a malicious observer sees content on the victims' screen without their authorisation by observing it directly or using cameras. This is a well-known problem in the literature, and many works aim at addressing this threat, especially in the authentication case. This issue is closely related to the threat of screenshot-taking malware because in the two cases, an adversary sees the victim's screen without authorisation; the difference lies in the method used to observe the screen. Therefore, the anti-shoulder surfing methods for secure authentication are discussed below.

Obfuscation and confusion Some techniques in the shoulder-surfing literature have attempted to remedy the shortcomings of password-based authentication by using obfuscation and confusion of the observer (Figure 2.2). As explained below, these methods are not effective against screen-recording attacks.

(a) [114]

(b) [115]

(c) [116]

Figure 2.2: Examples of anti-shoulder surfing authentication techniques using obfuscation and confusion.

One proposed solution consists of adding artefacts (noise) on the screen when a click occurs [117]. For example, it can be the display of an artificial mouse pointer to prevent the malware from knowing which part of the screen has been clicked. This approach would not stand up to malware recording the click coordinates.

Several works propose using virtual keyboards in which keys are mixed or hidden. Agarawal et al. suggested a dynamic virtual keyboard that mixes key layouts after each click and hides them at click events (figure 2.2 (a)) [114]. A colour code is used to easily remember character positions. The user can enter one character at a time. The position of the character to be typed is noted, and the user clicks on the 'hide keys' button to hide all characters. Finally, the position of the desired key is located using its colour code.

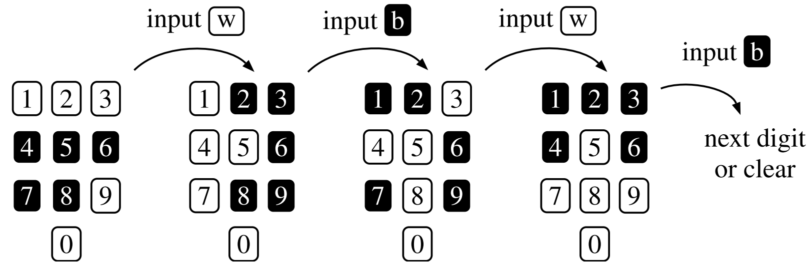
The keyboard proposed by Srinivasan et al. consists of 72 keys that are logically divided into four groups – A, B, C and D, each comprising 18 keys [118]. When the keyboard is shown to the user, the keys' positions are randomised. The user must locate and remember the current position of the desired key. Each key is indexed using its position and its group ID. Thus, the user needs to note down the index value and group ID corresponding to the required key (for example, A2, where A is the group ID and 2 is the index of the required key). At the next step, the user shuffles the keyboard keys and hides their labels so that only their indexes appear. After a key is entered, the keyboard is randomised and switched to visible mode. This process continues until the user enters all the password characters and chooses to submit.

A similar approach was proposed by Parekh et al., with a virtual keyboard that changes its appearance and disposition over time [115]. When the user clicks on a key, all the keys of the keyboard are transformed into asterisks, and the keyboard layout is changed randomly after each click (Figure 2.2 (b)).

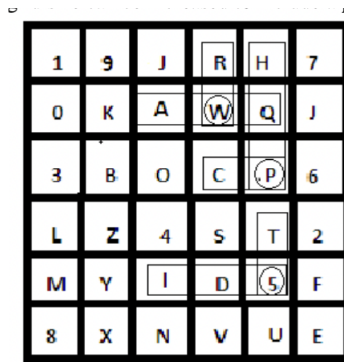
All these techniques have in common that they rely on the attacker's short-term memory because they cannot remember the position of each character between each click as the layout is mixed and the keys are hidden at the click events. The technique is effective against shoulder-surfing attacks performed by a human observer, but it is not the case when the process is recorded using a camera. Moreover, it is effective against screenloggers that take screenshots at each click but not against those taking screenshots at regular and short intervals.

Cognitive tasks Other techniques have attempted to make games out of the authentication procedure without needing to hide any information (Figure 2.3). These techniques use cognitive tasks to increase the difficulty of the login session.

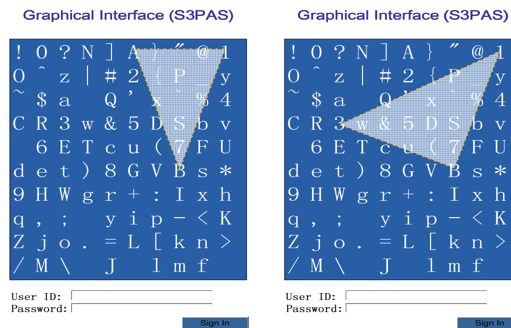
However, the use of cognitive challenges is not always efficient against screen-recording attacks depending on the proposed scheme.



(a) [119]



(b) [120]



(a) pass-triangle △A1B (b) pass-triangle △1B3

(c) [121]

Figure 2.3: Examples of anti-shoulder surfing authentication techniques using cognitive tasks.

Roth et al. presented an alternative PIN entry method called 'the cognitive trap-door game' to prevent shoulder surfing [119]. The main idea is to consecutively display the set of PIN digits as two partitions. Instead of clicking on the targeted digit, the user must indicate the partition to which it belongs. This process is repeated four times for each of the four PIN digits. Roth et al. proposed a similar approach in another work [122]. These approaches make use of the limited human visual short-term memory to counter shoulder surfing, because a human observer cannot remember the combination of the 16 partitions entered by the user with the

corresponding screen for each one. However, by taking screenshots, it is possible to deduce which digits correspond to each 4-partition combination.

Similarly, the approach described by Nand et al. is not resistant to screenlogger attacks [123]. In the paper, the researchers proposed the 'PassBoard' approach, relying on a dynamic grid of characters. The user must perform two button clicks to get the desired character. The first and the second click can be performed on any button belonging to the same row and column, respectively, as the desired character. Therefore, the user can enter their password without pressing any of the characters that occur in it. However, using screenshots, it is easy to recover the password characters by looking at the intersection of rows and columns corresponding to the clicked characters.

Other approaches are more difficult to break, even with screenshots, but may be subject to brute-force attacks. This is the case of the approach proposed by Surjushe et al., which aims to bolster the security level of virtual keyboards against shoulder surfing without hiding characters [124]. During the registration process, the user specifies an email address where a Factor of Displacement (FD) and a traversal direction are received. FD is a random number between one and five specifying the number of shifts from one key to another. The traversal direction indicates the shift direction. When logging on, for each character of the password, the user must calculate the new character to type by adding the number of shifts FD. For example, if the original character of the password is 'q', with FD equal to one and the traversal direction horizontal right, then the new character to be typed is 'w'. The weakness of this approach lies in the fact that the adversary can try all the combinations of FD and directions, which makes only 20 possibilities.

Some works propose password schemes that are resistant both to shoulder surfing and screen-recording attacks. Deulgaonkar et al. propose a pair-based scheme

[120]. A session password is built by processing the password characters in pairs. Each character of a pair is located on a grid, and their intersection is part of the session password. The latter changes for each session because of the changing of the characters' grid layout. Another example of a screen-recording resistant approach is presented by Zhao et al., who propose an authentication scheme called S3PAS [121]. An image containing letters, numbers and special characters placed in disorder is presented to the user, who has to enter a password by clicking on the image following a specific click-rule relying on 'pass-triangles'. For instance, if the password is 1234567, then the first click must be inside the pass-triangle formed by 1, 2 and 3 and the second click must be inside the pass-triangle formed by 2, 3 and 4, and the process is repeated until the password completion. These authentication techniques are resistant both to shoulder surfing and screen-recording attacks. Indeed, even if the observer has recorded the screen and can see all the actions performed by the victim, the number of different possibilities is too high, and it is impossible to deduce anything. However, it makes the login process longer and more complicated. Moreover, these protection mechanisms are specific to the authentication process, which is an active process where the user has to perform a succession of actions, but it is not generally extendable to protect any information displayed on the screen.

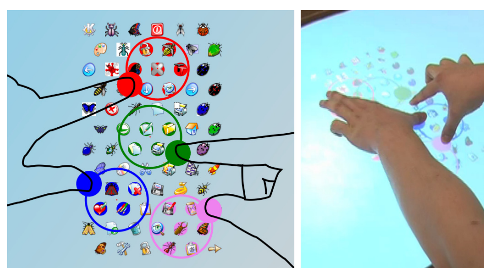
Graphical authentication Another technique to protect authentication against shoulder surfing is the use of graphical passwords. Such techniques replace the alpha-numeric password with the use of a series of images, shapes and colours (Figure 2.4).

However, these techniques are not necessarily resistant to screenlogging attacks. The authentication scheme proposed by Wiedenbeck et al. serves as an example [125]. Authentication is performed by clicking on a set of predefined pixels on

a chosen picture. The user picks up several points (3 to 5) on an image during the password creation phase and re-enters each of these preselected click-points within their tolerance square in the correct order during the login phase (Figure 2.4 (a)). The image helps the user to recognise the set password. The locations where the user clicks may be difficult to recover for a classic shoulder surfer, but if the screen is recorded during the login process, an adversary could recover the password by looking at the screenshots. A similar example is the solution proposed by Sun et al., which relies on a graphical authentication system named PassMatrix (Figure 2.4 (b)) [126]. Users choose one square per image for a sequence of n images rather than n squares in one image. such as in the case of Wiedenbeck et al.

(a) [125]

(b) [126]



(c) [127]

Figure 2.4: Examples of anti-shoulder surfing authentication techniques using graphical passwords.

A different visual authentication scheme is proposed by Kim et al. [127]. The login process is based on a set of icons, called key icons, affected to one single colouring: red, green, blue or pink (Figure 2.4 (c)). On the login interface, the user is presented with several grids of randomly dispersed icons. For each displayed grid, they must lasso the key icons with the correctly coloured ring. The authentication is allowed in the case where the user lassos the key icons of each grid with the intended colour rings. The size of the colour rings is adjusted so that it contains seven icons. However, this technique is not effective against screen recording because it may take an observer only a few login sessions to deduce the key icons with the associated colours (by intersecting the sets of lassoed icons for each colour on each grid), especially given that the users may tend to place the key icons at the centre of the colour rings.

Gesture-based authentication Using this method, the user is able to draw a picture to authenticate, or 'connect-the-dots' in a grid in a specific order or pattern (Figure 2.5). This method was primarily developed for convenience since password entry using small screened mobile device keyboards can be difficult.

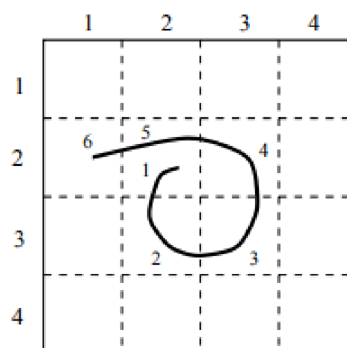


Figure 2.5: Example of gesture-based authentication [2].

Jermyn et al. proposed a scheme for a graphical input display that allows for drawing any shape or any character as a password (Figure 2.5) [2]. The pro-

posed technique is intended to increase the difficulty of stealing a password for a shoulder surfer. The user draws their password over a grid divided into cells. The entered shape is mapped into cell coordinates in the order which the stylus passes through them. The mapped coordinates are compared to those entered during signing up to allow or deny the authentication. However, this approach is not resistant to screen recording as only two screenshots are sufficient to know the shape and the starting point.

This scheme was extended by Martinez et al. [128]. The proposed authentication system is based on behavioural biometric characteristics which are extracted from the dynamics of the drawing process such as speed and acceleration. In other words, the attacker would have to imitate not only what the user draws, but also how the user draws it. In this case, it is more difficult to impersonate the user even with screen recording. However, if screenshots are taken at a sufficiently high frequency, it is still possible to deduce the speed at which the form was drawn. It may also be possible for a malware to use the phone's sensors such as the accelerometer. Moreover, this technique is specific to the authentication process and is not extendable to protect any information displayed on the screen.

Hardware-based authentication Some techniques propose to enter the password using other hardware components instead of limiting the process to the traditional keyboard and screen of the user's device. For example, Bianchi et al. proposed a tactile cues-based method to prevent shoulder surfing during login (Figure 2.6) [3]. The system is implemented using special hardware keys that allow encoding passwords as a sequence of vibration patterns rather than characters or images. The same authors proposed a PIN entry based on audio or haptic cues [129]. These techniques are effective against screen recording, but the draw-

back is that special hardware is needed. Moreover, the technique is specific to the authentication process and cannot be used for general screen content protection.

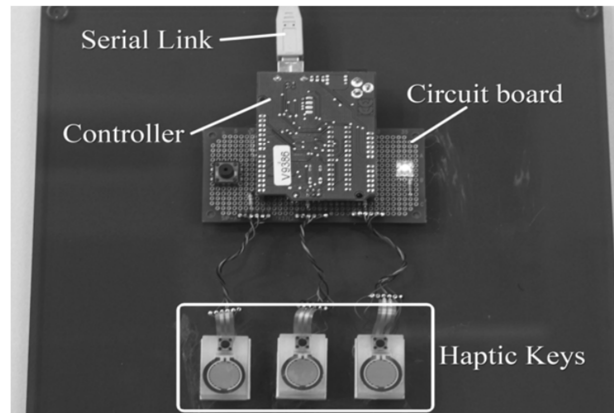


Figure 2.6: Secure Haptic Keypad [3].

Luca et al. presented an authentication mechanism named XSide that uses the front and the back of smartphones to enter stroke-based passwords (Figure 2.7) [4]. Users can switch sides during input to minimise the risk of shoulder surfing. The authors explored how switching sides during authentication affects usability and the security of the system. This technique is specific to smartphone devices and is also limited to the authentication process.

Figure 2.7: XSide system [4].

Protection against screenloggers

A few works proposed methods based on human vision features to prevent screenloggers from stealing credentials during authentication. Lim et al. proposed a solution based on retinal persistence (Figure 2.8) [5]. The goal of this technique is to divide each character into segments and display them one after the other in a quick succession. At a sufficiently high speed, a human can see the whole character. On a screenshot, the numbers are never fully visible. However, this solution is limited to digital numbers composed of a limited number of segments. Also, an adversary can deduce possible characters from one image (knowing that one segment is on may greatly reduce the set of possible digits).

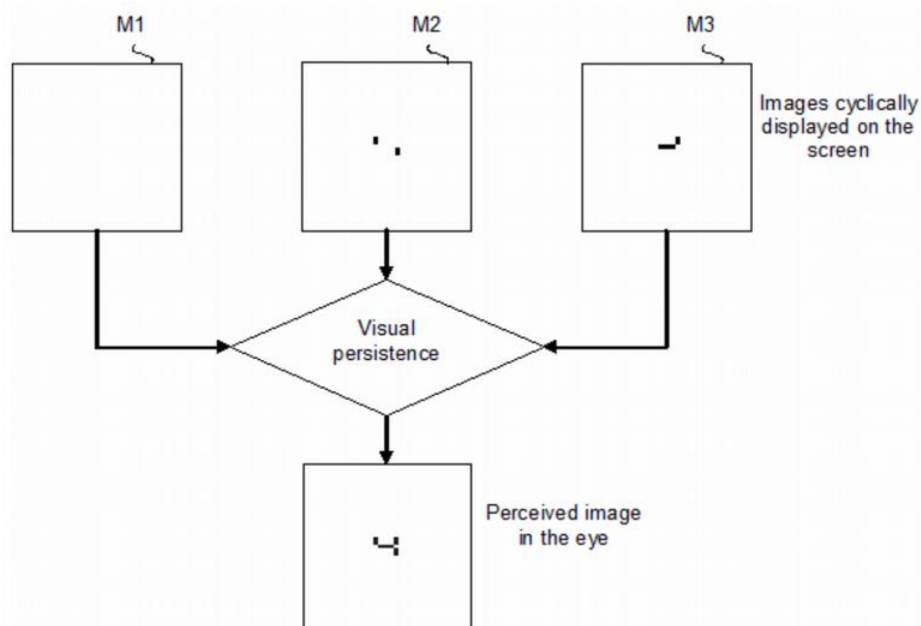


Figure 2.8: Formation of the number ‘4’ using visual persistence [5].

Echalier et al. proposed a new technique to create virtual keyboards based on the Gestalt psychology [49]. The proposed approach uses the law of common fate, which states that elements moving in the same directions are perceived as

belonging to the same form. The idea is to divide the keyboard into tiles that contain either a random or a pre-calculated texture. This technique stimulates the brain to detect the pre-calculated tiles on which the same rotation is applied as whole shapes while disregarding the random textures as background. In doing so, a human sees a traditional keyboard while a program only detects noise. This method can also allow for the prevention of screenshot exploitation by a human: motion perception allows a human to distinguish a given form through motion. When a screenshot is taken, however, there is no motion, and even a human cannot read the content. This idea is extended by Bacara et al. [48], who designed a virtual keyboard using motion perception and two other human vision properties : visual assimilation that allows a human to perceive the apparent brightness of an object depending on the contrast between the object and its surroundings, and visual interpolation, which allows a human to perceive the complete shape of an object even with missing parts [130]. Although these works afford adequate protection against screenloggers during authentication, it may be difficult to use them for general content protection because the user's visual comfort is significantly affected due to the techniques used, resulting in greyscale and noisy images. Another obstacle for the adoption of this method as a general solution against screenshots is that it is content-dependant and requires computations to be performed on each pixel, which is time- and resource- consuming. Moreover, these methods are limited to textual content protection.

A different approach is proposed by Nyang et al. [6]. The authors provide a secure authentication protocol that overcomes several attacks, including keyloggers and screenloggers, by relying on the use of two devices (a smartphone and a computer, for example). A blank virtual keyboard is displayed on the first screen, and after scanning a QR code, randomly dispersed keyboard keys are shown on the second screen. The user enters the password using a mouse pointer on the blank

virtual keyboard of the computer while seeing the arrangement of keys through the smartphone (Figure 2.9). However, this technique requires having two devices and cannot be used for general screen content protection.

Figure 2.9: Authentication system proposed in [6].

2.3.2 Screenlogger prevention: the general case

Although most works focus on countering observation-based attacks during authentication, some aim at protecting the screen's content from any privacy leak in a more general way. In this context, shoulder surfing is again an important preoccupation, but several studies also propose specific solutions to protect private content from screenshots. Works targeting the prevention of automatic character recognition are also interesting to mention in this context.

Shoulder surfing

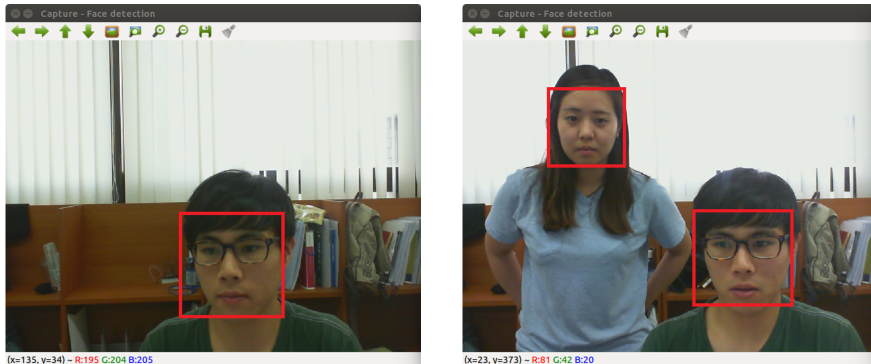
Offering general protection of the screen content against shoulder surfing is a challenging issue that is the object of several recent works in the literature.

A major difficulty of finding general solutions against shoulder surfing is that screen protection cannot be applied permanently and on the whole screen as it would otherwise prevent the users from utilising their devices. Therefore, some solutions propose protection mechanisms based on face detection to detect the presence of shoulder surfers.

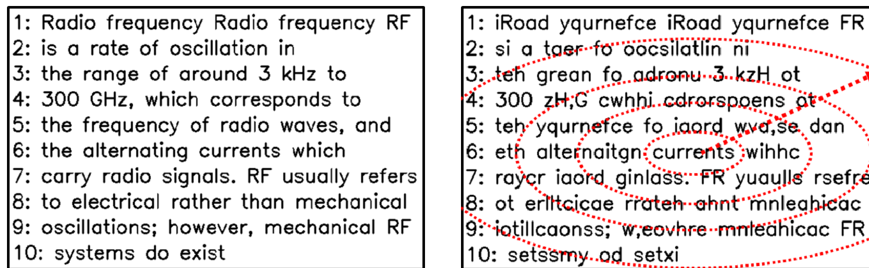
This is the case of Brudy et al., who proposed a solution to address the shoulder-surfing threat in public spaces for large displays (fixed device) [131]. The authors exploited the spatial relationships between the user, the shoulder surfer and the screen. Their solution warns the user about the presence of a shoulder surfer by flashing the border of the screen and providing a 3D model in which the position and gaze of the surfer are indicated. Then, the user can hide or collect the personal window together on one side of the screen. Moreover, the position of the user's head against the display and the shoulder surfer is estimated to darken the regions visible to the shoulder surfer and leave the shielded display area unaltered. However, this technique presents several drawbacks, such as its inapplicability to cases where it is a camera instead of a human recording the content of the screen, or in the case of a screenlogger. Moreover, this system may disturb the user in the case of false positives, where faces are detected from persons who are not shoulder surfers looking at the screen or persons whom the user wants to show information.

Another technique uses face detection and thus suffers from the same drawbacks [7]. The authors proposed the Shuffling Texts Method (STM) – a method aiming to prevent shoulder-surfing attacks. The STM displays shuffled texts to the malicious shoulder surfers. In plain mode, when only the user is reading the document, the shuffling level is 0. As soon as an intruder appears, STM provides shuffled texts with a higher shuffling level (Figure 2.10). As the solution proposed by Brudy et al., this method, which relies on face detection, is not applicable to

the case of an adversary using a camera to film the content of the screen nor to screenloggers. Moreover, the usability is affected since to allow reading, users must move the cursor while reading to indicate which zones must not be shuffled.



(a) Shoulder surfer detection using face recognition



(b) Displaying shuffled text

Figure 2.10: Shuffling text method [7].

Another solution, proposed by Khamis et al., is to use gaze-tracking [132]. The authors proposed to display textual content on smartphones by tracking the user's gaze while hiding the rest of the screen using different masks (blackout, crystallise or using a fake text). However, this solution is not effective against screen-recording attacks.

Other solutions give an active role to the users or let them choose what content they want to protect.

Eiband et al. proposed replacing texts with the user's handwriting to protect textual content from shoulder-surfing attacks [133]. This method assumes that the shoulder surfers are slower when reading the unfamiliar handwriting of other persons. The proposed scheme is split into two steps in which the user's handwriting is collected word-by-word then combined into original sentences of textual content. However, this method presents several drawbacks, including usability and that it is not necessarily effective against screen-recording attacks because modern OCR are able to recognise handwritten content.

Mitchell et al. offered a method to replace sensitive information with less meaningful data to combat shoulder surfing [8]. As illustrated in Figure 2.11, the user must define a list of sensitive words that will be replaced by aliases. In that users must choose the data they want to protect, the scope of the proposed approach is significantly reduced because most users are not security aware and would not do it. Moreover, the approach is quite intrusive because users cannot see the protected data.

SAMPLE MAPPING OF SENSITIVE DATA TO CASHTAG ALIASES

Type	Actual	Alias
Name	John Smith	$name
Email	jsmith@gmail.com	$email
Username	Jsmith1	$user
Password	p@ssw0rd	$pass
Street Address	123 Main St.	$address
Phone number	555-111-2222	$phone
Birthday	1/1/85	$bday
SSN	111-22-3333	$ssn
Credit Card	4321 5678 9012 1234	$visa
Account number	123456789	$accts

Figure 2.11: Sample mappings of sensitive private data elements to their corresponding Cashtag alias [8].

For the protection of graphical content, Zezschwitz et al. proposed the application of distortions to images displayed on smartphones to protect them from shoulder

surfers, as illustrated in Figure 2.12 [9]. The scheme allows the smartphone owner, who has already seen the picture, to recognise the image, in contrast to an onlooker who has not seen it before. However, as the quality of the images is significantly affected, the usability of this approach is minimal, and the solution cannot be widely applied.

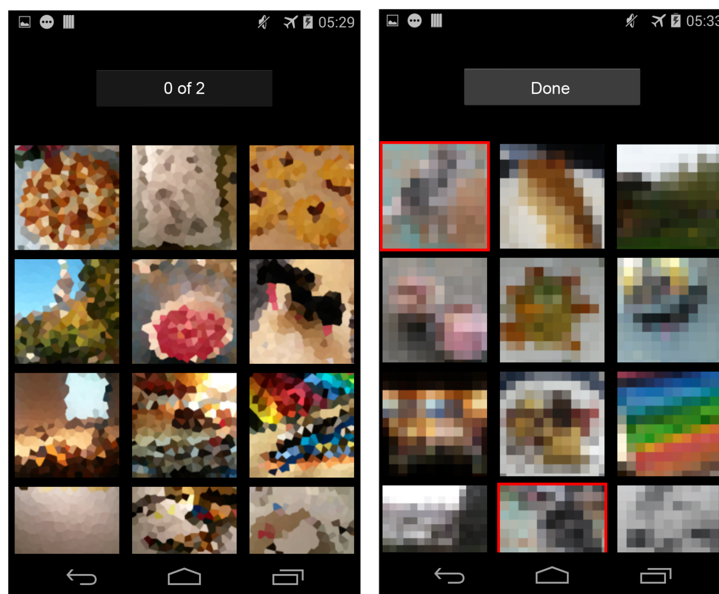


Figure 2.12: Filters applied to photos against shoulder surfing [9].

To conclude, some of the proposed solutions against shoulder surfing are not effective against screen-recording attacks because they rely on spatial information such as face recognition or gaze-tracking. Other solutions use subjective data from the users, such as using their handwriting or letting them choose which data they want to protect or completely hide. These approaches are, however, highly intrusive and cannot be adopted widely.

These drawbacks stem from the general difficulties of preventing shoulder surfing, which is quite a challenging problem. An attacker next to a user seeing the same screen must be prevented from acquiring sensitive data, while users must be able

to utilise their devices and read the content of their screens. The problem is quite different in the case of screen-recording spyware, and other solutions have been proposed to cover this case.

Anti-OCR techniques

Addressing the screenlogger threat implies taking steps to prevent exploitation, that is, preventing adversaries from extracting data from screenshots once they are taken. In this context, anti-OCR techniques may be used to prevent automatic screenshot exploitation.

During the last decades, OCR techniques have rapidly evolved. OCR systems take as inputs images containing text and produce a textual representation of the extracted characters. The recognition process is composed of two stages. The segmentation is done in the first stage: features are extracted from the images, and a series of separated individual character is extracted accordingly. The second stage is the recognition process: characters are recognised using ML, and their textual representations are produced.

Usually, OCR algorithms are legitimately used for digitising and making searchable typed or handwritten text from scanned images. In some cases, however, they are used in online bots to bypass a text-based Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA). A CAPTCHA is used to protect online services from attacks, such as denial of service and spamming, and allows only human interactions. To prevent OCR algorithms from recognising the content of text-based CAPTCHAs, researchers have proposed different noising techniques. Each of them works on specific stages and attempts to introduce noise automatically.

Anti-segmentation techniques allow for the production of images with text, where the position, size and other characteristics are intentionally chosen to prevent segmentation. In contrast, anti-recognition techniques are used to mislead the recognition process. In this section, we review recent studies proposing techniques to prevent OCR.

Bursztein et al. presented a new text-based CAPTCHA scheme [134]. The aim was to design a user-friendly scheme that minimises user error. The authors used three categories of features: visual features, anti-segmentation features and anti-recognition features. Visual features describe the visual aspect of the generated image, which influences users' errors. The anti-segmentation features include character overlaps, random dot size, random dot count, type of lines, line positions, count and width and similar foreground and background colours. Anti-recognition features encompass rotated character count, rotation degree, vertical character shifting, character size variation and character distortions. The impact of each of these features was evaluated with users and quantified according to solving time and error.

Roshanbin et al. proposed ADAMS, a text-based CAPTCHA generating algorithm [135]. The authors began by citing common text-based CAPTCHA vulnerabilities: (1) small input set due to the limited number of characters on physical keyboards, (2) non-randomness in the presentation, or limited combinations of characters and (3) lack of proper degradations. The authors then detailed the design of their solution, which attempts to remedy the cited vulnerabilities. They proposed the use of UNICODE as an input space, and characters from different languages were used. This method disturbs the recognition, as most OCR software is trained to recognise a small set of inputs, or only inputs from a specific keyboard where the number of keys is already known. The use of UNICODE characters requires the use of a virtual keyboard. Another technique to influence

recognition is the use of similar objects, which are shapes intentionally introduced into images to mislead the recognition. Humans can recognise them as shapes, while OCR recognises them as characters since they are segmented as characters. During the recognition stage, the OCR returns a false classification. ADAMS uses a randomised colour palette in both the foreground and background to disturb segmentation, as segmentation algorithms are highly influenced by the use of colours. The proposed solution has been validated using different segmentation and recognition methods.

Tangmanee et al. presented a study evaluating the impact of three characteristics of text-based CAPTCHA: character rotation, format and the length of the text [136]. The text format describes the use of handwritten or typed characters. Each parameter has been used to generate images with different angles of rotation, formats and text lengths. The generated images are processed with a selected state-of-the-art OCR system. The experiment demonstrated that all parameters have a significant statistical impact on the robustness rate [136]. However, this conclusion is specific for the used OCR algorithm and needs to be validated for other algorithms.

Even if new CAPTCHA techniques are proposed, OCR evolves rapidly, and it has been demonstrated that many anti-OCR mechanisms can be broken using specific techniques. Bursztein et al. offer a review of the weaknesses and strengths of current text-based CAPTCHA systems [137]. Table 2.1 provides an overview of their findings. Moreover, Bursztein et al. present an algorithm for enhancing the breaking of recently used text-based CAPTCHAs [138]. Authors start by describing an anti-segmentation technique called negative kerning, which collapses characters together to prevent segmentation. The algorithm shown could break a dataset of currently used CAPTCHAs using this technique. First, the algorithm segments

the images to all possible cuts, and then applies a recognition algorithm to each of them based on KNN to identify the most-correct cuts.

Table 2.1: Anti-segmentation and anti-recognition techniques and their weaknesses.

Techniques	Category	Weakness
Complex background: the use many colours or images in the background	Anti-segmentation, background confusion	Algorithms can remove all colours not used in characters
Colour similarity: use of a gradient of colour for background and foreground	Anti-segmentation, background confusion	Use of hue, saturation and lightness colour representation.
Noising the background: using noises with the same colours as the text	Anti-segmentation, background confusion	Algorithms can remove pixels using the Gibs method [139], where pixels are removed according to a quantified parameter computed using their 8 neighbours.
Small lines: use of lines passing through the text	Anti-segmentation	Histogram of gradient-based method can detect intensity, and lines could be removed.
Big lines: use of big lines passing through the text	Anti-segmentation	In some cases, a line-finding algorithm could detect the lines.
Collapsing: removing space between characters	Anti-segmentation	In case where the length of the text is known, the character collapsing could be predicted. In the case where the length and size of characters are unpredictable, the segmentation becomes hard; thus, direct recognition is employed.
Enlarge input space	Anti-recognition	-
Distortion	Anti-recognition	-
Character rotation	Anti-recognition	Must include anti-segmentation otherwise it is ineffective.

Therefore, CAPTCHA-based techniques present several weaknesses and may be broken by sophisticated OCR techniques. Moreover, it would be difficult to use them in the context of screenlogger prevention due to usability concerns, as users may not want all the content of their screen to be displayed in a CAPTCHA for-

mat, which is difficult to read. A solution may be to limit the use of CAPTCHA to sensitive data; however, the issue would then be to define what must be considered as sensitive data, requiring either the intervention of the users (reducing the scope of the solution) or using automatic techniques such as Natural Language Processing (NLP). It may be the case that an entire text is considered sensitive for a user. Finally, the use of CAPTCHA-only prevents automatic screenshot exploitation but is not applicable in the case where the human adversary directly inspects the screenshots.

Screenshot prevention

Several solutions have been proposed to protect the user's data from malicious screenlogging.

Commercial solutions A first set of commercial solutions detect when a running process invokes an API call to perform a screenshot and take action to protect the screen content. Some tools, such as [140], show a notification to the user asking permission to allow or deny the operation when detecting an application taking a screenshot. However, asking for the permission of the user is not fully secure since they may, in some conditions, be misled by the malware disguising as a legitimate application. Moreover, this solution is ineffective against a malicious insider who might install the screenlogger and allow it to take screenshots. Another weakness is that these tools do not detect all screenlogging operations, as demonstrated in [141], where some anti-spyware tools were tested against several screenshot-taking malware types and failed in some cases.

Similar solutions blacken the screen when detecting a screenshot operation by putting a black window in the foreground [142]. These tools rely on the user activating them when required, for example, when looking at sensitive information.

Other solutions allow the user to open sensitive documents through an application that prevents screenshots [143].

All these solutions have in common that they are not intended for the overall protection of the system but rather target specific applications or files, often chosen by the user. This assumes that users are security aware to a certain extent, whereas most of them have not been made aware of the threat, do not have enough skills or may be negligent or mislead by malicious attackers. This general lack of security awareness is demonstrated in several surveys which reveal, for example, that more than half of respondents in Germany and Italy did not know anything about ransomware [143]. Therefore, solutions requiring an active role from the users are quite limited in scope and effectiveness.

Note that some operating systems, such as Windows, allow the application developers to forbid other programs to take screenshots of their application by using a specific flag, SetWindowDisplayAffinity on Windows devices [144]. However, this solution relies on the fact that developers will think of activating this flag for their sensitive applications, which is not a safe assumption. Moreover, some applications are so wide that developers cannot know in advance if the information displayed will be sensitive or not. Such is the case of Microsoft Office, which can be used to display benign content as well as sensitive, confidential documents.

Manmohan et al. propose another solution [145]. When detecting a screenshot request, the method looks for files associated with the displayed application windows and determines whether they contain confidential information. If this is the case, some actions are taken, such as preventing the screenshot operation or notifying the network administrator. However, this solution targets the specific context of corporate networks. Defining and determining what information is confidential in a general case may be difficult, as it may be subjective. For example, intellec-

tual property data may look like normal data, whereas it is highly sensitive data in industrial contexts. Users may define what information they want to protect, but it cannot be a widely adopted solution because, as already explained, most users are not security aware. Moreover, this solution is rather extreme because it prevents users from taking screenshots containing sensitive information even if they would legitimately want to do so.

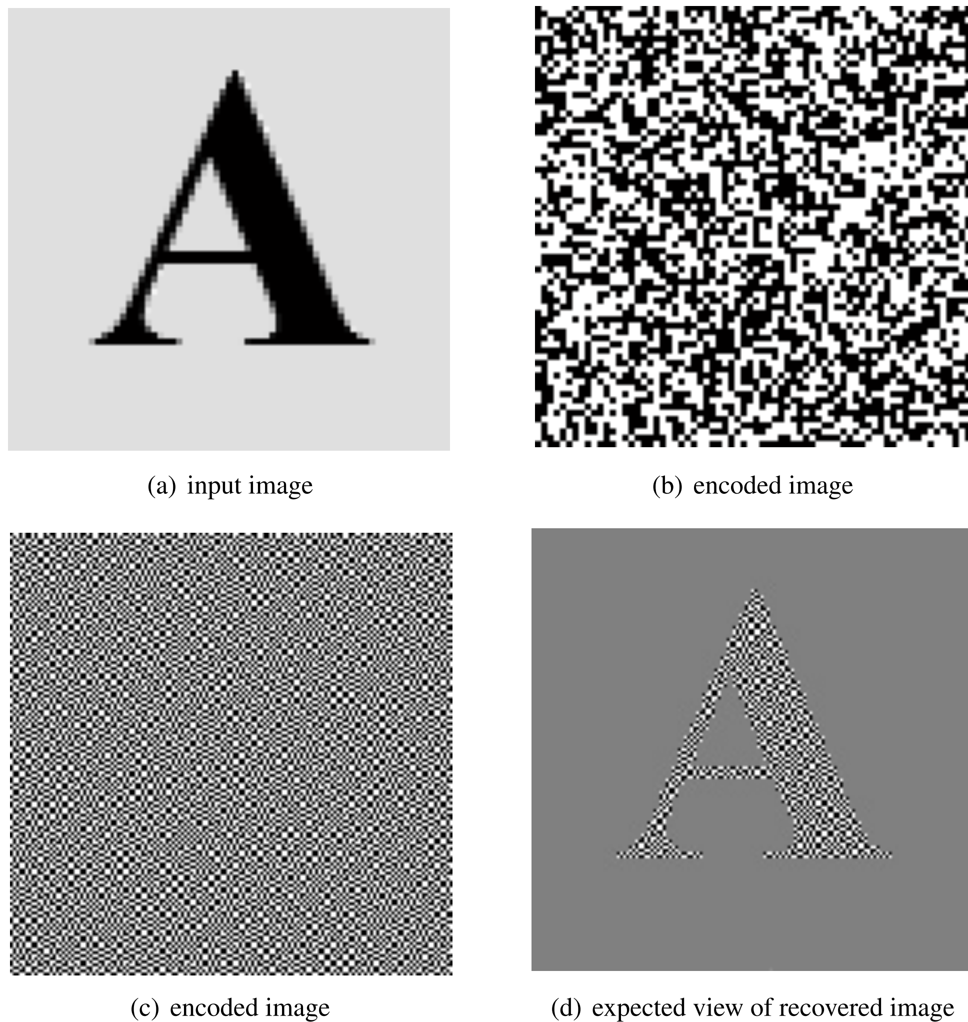
Theoretical solutions using the HVS Different solutions are proposed in the literature. These solutions, instead of forbidding screenshot operations, propose specific methods based on properties of the HVS to display information so that it is visible for users looking at their screen, whereas an isolated screenshot does not provide any meaningful information.

The HVS is a combination of two main parts. The first is the perception part, which involves the eye and its components, and the second is the visual cortex, where perception is processed [146]. The optics nerves are responsible for connecting the two parts [147]. The HVS is a powerful system as it can perform many image-processing tasks, many of which are too complex or even impossible to perform on computers.

For this reason, HVS properties can be used to distinguish machines from humans. We can cite, for example, the law of pragnanz [148], according to which the HVS interprets images in the simplest possible way. Thanks to this property, the use of lines passing through the text does not disturb the human viewer, as opposed to automatic tools, which may make errors in the segmentation step.

Hou et al. achieve protection through a visual cryptography algorithm that divides the target images into several frames [10]. The technique consists of encrypting successive N images by affecting new values for each pixel of each image. In

addition to the pixel value in the previous encrypted image, the new value of the pixel is affected by whether it belongs to the foreground or the background of the original image. Pixels in the foreground are modified across the N images while pixels in the background are not. The decryption step is done by sequentially displaying the encrypted images during a calculated duration. This allows the human eye to recover the displayed images (Figure 2.13). However, two screenshots can allow an adversary to determine which pixels change values (foreground pixels) and which don't. A similar solution is proposed by Grange et al. (Figure 2.14) [11].

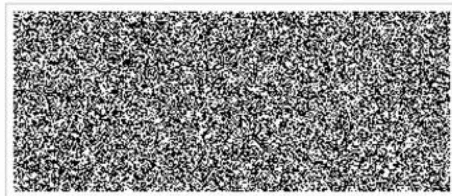


(a) input image (b) encoded image (c) encoded image (d) expected view of recovered image

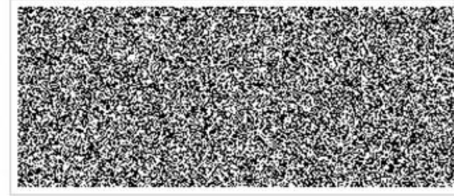
Figure 2.13: Samples of results from the visual cryptography approach proposed by Hou et al. [10].

Chia et al. applied distortion to images representing the content of the display by randomly selecting N distortion planes (Figure 2.15) [12]. The aim is to limit the meaningful visual static contents of a display from being captured by screenshots. The authors exploited image-processing techniques to distort the visual data of a display and present distorted data to the viewer. Given that a screenshot captures distorted visual contents, the method yields limited useful data. The idea is to

exploit the HVS to allow viewers to automatically recover the distorted contents into a meaningful form in real-time.



(a) generated encrypted image1



(b) generated encrypted image2

(c) perceptible image, visible but never displayed

Figure 2.14: Samples of result from the visual cryptography approach proposed by Grange et al. [11].

All the approaches discussed present the same four main drawbacks.

The first is their limited scope: they are restricted to textual content. Moreover, the techniques require images to be converted to greyscale.

The second drawback is poor usability: the quality of the visualised images is significantly impacted compared to the originals. Users of legitimate screenshot-taking applications would not accept such a loss of quality.

The third drawback is that they cannot be performed in real-time. They require important pixel by pixel computations. Furthermore, they are 'content-dependent'. Indeed, they make a different treatment to each pixel depending on whether the

pixel is inside or outside a letter ([10, 12, 48, 49]). This implies that the content to protect must be known in advance, which prevents them from being performed on the fly.

The fourth drawback is that an adversary capable of taking multiple screenshots could easily infer the screen content as foreground and background pixels are treated differently.

All these points are obstacles to the adoption of these solutions for the general goal of protecting any content displayed on the screen.

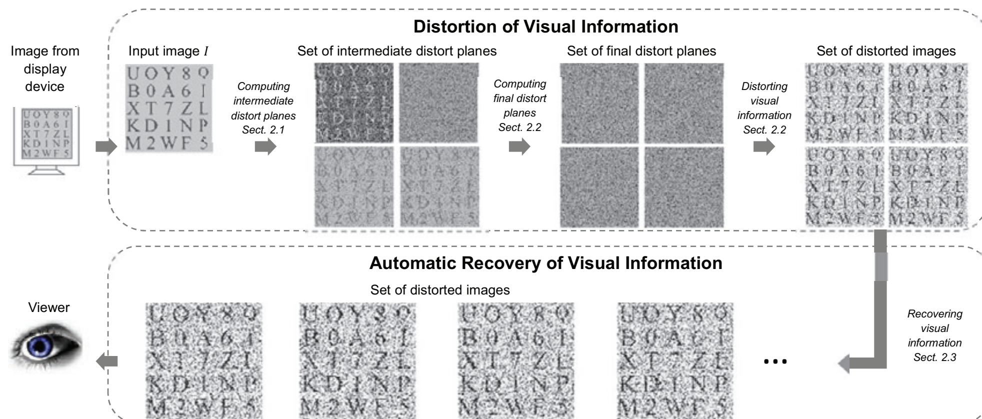


Figure 2.15: System overview and distortion planes used by Chia et al. [12].

Retinal Persistence Some approaches found in the literature partially overcome the previously mentioned obstacles to general screen protection. These approaches are based on a property of the HVS called retinal persistence (also known as the afterimage effect).

This property is defined as the fact that the HVS can process 10 to 12 images per second. After staring to an image for a fixed time and due to photochemical properties of the retina, the image will be retained for few milliseconds, even when it is no longer displayed in front of the viewer [149]. Thus, when an image is replaced

by another image during a period above the fifteenth of a second, there is an illusion of continuity [150]. This implies that if subparts of an image are displayed at high speed, the human viewer can see the whole image with no changes, whereas one frame alone only contains a subpart of the original information.

As seen in Section 2.3.1.2, the first work to use retinal persistence against screenloggers targets the specific case of authentication with a virtual keyboard [5].

Park et al. extended the use of retinal persistence to pictures [13]. Their goal was to prevent screenshots of images published on online social networks to avoid identity theft. The proposed mechanism is to use 'privacy black bars' that move sufficiently fast so that the viewer does not perceive them, but when using the screenshot function, the bars appear in the resulting image. Contrary to the approaches based on visual cryptography ([10–12]), this approach does not require the image to be in greyscale form and does not apply computations by pixel but instead by groups of pixels. The consequences are that the image quality as measured using peak signal to noise ratio is much less affected and that computations can be performed on the fly in real-time.

However, the use of a precomputed mask and a periodical pattern makes it relatively simple to reconstruct the original images from few screenshots. Moreover, in the case of textual data, the use of large bars may not be the best solution because important parts of the text would still be visible on one screenshot, which may be enough for an attacker to extract sensitive data.

As a result, the authors proposed to display random blur blocks on the image instead of black periodic bars. However, only the usability of this method was measured, and not the security in the case of images or sensitive textual content.

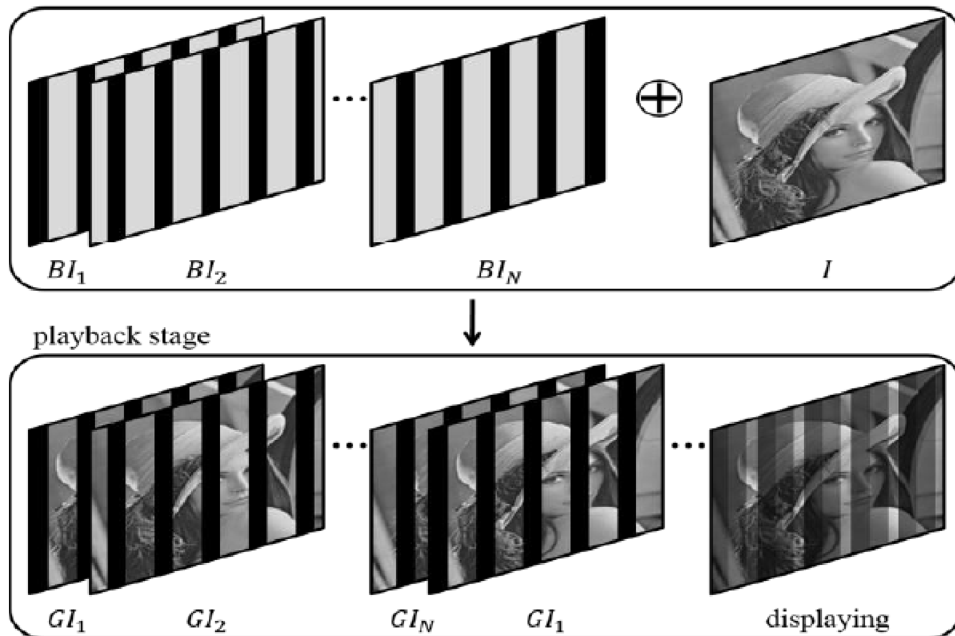


Figure 2.16: Images projection using moving privacy black bars [13].

2.3.3 Synthesis

Most of the works aiming at protecting the screen's content target the shoulder-surfing attack. Most of the approaches proposed in this context specifically focus on the login operation [119, 121]. The goal is to prevent attackers from stealing passwords by looking at the victim's screen. The limitation of these approaches in our context is that they rely on authentication as a dynamic process with different actions performed by the user, whereas we also aim at protecting screens containing static information with a passive user.

Scant few works address the shoulder-surfing problem in the general context. These studies mainly aim to protect confidential documents from malicious viewers. Many of these methods use spatial data [7, 131]. However, the screen-recording problem does not imply any spatial considerations as the adversary remotely observes the victim's screen by using the screen logging function offered

by the OS. Moreover, anti-shoulder surfing solutions are highly intrusive. Some of them replace displayed data with aliases [8], while others propose to display documents with the user's handwriting [133] or display images using filters [9]. These techniques reduce the usability of these approaches. Therefore, these solutions must be limited to sensitive data. Determining what data is sensitive may be done automatically using NLP techniques [151], but the drawback is that what is considered as sensitive may be different for different users. Moreover, this additional processing may prevent real-time display. Another solution is to ask users to explicitly specify what data they want to be hidden. This is the approach proposed by Mitchell et al. [8]. However, this greatly reduces the scope of these approaches as they are limited to security-aware users who must be well-informed and care about the threat. This issue is intrinsic to the anti-shoulder surfing solutions as the attacker sees exactly the same information as the victim by looking at the same screen, as opposed to screenlogging attacks, for which it may be possible to find less intrusive solutions that are transparent to the users.

In this context, approaches aiming at preventing automatic character recognition may be useful. These are CAPTCHA-based techniques [134, 135]. The methods are effective for short texts, but their application on longer texts would considerably affect users' comfort. A possible solution would be to detect what can be considered as sensitive information and display only this data using CAPTCHA. However, determining what data is sensitive is another issue, as explained earlier. Moreover, CAPTCHA techniques would only protect the screenshots from automatic but not human exploitation. Even to prevent automatic exploitation, many works in the literature have demonstrated the limits of CAPTCHA-based approaches [137, 138].

Few works in the literature specifically address the screenlogging threat. Some target the authentication process [48, 49]. Specific treatments are applied to the

pixels to display the virtual keyboard's keys in a way that a screenshot does not contain any meaningful information while the user is able to see the content of the keys. These methods are based on the Gestalt laws, which include several properties of the HVS [152, 153]. The application of all these properties results in noisy images in greyscale representing the keys of the virtual keyboard. Therefore, this is not adapted to protect general content displayed on the screen.

Closer to the subject of this work, few methods aim at countering screen capture in a more general context not limited to authentication. However, these methods present several drawbacks. Some achieve screenshot protection using image-processing techniques [12] or visual cryptography concepts [10], generating important computations, particularly when applied to large images. Moreover, they require converting the target image to greyscale. The resulting image quality displayed is significantly degraded compared to the original. Finally, these approaches are only secure against adversaries taking a single screenshot. These constraints are obstacles to the adoption of such methods for general screenshot protection.

A different approach is presented by Park et al. [13]. The proposed solution is based on visual persistence. The first work that used this property to offer screenshot protection was limited to digital numbers composed of known segments displayed at a high frequency [5]. Park et al. extended this approach to pictures. Their goal was to find solutions against identity theft by screen capture on online social networks. This approach is less complex than visual cryptography approaches ([10, 12]), and the original image is less affected because it only relies on large vertical bands sliding on the screen at a high enough frequency to avoid disturbing the users. However, as the method is based on large bands covering the screen periodically, the image can be recovered from a few screenshots.

Moreover, even a single screenshot can contain sensitive information between the widely spaced bars.

Table 2.2 shows that existing anti-screenshot approaches are all limited in terms of scope, usability and security.

Table 2.2: Limitations of existing approaches against screenshots.

Reference	Objective	Technique used	Limitations
[48] [49]	Secure virtual keyboard authentication	Use of Gestalt psychology (motion perception): pixels inside numbers move in a different direction than pixels outside	• Noisy images • Need to identify pixels inside letters • Per pixel computations • A few screenshots can suffice to identify the direction in which pixels are moving
[5]	Secure virtual keyboard authentication	Use of retinal persistence on digital numbers divided into segments	• Limited to digital numbers • Different treatment for each digit • A few screenshots suffice to deduce the digits (only seven segments)
[10] [12]	Protect copyright images from screenshots	Visual cryptography: pixels inside letters move while others don't	• Noisy images • Need to identify pixels inside letters • Limited to text (vs images) • 2 screenshots suffice to determine which pixels move and which don't
[13]	Protect social network images from screenshots	Retinal persistence: vertical bars sliding on the image	• 2 well-timed screenshots suffice to see the whole screen • Wide space between bars • No usability study

Therefore, all these works have in common that they are not intended for overall protection of the system but rather target specific applications or files, often chosen

by the user. This assumes that users are security aware to a certain extent, whereas most of them have not been made aware of the threat or do not have enough skills. Moreover, users might be negligent or mislead by malicious attackers. This general lack of security awareness is well illustrated by a recent international survey [143] that revealed, for example, that more than 50% of the respondents in Italy and Germany were unaware of what ransomware is. Application developers also have tools against malicious screen recording, such as security flags forbidding screen recording of specific pages. However, this solution relies on developers thinking of activating this flag for their sensitive applications, which is not a safe assumption. Moreover, some applications are so broad that developers cannot know in advance whether the information displayed will be sensitive or not.

For these reasons, the necessity is clear to have a screenshot-protecting approach that is transparent to users and offers global data protection for screen displays without the need for user intervention.

3 | Adversary Model

Screenloggers have the advantage of being able to capture any information displayed on the screen, offering a large set of possibilities for the adversary compared to other spyware functionalities. We therefore start this chapter by defining the different capabilities of a screenlogger (Section 3.1). We show that taking a screenshot is relatively simple and frequently does not require specific privileges. This threat is compared to other possible attacks aiming at the same goals, showing that screenloggers are often simpler yet less studied in the literature. We then define some applications scenarios where the identified capabilities can be put in practice and be particularly harmful (Section 3.2). Finally, we present the system and threat models that define the scope of the thesis (Sections 3.3 and 3.4).

3.1 Screenloggers' capabilities and comparison with other attacks

Screenlogging attacks uniquely cover a large set of capabilities since they can capture any information displayed on a screen, ranging from passwords entered with a virtual keyboard to any sensitive data displayed on the screen.

This attack is quite simple to perform in a PC environment using legitimate commands or API calls (i.e., Graphics Device Interface (GDI) API, Desktop Duplication (DD) API), whereas it is more limited on smartphones.

The possibility of taking screenshots of other apps exists only on Android systems, through exploiting the ADB or the MediaProjection API vulnerabilities.

In order to use MediaProjection, applications need to acquire the CAPTURE_VIDEO_OUTPUT permission. Note that for security reasons, the API

cannot take screenshots of applications that use FLAG_SECURE. However, the keyboard is not hidden as opposed to the rest of the screen[154, 155]. Other limitations include the user being asked for screen-recording permission at the beginning of each session, and a screencast icon appearing in the notification bar [156].

The ADB screencap is another way to take screenshots of apps on Android devices [46]. ADB is an Android SDK tool used by developers to debug Android applications [157]. The Android device must be connected to a PC for launching ADB with its USB debug option activated. ADB commands, such as install, debug and more can be sent. 'Screencap' is a shell command that allows screenshots of any app without any required permissions. Several malware programs exploiting ADB have been found [158–160]. A screenlogger using ADB to take screenshots was implemented [46].

The main capabilities of screenloggers are described in the remainder of this section. For each capability, alternative methods of execution are discussed and compared to screenshot-taking.

3.1.1 Credential theft on virtual keyboards

The process of entering credentials has always been a very sensitive task targeted by attackers. For this purpose, keyloggers, which can record the victim's keystrokes, are widely used. However, due to the very high number of machines infected by keyloggers (in 2006, the SANS Institute estimated that up to 9.9 million machines in the US were affected [32]), some website developers, especially banks have switched to virtual keyboards (e.g. State Bank of India, ICICI Bank, Bank Millennium, Seven Bank, Bank ABC, Canara Bank, HDFC Bank, Credit Libanais, Heritage Bank, Syndicate Bank, LCL france, SocietÃ© Generale, la-

banquepostale). However, attackers have evolved and use more sophisticated techniques in order to recover the entered passwords. In addition, the emergence of touchscreen smartphones has increased the number of people using a virtual keyboard.

The first way of stealing credentials entered with a virtual keyboard is to take screenshots during the log in, either at regular time intervals or at each user event. The screenshots are then sent to the attacker for analysis to infer the password. A recent example is the well-known mobile banking malware family Svpeng which added screenshot capability to its functionalities in July 2017 [25].

The other techniques which can be used by attackers to recover a password entered with a virtual keyboard are:

Keylogging attacks on smartphones

Another method of stealing passwords entered with a virtual keyboard on smartphones is to use a keylogger. Traditionally applied to PCs, keylogging can be performed in-band (i.e., through the main channel of keystrokes) either by intercepting keystrokes using specific hardware before it reaches the OS, or by intercepting keystrokes at the OS level (i.e., kernel or API-based) [161]. Some studies have discussed out-of-band keylogging, using side channels such as electromagnetic or acoustic emanations [162, 163].

The smartphone keylogger landscape is relatively different from that of PCs. In-band attacks are almost infeasible, except on rooted or compromised devices because of the restrictions of the OS security design [164]. Indeed, according to the Android security model, an app cannot read touch events performed by the user on other apps.

Another technique for keylogging on smartphones is by disguising as a benign keyboard that the user would download from the store and set at the default keyboard. The attack needs an active and voluntary action from the user to set the malicious keyboard as the default keyboard. An example of popular keyboard app for Android and iOS is AI.type [161]. This type of keylogger is used, for example, by commercial apps to enable spying on children or spouses by installing it on their device [165].

Another way to record keystrokes on smartphones is to use ADB, on the condition that it is enabled. Additionally, ADB is only able to record the coordinates where the user tapped. Therefore, the attacker must know the keyboard layout to infer the keystrokes. Such an attack was implemented [46].

Android Accessibility Services, designed to allow people with disabilities to interact with Android devices, provides another method of recording keystrokes [34, 35]. The Accessibility Services permission offers a large set of actions, including recording what the user taps, and is used by several apps not targeting users with disabilities. However, a limitation is that Accessibility Services cannot access password input fields, so it cannot be used to steal credentials [166].

SYSTEM_ALERT_WINDOW is another Android permission that can be abused to perform an attack that records users' keystrokes [34, 35]. Users are not explicitly prompted to allow apps to access this permission when installing apps from Google Play. With the permission, an attacker can create a transparent layer that overlays the virtual keyboard of an Android device and captures all attempts to tap the screen. Attackers can determine what a user is typing by correlating the coordinates of the user's tap with the character positions on the keyboard. However, the keyboard layout must be known by the attacker. This attack has been demonstrated [35].

In the literature, several studies about keyloggers on mobile devices focus on tap inference using side channels [164]. These methods consist of using smartphone sensor data, such as accelerometers or gyroscopes, to infer the coordinates where the user tapped. However, the methods have the drawback of not being accurate in noisy data situations. For example a prediction model to infer a 4-digit PIN using accelerometers had an accuracy of 43% (within five attempts) when the users entered the PIN in a controlled setting while sitting, whereas the accuracy was 20% in uncontrolled settings, such as entering the PIN while walking [36]. Moreover, the model of the targeted smartphone must be known to compute the correspondence between the keys and the coordinates.

Malicious web extension using the document object model

Web extensions are JavaScript modules installed in the browser to offer functionalities such as preventing ads or showing the number of inbox emails. It was found that both Google Chrome and Firefox web extensions have serious security issues and can access various types of data, including the user's web history and entered passwords [37]. Indeed, they can access all the data entered by the user, including passwords, if they are granted the 'access all website data' permission. A statistical study was performed showing that 71% of the top 500 most-used web extensions require this permission [38]. Using this permission, the malicious extension injects scripts into the web pages the user is visiting. Since the scripts are running in the page's environment, they can read the password the user enters from a Document Object Model (DOM), which defines the logical structure of documents and the way a document is accessed and manipulated [39]. However, this attack is much more difficult or even impossible when a website requires the user to enter a password with a virtual keyboard. An example is illustrated in Figure 3.1 (LCL bank). The customer uses a keyboard provided by the web site,

which is dynamically loaded at each visit. Instead of using the standard password field, the virtual keyboard uses a hidden input field called postClavier. When the user presses a key, the field records the key's position instead of its value.

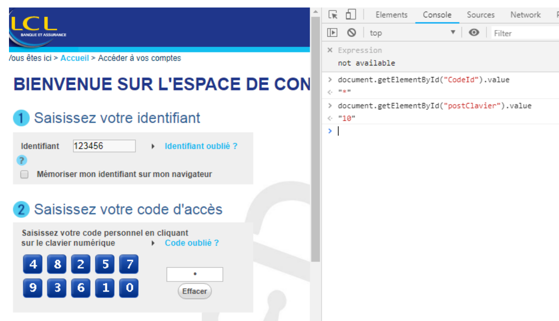


Figure 3.1: LCL bank's virtual keyboard.

For example, in Figure 3.1, when the user enters '0', the recorded value is 10, which is the key's position on the virtual keyboard. Another example is illustrated in Figure 3.2 with Oney's virtual keyboard: when the user clicks on number 0, a hidden field is filled with the pair (1, 3) where 1 is the row index and 3 is the column index. In this situation, the malware developer cannot know the password without taking screenshots.

Figure 3.2: Oney bank's virtual keyboard.

Phishing attacks

Phishing is a common and effective attack for credential-stealing. The attacker misleads the user to a fake web site or application which has the exact same design and user interface as the real one, and the user is asked to fill in credentials. Once the login button is clicked, the data is collected and sent to the attacker. The phishing attack is the result of combining social and technical elements. The success of such attacks depends on the strategy used to combine those two elements. In the case of a virtual keyboard, we can identify two cases.

The first is the scenario of a web application using a virtual keyboard. The attacker uses social engineering techniques such as email or SMS to redirect the user to a website with the same look as the real one. The fake site reproduces the same virtual keyboard, and users are asked for their login information. The phishing threat, however, is widely studied and there exists many anti-phishing mechanisms which can for instance block suspicious emails. Automated ML solutions are employed such as probabilistic models, rule-based classification and more. Several anti-phishing techniques have been proposed [40].

The second phishing scenario steals credentials entered on a mobile phone app with the phone's virtual keyboard. Making use of SYSTEM_ALERT_WINDOW and the ability to overlay other apps, the malicious application detects when a banking app is opened and shows unsuspecting users a window mimicking the targeted app's password prompt, into which users enters their account logins and passwords [35].

Shoulder surfing

One of the weaknesses of virtual keyboards is 'shoulder surfing'. This form of attack occurs when a user types a password using a visual keyboard and the attacker

is behind watching the displayed characters. In some cases, cameras can be used. Generally, a trained user takes precautions to avoid shoulder surfing, whereas others do not take measures to cover their inputs. To perform this attack, the attacker must have the opportunity to be physically present near the user, and guessing the password requires several observations [167].

3.1.2 Sensitive data breach

In addition to stealing credentials, sensitive data theft is another important goal for an attacker. The stolen information could be intellectual property or industrial data that the attacker can sell to a competitor or use to make a rival product. Possible stolen data also include a target's identity cards, credit cards or banking details that could allow an attacker to impersonate the victim or make transactions.

Screenloggers can capture any data displayed on the screen, including locally stored documents, visited websites or any software employed by the user and displayed on the screen.

An attacker has several other options to reach their goals. The following are the principal attack methods.

Keyloggers

As mentioned above, keyloggers can be used to steal sensitive data when typed using the keyboard; however, they cannot steal data visualised by the user.

Network-based attacks

Three main kinds of network-based attacks can be used to steal sensitive data from victims: sniffing, Address Resolution Protocol (ARP) spoofing and Domain Name System (DNS) spoofing.

A sniffing attack allows attackers to intercept packets and capture data from a computer network.

To perform an ARP spoofing attack, an attacker sends spoofed ARP packets to intercept data on a network and steal information by diverting communications flows between a target machine and a gateway such as a router or box [36].

Similarly, DNS spoofing consists of corrupting the DNS by introducing data into the DNS resolver's cache, causing the traffic to be diverted to the attacker's computer to steal sensitive information [168].

However, encryption and the use of SSL/TLS ensures that data sent across a network from one host to another is unreadable to a third party. Moreover, there is a wide range of mechanisms for ARP spoofing detection and prevention. Examples include using static ARP or dedicated anti-ARP tools [169]. For DNS spoofing, there also are some protection mechanisms used by updated DNS software applications (e.g., versions of BIND 9.5.0-P1 and above).

Spyware accessing the file system

When a malware program infects a target in a desktop environment, the malware has the same rights as the user who installed it in the user session. As a result, the malware can access all the user's files. However, it cannot access the other users' data. Moreover, the malware cannot read the information visualised on a browser, which covers a large set of sensitive activities such as making online banking transactions, filling in a form or working in the cloud.

Note that mobile systems have a file access pattern which is quite different from that of desktop environments. The different apps are strictly separated and cannot access other apps' data. The apps also cannot access the user's data unless specific permission has been explicitly granted by the user (such as the

READ_EXTERNAL_STORAGE permission on Android, which is a dangerous permission explicitly granted by the user at run time [170]).

Malicious web extension

A web extension can use the 'access all websites data' permission to access the DOM of every web page. Some extensions legitimately need access to this data to perform their function. For example, video-blocking software must read a website's code to see that a clip is set to play automatically and to know how to stop it. However, the 'access all websites data' permission can be diverted. Malware makers have hijacked or even bought legitimate extensions from their original developers and used the access to pump invasive advertisements into web pages [166]. Thus, these malicious extensions can access any sensitive information displayed in the browser, with a few exceptions such as the virtual keyboard case (explained in the previous section). However, malicious extensions cannot access user data that is outside the browser activity. Some techniques were developed to detect suspicious behaviours depicted by browser extensions [171].

3.1.3 Spying on the victim's activity

Monitoring the victim's activities is another possible goal for an attacker. The attacks performed in this case usually are targeted attacks.

Screen-recording functions may allow an attacker to observe activities on victims' devices, for example, the Odlanor spyware discovered by the security firm ESET in 2015 [172]. The Odlanor trojan targets two of the largest poker sites, PokerStars and Full Tilt Poker. Odlanor infects the victim's machine either by masquerading as various general-purpose programs, such as Daemon Tools, or by loading onto the victim's system through various poker-related programs (players

databases, poker calculators, etc.). Once installed, the screenlogger takes screenshots if the victim is running PokerStars or Full Tilt Poker. The screenshots are then sent to the attacker's remote computer to extract the players ID and the hands of the infected opponents. Both targeted poker sites allow searching for players by their player IDs, hence the attacker can easily connect to the tables on which they are playing and start cheating. Researchers have found Odlanor on machines in several Eastern European countries.

Another significant example is the attack performed by the 'Carbanak cybergang' discovered by Kaspersky in 2015, which hit more than 100 financial institutions in 30 countries and stole up to $1 billion [21]. The employees' workstations were infected through phishing emails containing infected attachments. Then, a RAT was installed on the workstations of key employees controlling ATMs. The hallmark of this attack was the screen-recording task: the attackers used the screengrabs to create a video recording of daily activity on employees' computers, amassing knowledge about internal processes before stealing money by impersonating legitimate local users. According to Kaspersky, evidence suggests that the attackers would hide in a compromised system without being detected between two and four months, taking screenshots getting to know the way the organisation worked and tailoring their tools and techniques accordingly. The managing director of the Kaspersky North America office in Boston, Chris Doggett said: 'This is likely the most sophisticated attack the world has seen to date in terms of the tactics and methods that cybercriminals have used to remain covert' [22]. Indeed, the screenshots enabled the attackers to mimic the employees' activities and everything would look like a normal, everyday transaction, enabling the malware to remain undetected.

Keyloggers may also be used to spy on other persons' activities by monitoring what they are typing on their devices. For example, this functionality is used for

parental control. However, as we have seen earlier, the information collected by keyloggers is limited to a user's input, which represents only a small portion of what happens on the computer.

Screenlogging, therefore, is the only way to observe with precision all the victim's computer activities as it can capture everything displayed on the screen.

3.1.4 Blackmail

Another possible goal for an attacker is to blackmail the victim for money by acquiring sensitive or private information. The means to achieve this goal are similar to those enabling the theft of sensitive data as described earlier. In addition to these attacks, we must add ransomware and webcam blackmail.

Webcam blackmail

According to the UK National Crime Agency, webcam blackmail, also called sextortion, happens when criminals use a fake identity online to persuade the victims to perform sexual acts in front of their webcam, often by using an attractive woman to entice the victim to participate [173]. The attacker then threatens the victims to share or publish the recorded images if they do not pay an amount of money. This attack may be effective and has the advantage of not requiring any infection of the victims' devices because they turn on the webcam deliberately.

Webcam/Microphone activation without the user's consent

Another way to perform blackmail is to activate the user's webcam or microphone without their consent and use any sensitive captured audio or video against the victims. For example, it is possible to perform 'clickjacking' to trick users into enabling a webcam or microphone by misleading them (e.g., through the Flash

Player webcam settings dialog). Clickjacking occurs when a user clicks on something seen on the screen but is actually clicking on something controlled by the attacker [174]. Several defence mechanisms against clickjacking have been proposed in the literature and included into browsers [175].

Another blackmail avenue is to exploit specific vulnerabilities that enable turning on cameras or microphones of the victims' devices. An example of such spyware is Chrysaor, which was discovered in 2017 [176]. Chrysaor is Android spyware believed to be created by the Israeli firm NSO Group Technologies. The spyware appears as a targeted attack in which 'a few authors spend substantial effort, time, and money to create and install their harmful app on one or a very small number of devices' [176]. Once Chrysaor is installed, a remote operator is able to surveil the victim's activities on the device and within the vicinity, leveraging microphone, camera, data collection, and logging and tracking application activities on communication apps such as phone and SMS [176]. Upon installation, the app uses specific vulnerabilities to escalate privileges [176]. Therefore, this type of attack requires an important investment from the attacker to find and exploit specific vulnerabilities, and is limited as vulnerabilities are regularly patched.

Ransomware

The typical definition of a ransomware is malicious code that encrypts important files of a user and then demands a ransom payment in exchange for the decryption of the files [177]. This attack gained a reputation as an effective means of extortion and broadly spread since its first appearance. A long-term study of ransomware attacks observed between 2006 and 2014 have been conducted [178]. The authors analysed 1,359 samples belonging to 15 different ransomware families. Results suggested that despite a continuous improvement in encryption, deletion and communication techniques, it is possible to build a defence system to

stop many ransomware attacks by monitoring abnormal file system activity. Moreover, numerous countermeasures have been proposed since the first ransomware appearance, and a common way of protecting against ransomware is to back up files.

3.1.5 Reconnaissance

Before infecting a system, it is current practice to gather information about the target system to know if it is a potentially interesting target worth attacking. This is known as reconnaissance.

As argued by Symantec, a screenlogging operation can be used by malware downloaders before an attack to better choose potentially interesting targets (e.g., a machine belonging to a corporate network) [179, 180]. Screenlogging can also be used when infecting a corporate network to identify which employees have higher responsibilities and thus which computers can be the most interesting targets for the attacker.

Other ways of performing the reconnaissance task include the following.

Examining the file system, installed applications and running processes

Similar to a downloader taking screenshots before deciding whether to infect a system, another possibility is to investigate the file system and search for keywords, for example, to detect if a system is a corporate environment. A malware program may also look at the installed applications and running processes. Although this process enables access to a larger quantity of information about the system than taking screenshots, it does not allow for a precise view of the victim's activity, such as browsing, which can give a more precise idea of the victim's situation.

Gathering public data

The simplest way to gain information about a person in preparation for an attack is to gather publicly available information about the targeted victim by looking at search results and public data on social networks. However, the amount of information that be collected in this way is limited, especially if the victim's social networks are private.

Social engineering

Social engineering consists of manipulating a person into giving information to the social engineer. The technique is often one of many steps in a more complex fraud scheme [181]. Although social engineering is often a targeted attack aiming at infecting a specific system, it can also be used to decide whether it is worth infecting a system based on the collected information. Social engineering attacks can breach even the most secure systems, as the users themselves are the most vulnerable part of the system [182]. In a recent reconnaissance attack, attackers used online social networks and attempted to approach their targets by sending friend requests to multiple users in the target's social circle [183, 184]. Performing social engineering can be complex, time-consuming and, therefore, costly to undertake [185].

Packet-sniffing

Usually, the analysis of network traffic is done in the context of network reconnaissance, which is a way to test potential vulnerabilities in a computer network before an attack, such as through port scanning. Network analysis is done once the target is already chosen and the hacker is planning the attack process. If the goal instead is to collect information about potential victims to choose whether to infect them, then network analysis is more limited. The adversary can per-

form packet-sniffing to see the type of applications the victims use and the type of data they send and receive, but it is highly limited by packet encryption and the attacker's necessity to have access to the LAN.

3.1.6 Synthesis

This study of various screenlogger capabilities highlights the difference between desktop and mobile environments.

Indeed, in desktop environments, taking screenshots is a legitimate functionality accessible to every application. Screenshot-taking does not suffer from the same restrictions imposed upon other hacking methods (e.g., requiring 'access all website data' permission for web extensions, activating the webcam or microphone without the user's consent, physical presence for shoulder surfing). Moreover, taking screenshots is not limited by scope (e.g., only accessing data entered by the user for keyloggers, social engineering, shoulder surfing) or countermeasures (e.g., SMS/spam filters against phishing, backups and other countermeasures against ransomware, specific virtual keyboard implementation against malicious extensions).

On the contrary, screenloggers in mobile environments do suffer from restrictions that hamper their effectiveness compared to alternative attacks (e.g., the impossibility for an application to take screenshots of another application, having to get the user's consent at the beginning of each screenshot session when using the MediaProjection API, having to enable the debug mode). These restrictions explain why screenlogging does not stand out as a threat for mobile environments as much as it does in desktop environments.

3.2 Attack scenarios

In this section, we present three examples of attack scenarios where attackers achieve their goals using the different capabilities of a screenlogger.

3.2.1 Online banking customer attack (capability 1 + capability 2)

In this scenario, the attacker targets online banking customers. The goal is to steal login credentials to access bank accounts and perform transactions (capability 1), or to impersonate the victims and make changes to their banking data by stealing personal information (capability 2). The process is illustrated in Figure 3.3.

Once a machine is infected, the screenlogger monitors the on-screen activity to know if an online banking website is opened. There are several ways of doing this. For example, on Windows systems, a possible solution is to use the EnumWindows function, which enumerates all top-level windows on the screen by passing the handle to each window containing the window's name [186]. The screenlogger must contain a function that facilitates matching the collected windows' names with a pattern defining a list of bank site names. Another possibility is to seek a specific window by its name or by a substring of its name using FindWindowA and FindWindowEx [187]. When a screenlogger detects that an online banking login webpage is on the screen, it can start taking screenshots to steal the username and password. Screenshots can be used to bypass virtual keyboard security. In other cases, traditional keylogging functions can be used. Then, the images are compressed and sent over the network to the adversary's server. By exploiting the screenshots either manually or using OCR, the attacker can obtain the victim's credentials. The feasibility of such an attack has already been demon-

strated [46] . The authors implemented a screenlogger that steals credentials by taking screenshots and sending them over the network [46].

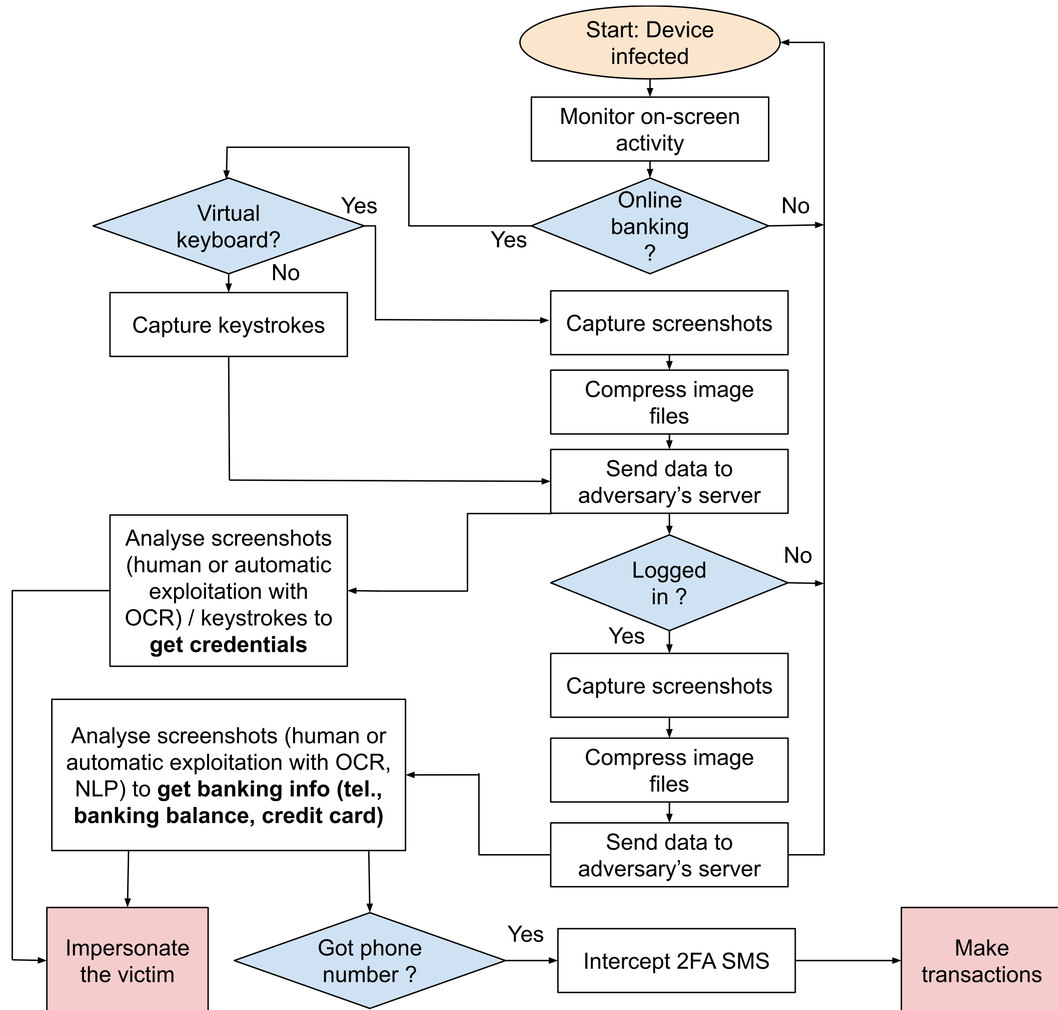


Figure 3.3: Screenshot attack scenario 1.

In this scenario, the attacker goes beyond stealing credentials by continuing to take screenshots while the user is logged in. This process can enable the attacker to obtain sensitive data and personally identifiable information such as the customer's name, banking balance or credit card data. Again, the screenshots are compressed and sent over the network, and they can be processed either manually or using

automatic tools. In this case, besides OCR, the adversary can use NLP tools to automatically detect sensitive information in the screenshots' content [151]. The retrieved information can allow the performance of several malicious actions. For example, credit card data can be used to make online payments. Moreover, obtaining the customer's phone number can be useful in case of two-factor authentication using SMS. Indeed, it would allow the attacker to intercept the received SMS by using the well-known vulnerabilities of SMS, such as SS7 [188]. The attacker can also use the victims' personal information to impersonate them and change their bank account data by calling the bank, for example.

3.2.2 Real-time monitoring of the victim's activity (capability 3)

In this scenario, we suppose that the attacker wants to have a real-time view of the content visualised on the victim's screen. Many cases require such a capability:

- Real-time monitoring can be used when governments want to spy on activists or, more generally, to spy on opponents in a conflict. The content visualised on the screen helps anticipate the actions of the opponent and manage ongoing political unrest. An example is XtremeRAT, which was used against Syrian anti-government groups [189].
- Observing the victim's screen in real time has also been used for an attack targeting poker players to see their hands and cheat [172].
- Another case is when the attacker is spying on the victim for business purposes, such as when competitors want to have real-time data about deals in preparation. Attackers might also want to anticipate financial market movements to buy or sell securities by targeting stock-exchange employees.

The process of the real-time monitoring scenario is illustrated in Figure 3.4.

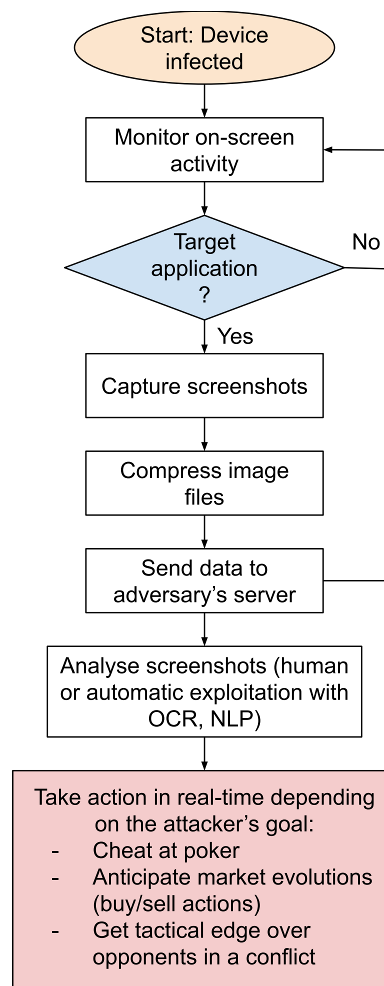


Figure 3.4: Screenshot attack scenario 2.

The malware can monitor on-screen activity, as explained in the previous scenario. In this case, the target applications depend on the attackers' purpose: for example, messaging applications in the context of an armed conflict, or poker websites in the case of an attack targeting poker players. The screenshots are sent, compressed and analysed. In a scenario targeting specific people, it is most likely that the screenshots will be seen by a human directly. However, screenshots could also

be exploited automatically using OCR and NLP tools to search for specific information. The retrieved information can then be used to act in real time depending on the attacker's goal.

3.2.3 Blackmail (capability 4)

With blackmail, screenshots are used to obtain private content that the victim does not want to be published, such as messages or photos (Figure 3.5).

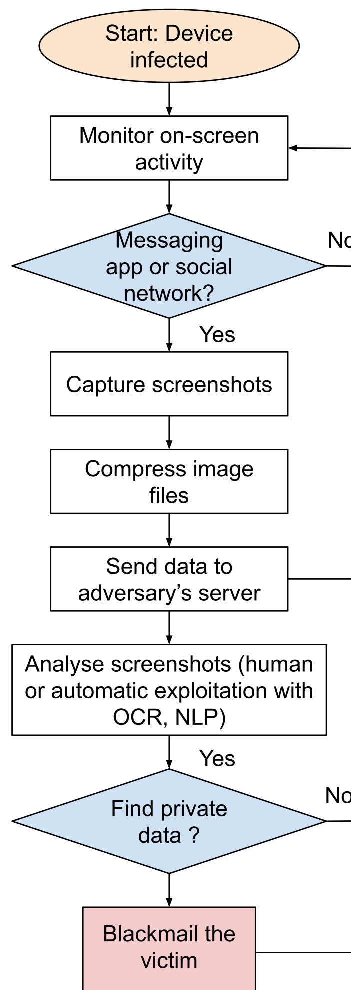


Figure 3.5: Screenshot attack scenario 3.

If made public, the revealed content would harm the victim's social or professional life. Once a device is infected, the malware detects when the victim consults emails, social media accounts, messaging applications, dating websites and more. The compromised data can also be a video conversation, such as on Skype or Viber. The malware then takes screenshots of the visualised content and uses it to blackmail the victim.

3.3 System Model

3.3.1 Targeted systems

The targeted systems are desktop environments. The main reason why our work focuses on computer operating systems is that the screenshot functionality is a legitimate functionality offered to any application. In contrast, on smartphones, the principle is that apps cannot take screenshots of other apps, and the only way to accomplish this is to exploit specific vulnerabilities or to divert some libraries. However, many limitations exist for these techniques, such as permission required from the user at the beginning of each session, or a recording icon displayed in the notification bar. In sum, the architecture designs of mobile systems and computer systems are fundamentally different, which may lead to different solutions.

3.3.2 Targeted victims

Targeted victims may be any individual or organisation, ranging from typical laptop users to small companies or powerful institutions.

Another assumption in our system model is that the victims are not particularly security aware, which implies they are not necessarily cognizant of the existing threats and will not install a specific protection against screenshots, such as a spe-

cific viewer to open documents in a secure environment, which prevents screenshots.

3.4 Threat Model

3.4.1 General Description

Our threat model is composed of a victim, an attacker and spyware with a screenshot functionality.

In this model, a screenshot is defined as a reproduction in an image format of what is displayed on the screen, even if all pixels may not be visible. Screenloggers must rely on a functionality offered by the operating system to perform their attack.

The adversary's goals are diverse. They can range from general activity monitoring, which requires to see the whole screen, to sensitive data theft, which can be limited to some areas of the screen.

3.4.2 Operating process

Attackers may infect a system using common methods such as trojans, social engineering or through a malicious insider. The adversary has no physical access to the victim's device (except in the case of a malicious insider). They have no knowledge about the system and tools installed on it before infection. We also assume they have not compromised the victim's device at a kernel level. Apart from that, the attacker can use any technique to evade detection, including hiding by injecting API calls into system or legitimate processes, dividing its tasks between multiple processes, making the API calls out of sequence, spaced out in time, or interleaved with other API calls.

To reach their objective, attackers take screenshots of the victim's device. The data may be either (1) extracted automatically using OCR tools inside the victim's device locally, then sent to the attacker's server using the victim's network interface or (2) extracted, also using OCR tools, on the attacker's server after screenshots have been transferred from the victim's machine to the attacker's as compressed image files. The screenshots can also be analysed manually by the attacker. Moreover, the screenshots may be taken and sent at regular or irregular rates. These different options are depicted in Figure 3.6.

The adversary does not need any advanced hardware - only a remote machine with network access and known by the malware. They also have state-of-the-art OCR and NLP tools at their disposal and enough resources to run such tools and store the screenshots. They have no interaction with the victim's device except the capability of receiving files transmitted by the spyware via the network and optionally the possibility to send commands to the spyware, as illustrated in Figure 3.6.

3.4.3 Scope

Exclusion of shoulder surfing

Shoulder surfing is a form of social engineering used to covertly obtain information such as passwords or identifiable data. A simple glance over the shoulder can be used to see the PIN code of someone's main bank card. Shoulder surfing can also be used to spot login details for an online service or obtain details to access business services [190].

Although shoulder surfing and screenlogging both exploit information seen on a victim's screen, they are different methods. Screenloggers must rely on a functionality offered by the operating system to perform their attack, whereas shoulder

surfers directly observe the victim's screen. For this analysis, we assume that a hacker is intervening from a remote site using software and networking capabilities.

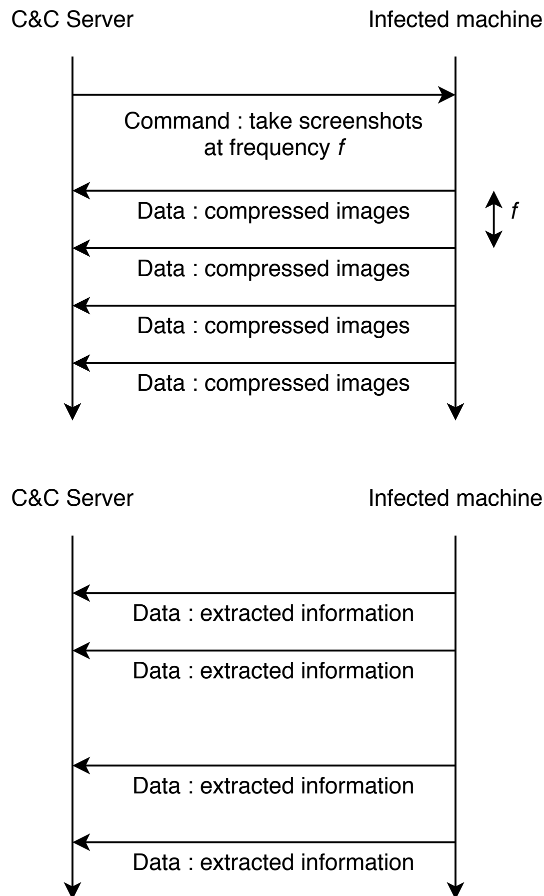


Figure 3.6: Threat model message sequence chart with different settings.

Upper chart: Screenlogging attack where screenshots are triggered by a screen capture command and sent at a regular frequency f after being compressed. Lower chart: Other settings where screenshots are triggered by a specific event on the victim's machine, such as opening a specific website, and taken or sent irregularly. In the illustrated case, the malware program performs local processing to extract data from the screenshots and then sends only the extracted information.

Exclusion of session replay

Session replays let app developers record a screen and play it back to see how its users interacted with the app to determine whether something did not work or there was an error. Every tap, button-push or keyboard entry is recorded – effectively screenshotted - and sent back to the app developers [191].

Some firms, such as GlassBox, offer customers the possibility to record the screens of the users of their apps [192]. This has raised serious concerns as it results in applications sending unencrypted screenshots containing sensitive data such as passport or credit card numbers [191].

As dangerous as session replay can be, we exclude it from the scope of this thesis because the screenshots are not taken by an adversary but by the application itself.

Exclusion of document object model-based attacks

The DOM is an API for valid HTML and well-formed XML documents. As mentioned above, a DOM defines the logical structure of documents and the manner in which a document is accessed and manipulated [193].

In most browsers, the content of webpages is represented using the DOM. Some attacks consist in getting information from by abusively accessing the DOM of webpages.

These kinds of attack are excluded from our work, even if they permit stealing some data targeted by screenloggers. Their operational mode is limited to browsers as opposed to screenloggers.

4 | Threat Analysis

To propose countermeasures and effectively combat screenloggers, it is essential to understand the details of their operating process, including how they take screenshots (Section 4.2), store (Section 4.3) and exfiltrate (Section 4.4) them. This understanding necessarily requires a thorough study of existing spyware (Section 4.1).

Analysing the behaviours displayed by screenloggers in the wild allows for the definition of completeness criteria (Section 4.5) for the malicious dataset, which will be used for detection.

4.1 Methodology

The first step in analysing the behaviour of screenshot-taking malware was to gather high-level information from 100 security reports (Section 4.1.1) recovered from the MITRE ATT&CK database [14]. Next, a set of novel criteria that can discriminate screenshot-taking malware from each other was identified (Section 4.1.2).

4.1.1 Extensive study of security reports

The MITRE Corporation was chartered in 1958 as a private, not-for-profit company to provide engineering and technical guidance for the US federal government. The MITRE ATT&CK Framework was created by MITRE in 2013 to document attacker tactics and techniques based on real-world observations. The ATT&CK knowledge base is used as a foundation for the development of specific threat models and methodologies in the private sector, government and the cybersecurity product and service community [14].

One of the documented attacker tactics is 'collection' (TA0009), by which 'the adversary is trying to gather data of interest to their goal' [14].

Screen capture (T1113) has been identified as one of the techniques that can be used by the adversary to implement a collection tactic. As a result, 127 reports from security firms such as Symantec have been compiled and referenced on a webpage [14]. These reports cover about 103 different screenshot-taking malware programs targeting desktop environments. The list is regularly updated (at the time of writing, the last update was 24 March 2020).

Therefore, we assume that by analysing these reports, we can have a realistic view of the behaviours displayed by screenloggers in the wild.

4.1.2 Proposed taxonomy

Although the MITRE ATT&CK database provides the most exhaustive list of screenshot-taking malware, it fails to give insight into their behaviour. Indeed, merely referencing security reports cannot facilitate gathering sufficient knowledge to develop effective countermeasures.

As a result, we had to thoroughly analyse the 127 security reports. However, as they emanated from many different security firms, they described the malware in a heterogeneous way. The information given by a security firm can greatly differ from the information found in a report provided by another entity.

We propose a taxonomy that systematises the description of screenlogger behaviours. This systematisation is based on a set of novel criteria that can discriminate among screenshot-taking malware programs. While analysing the 127 security reports, we noted down the different behavioural aspects mentioned in each individual report. When a given aspect was present in more than 45% of the reports, it was integrated as a criteria in our taxonomy.

Table 4.1: Criteria of completeness.

Criterion	Options
Windows library used to take screenshots	- Windows graphics device interface - Desktop duplication API
Screenshot-triggering	- Frequency - Punctual command - Application of interest - Mouse clicks/Keyboard presses - Unique screenshot upon infection
Captured area	- Whole screen - Window of interest - All overlapping windows - Around the mouse pointer - Configurable area
Place where the screenshots are stored	- HDD - Memory
Image file format	- JPG - PNG - BMP
Encryption	- Yes - No
Communication protocol	- TCP - HTTP/HTTPS - FTP/FTPS - SMTP/SMTPS
Screenshots sending triggering	- Directly after the screen capture - Frequency - Command - After a given number of screenshots

The criteria we used are recapitulated in Table 4.1.

Our criteria cover the main operating steps of the screenlogging process: screenshot-taking, screenshot storage and screenshot exfiltration.

Screen capturing Depending on the adversary's objective, screenshots can be taken at different moments. Moreover, the functionalities offered by the Windows operating system for screenshot-taking offer several possibilities. The criteria we use at the screen-capturing stage are:

- Used API: The traditional way of taking screenshots on Windows is to use the GDI API. More precisely, the content of the DC is retrieved using a first API call. Then, a destination bitmap compatible with the retrieved DC is created using another API call. Finally, the content of DC is copied into the destination bitmap. When using GDI, every time a screenshot is taken, the updated DC must be retrieved, and the sequence of API calls must be called again. Using GDI is therefore not suited for an application wanting to have a real-time view of the screen.

  Starting from Windows 8, the DD API was introduced. This API was created precisely to make high-frequency screenshot-taking less cumbersome.

  Knowing what API is used by malware provides information about its behaviour. On the one hand, malware using the DD API intends to monitor all the victim's activity. This is the scenario, for example, in the case of RATs. On the other hand, malware using the GDI API might be looking for more flexibility in the screenshot-taking.

- Screenshot-triggering: A first sub-criteria is the need for the malware to receive a command from its C&C server to start taking screenshots. Indeed, the modalities of screenshot-triggering can either be defined in the malware files upon infection or contained in a command received from the C&C server. Two kinds of C&C commands can be found: a command for a punctual screenshot (each time the adversary wants a screenshot, a new command is sent) or a command for continuous screenshots, which defines the events that trigger the screen capture.

  The second sub-criteria regards the modalities of screenshot-taking. As seen above, these modalities can be found in the malware files or in a command for continuous screen capture. Depending on the attacker's goal, different

events can trigger the screenshot-taking. For instance, malware targeting a specific application can be set to take screenshots only when the application is open. Malware wanting to minimise the number of screenshots taken might track user events, such as key presses or mouse clicks that indicate screen content has changed. For malware intended for real-time monitoring, a frequency will be defined at which screenshots will be taken. Multiple modalities can be used simultaneously. For example, malware designed for the theft of banking credentials entered using a virtual keyboard might trigger screenshots only when banking applications are open, and there is a mouse click (indicating that the user has pressed a key).

- Captured area: The GDI API offers several possibilities regarding the screenshot's content. Indeed, two distinct functions can alternatively be called to retrieve the DC: GetDC and GetWindowsDC. On the one hand, GetDC allows for capturing a subpart of the screen if the coordinates of a rectangle's corners are given as parameters. If GetDC is called with the NULL argument, the DC of the whole screen is returned. On the other hand, GetWindowsDC obtains the DC of a given window if a handle to the window is provided. Even a window that is not in the foreground can be captured. If the NULL argument is passed instead, the DC of the full screen is returned.

  Again, depending on the attacker's goal, different screen areas can be targeted. For instance, when specific applications are targeted, it is possible to capture only the corresponding windows. On the other hand, generalist malware programs not looking for specific information might tend to take the whole screen.

Moreover, the captured area has a direct impact on the screenshot size (in bytes) and thus on the storage and network usage. If the malware author wants to avoid notice, a limited area of interest can be captured (e.g., the area around the mouse pointer in the case of a virtual keyboard).

Screenshot storage Once the screenshot is taken, an image file can be created, stored and manipulated on the victim's machine. The criteria we consider at this stage are:

- Compression algorithm: Many different image compression algorithms can be used by screenloggers. Like the captured area, the format chosen for images representation has a direct impact on storage and network usage.
- Storage media: The image file can either be stored in memory for short-term use or in the disk for long-term use. Memory storage might imply that the screenshot will be sent over the network shortly after it was taken, whereas disk storage is more adapted for local processing of the screenshot (e.g., using OCR tools) or delayed sending.

Screenshot sending To exploit the screenshots or to send the results of local exploitation, network packets are necessarily sent to the malicious server. The criteria we use at this stage are:

- Communication protocol: Screenshots can be sent using a wide range of application layer protocols or can directly be sent on the transport layer.
- Screenshot sending triggering: The adversary can develop different exfiltration strategies depending on their goal. When real-time is a key aspect of the attack, screenshots will be sent directly after they are taken. To avoid detection, several strategies can be used to group screenshots together instead

of sending multiple network messages: timers, buffers, command-triggered sending and more.

This criterion is important because it determines the network traffic characteristics (size of the packets, frequency of sending, etc.).

- Encryption: Even if confidentiality and authentication might not be relevant to malware authors, there are many possible algorithms that can be used to encrypt screenshots.

  Knowing the encryption methods which are used to send the data can be important to detect the exfiltration of image type files, which will be more easily discovered if there is no or very simple encoding (base64, ZIP, RAR).

4.2 Screen capturing

4.2.1 Used API

On Windows systems, which constitute the target of this work, two main libraries can be used, Windows GDI [194] and DD API. Our analysis showed that existing malware does not seem to use the functionalities offered by DD API. However, some of them (i.e., Azorult [195], Bandook [196], RTM [197] and Proton) use VNC, a legitimate remote desktop manager which uses DD API and GDI.

4.2.2 Screenshot-triggering

As illustrated in Figure 4.1, a vast majority of screenloggers (67%) wait for a command from the C2 server to start capturing the screen. This imposes an important constraint on our malicious dataset: we must gather both the server and the client

parts of the screenloggers to be able to simulate the triggering of the screenshot functionality by sending commands.

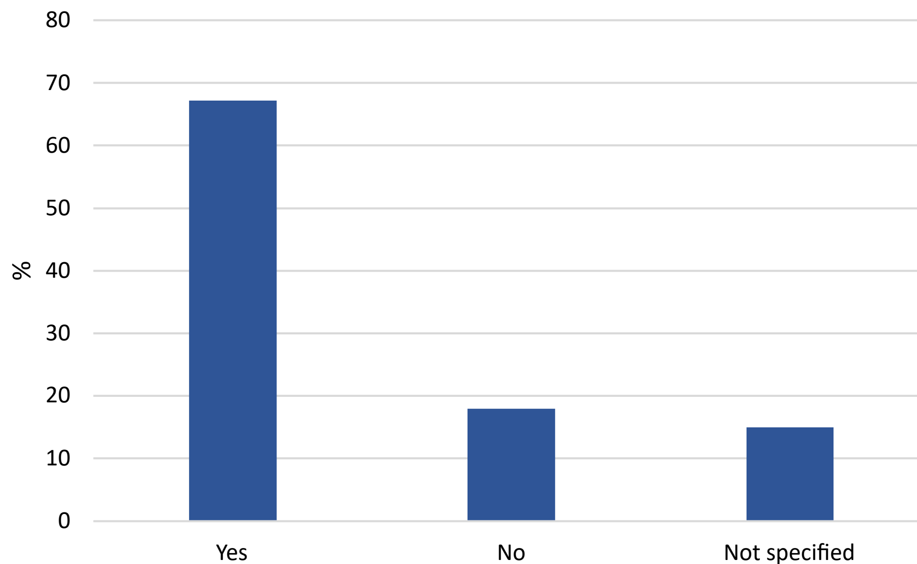


Figure 4.1: Need for a screenshot command to start screen capturing (in the malware of Mitre [14]).

Independently of the need for a command to start taking screenshots, different events can cause screen capture triggering. These events can be preconfigured in the malware's compiled files, or they can be set on the fly upon the reception of a command once a victim has been reached. For instance, Remexi is preconfigured to take screenshots when some applications of interest are opened after a configurable number of mouse clicks, and does not need to receive any command to specify these screenshot-taking parameters [198]. In comparison, Biscuit [199] and PowerSploit [200] take screenshots at a configurable frequency, which is set in a command received from the C&C server.

As illustrated in Figure 4.2, the different screenshot-triggering events that were identified are frequency (35%), punctual command (38%), application of interest (9%), mouse clicks (3%) and unique screenshot upon infection (5%).

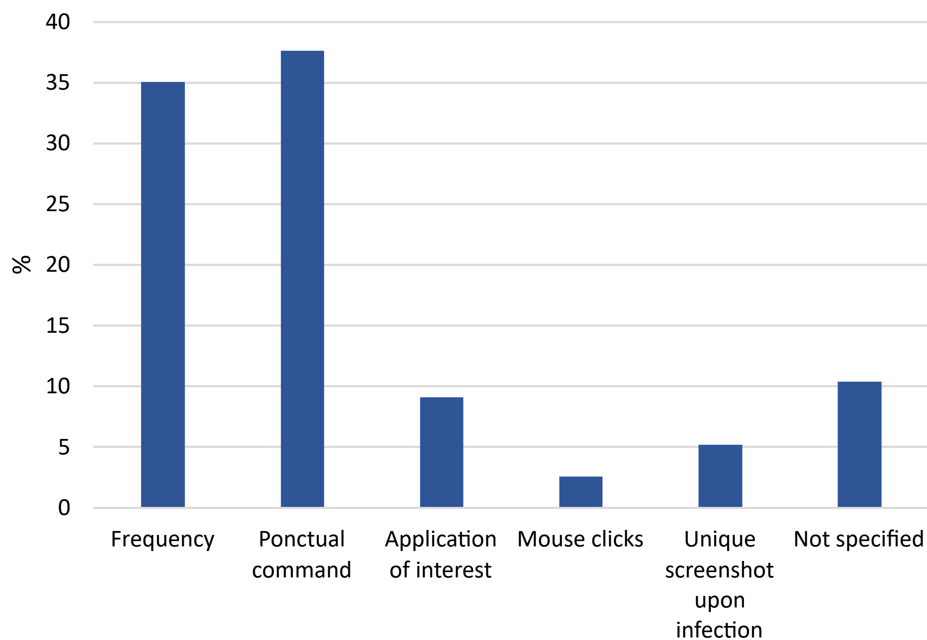


Figure 4.2: Screenshot-triggering (in the malware of Mitre [14].)

Regarding frequency, the observed screen-capturing frequencies range from 2 s (Cross RAT [201]) to 15 min (Prikormka [202]). Malware offering remote desktop functionality has a higher frequency, which may reach 30 frames per second (Xtreme RAT [62]). The screenshot frequency may vary over time: for example, Prikormka has a normal frequency of 15 min but it increases to reach 5 s when VoIP applications such as Skype or Viber are open.

Some screenloggers allow on-demand screenshots, having a specific command to take one screenshot at a time (this is the category 'punctual command' in Figure 4.2). Most malware offering a remote desktop functionality also offer an indepen-

dent punctual screenshot command (e.g., Azorult [195], Carbanak [21], NetWire [64]).

A few malware types have different behaviours, as they capture screenshots in response to mouse clicks (e.g., Remexi [198]) or by the launching of a target application (e.g., Catchamas [203]). This last category of screenlogger waits for a specific application to be open by looking at window titles (Catchamas [203], Remexi [198], RTM [197], T9000 [204]) or the list of running processes (e.g., Biscuit [199] or Flame for instant messengers [205]).

Finally, some malware take one screenshot for reconnaissance during their entire execution to see if the victim is worth infecting (e.g., Cannon [206], Zebrocy [207]).

4.2.3 Captured area

Regarding the screenshots area, even if this information is often unavailable (no information for 55% of the security reports), it follows from our study that more than 37% of screenloggers capture the entire screen without targeting a particular area or window (Figure 4.3). Nevertheless, three other operating modes are represented, even if it is in small proportions. These are the capture of a target window (T9000 [204]), of all overlapping windows (InvisiMole [208]) or of a delimited area of the screen (ZeusPanda [209]). Interestingly, the Remexi malware [198] proposes a parameter for full screen or only active window screenshots.

4.3 Screenshots storage

4.3.1 Image compression

It emerges from our analysis that the formats preferred by screenloggers are, without surprise, the most common among the general public. In fact, 43% of screenloggers for which information is available use the JPG format and 32% the PNG format. However, some other malware types opt for other formats, such as BMP (12%) and, to a lesser extent, AVI, WCRT and RAR, as shown in Figure 4.4.

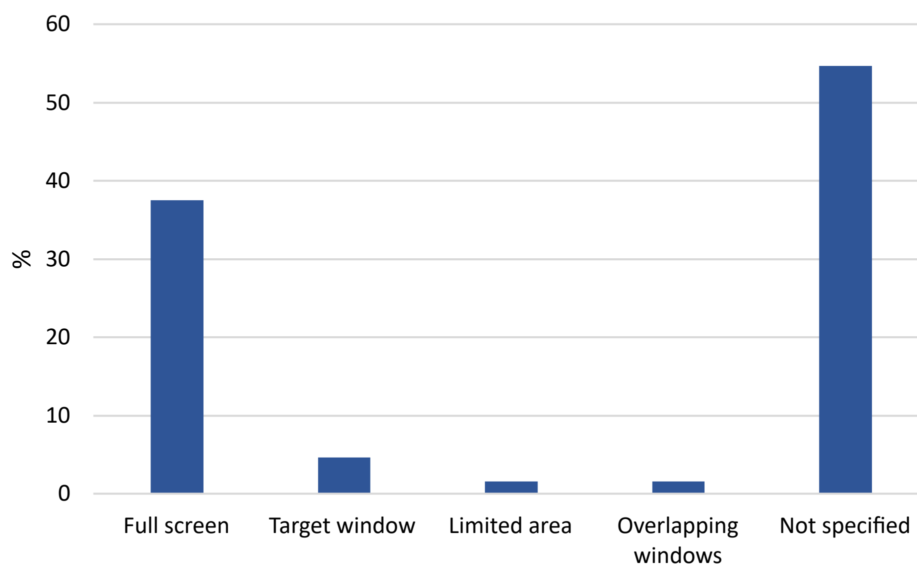


Figure 4.3: Captured area.

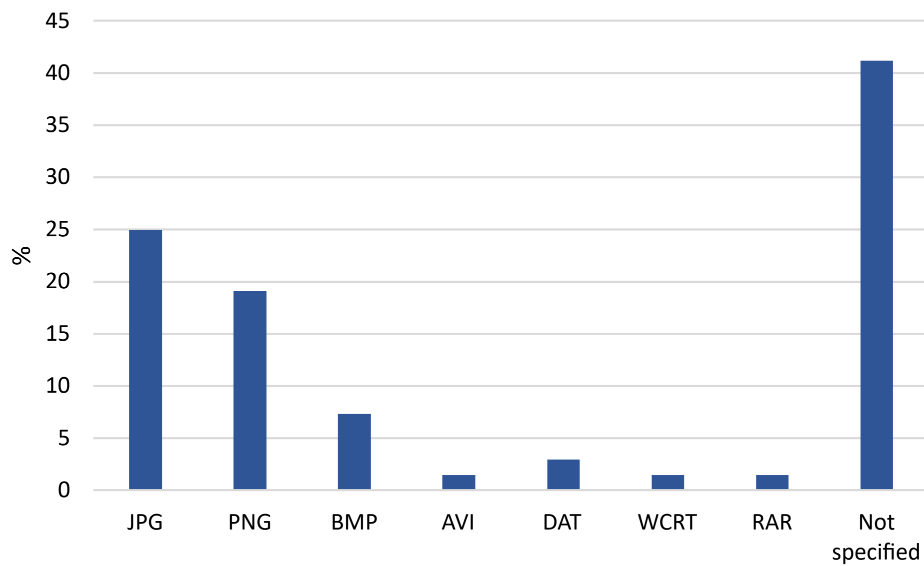


Figure 4.4: Image files format.

4.3.2 Storage media

The information of whether malware uses memory representations or persistent disk storage of image files is unfortunately unavailable for almost half of the considered screenloggers. However, hard drives seem to be the most frequently used storage means, as illustrated in Figure 4.5.

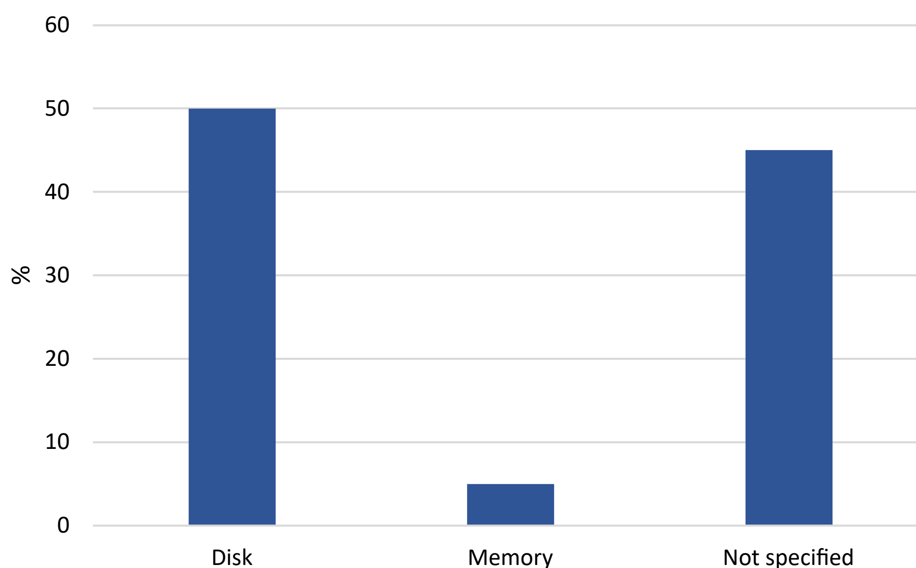


Figure 4.5: Files storage.

4.4 Screenshots exfiltration

4.4.1 Communication protocol

All the malware programs of this study transmit the captured screenshots to remote and malicious servers. This even includes malware taking only one screenshot to hide the on-screen activity, such as FruitFly.

The communication protocols used cover a wide spectrum (Figure 4.6). 40% of screenloggers for which the protocol is identified use HTTP, which increases their chances of going unnoticed in the large HTTP flow passing through almost all machines. HTTPS, FTP, SMTP and SOAP complete the list of used protocols. Note that a non-negligible proportion of malware (20%) does not have an application layer network protocol and simply uses the transport layer by sending TCP

packets. Finally, only one of the studied screenloggers, Biscuit, uses a proprietary and non-standard network protocol [199].

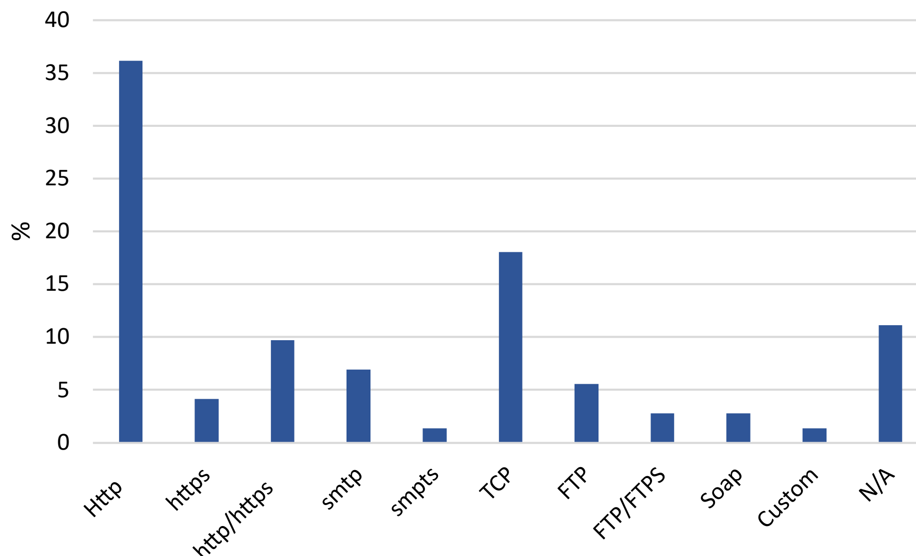


Figure 4.6: Communication protocol.

4.4.2 Encryption

The encryption technique used by malware to exfiltrate data was unavailable for almost half of the malware programs studied. This result may be because the information is difficult to obtain. Another possible explanation could be that some malware programs do not encrypt their data.

However, for the malware for which this information was available, we observed that the encryption techniques are highly diverse. As shown in Figure 4.7, each malware program designs its own encryption method, often by combing several techniques. The encryption methods are more or less sophisticated, for instance, some malware only use an eXclusive OR (XOR) operations (e.g., T9000 [204]) or base64 encoding (e.g., POWRUNER [210]). Others combine several encryp-

tion algorithms, such as Chopstick, which encrypts communications using RC4 and TLS, or BadNews [211, 212], which applies a rotate right (ROR) operation followed by a XOR, conversion to hexadecimal and base64 encoding.

Note that Proton and Micropsia [213] simply send a ZIP/RAR archive, which may be protected by a password.

BRONZE BUTLER uses two alternative encryption techniques: RC4+base64 and AES [214].

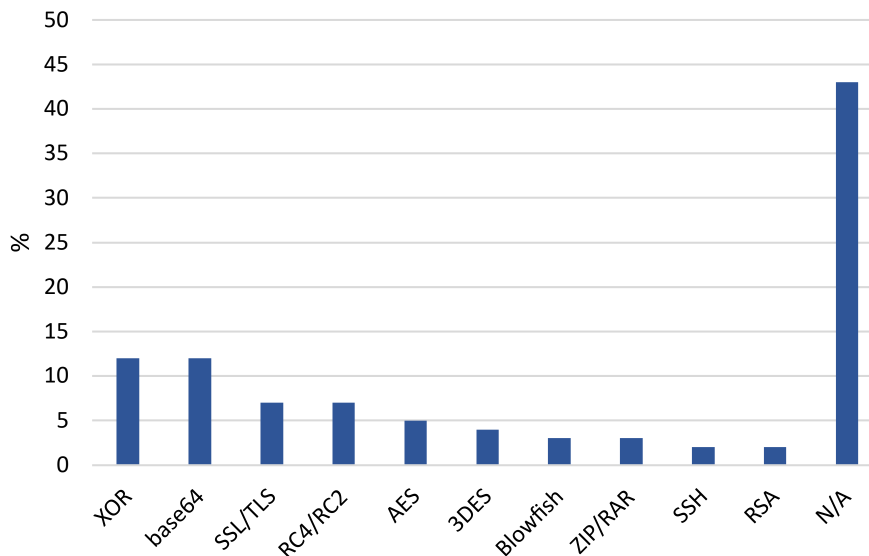


Figure 4.7: Image files encryption.

4.4.3 Screenshots sending triggering

There were no reports of malware locally exploiting the screenshots using OCR tools. The security reports we analysed all mention that the screenshots are sent as is, without local processing.

Regarding the event triggering screenshot sending, most of the time this information is not available. However, we were able to identify four main possible behaviours among the studied screenloggers.

The first one, which is also the most common, is the immediate sending of the captured image directly after the screen capture (e.g., Magic Hound [215], RTM [197]). The explanation could be that many of these malware programs use screenshots for a real-time purpose, such as remote control on the victim's machine or real-time observation of the victim's activity.

The three other behaviours that were found include sending screenshots at a regular frequency (e.g., Micropsia [213], Flame [205], Rover [216]), when a specific command is received from the C2 server (e.g., Biscuit [199], Powruner [210]), or each time a predefined number of screenshots is taken (e.g., RTM after six screenshots [197], Pteranodon after a configurable number of screenshots [217]).

4.5 Synthesis: Completeness requirements

As pointed out in Section 2.1, no existing dataset is dedicated to screenshot-taking malware. We only found a dataset containing two screenlogger samples with limited behaviours.

Therefore, to propose an effective detection approach against screenshot-taking malware, it was necessary for us to construct our own dataset.

The results we obtained by analysing the 127 security reports of the MITRE ATT&CK database enabled us to define completeness criteria for our malicious dataset.

We propose two definitions of the term 'completeness': 'behavioural' completeness (Section 4.5.1) and 'proportional' completeness (Section 4.5.2). These two

definitions are based on different assumptions concerning the MITRE ATT&CK database.

4.5.1 Behavioural completeness

A first definition that can be given to the word 'completeness' is that our dataset will be complete if its malware samples display all the behaviours found in the security reports, regardless of their proportions. This definition assumes that the MITRE ATT&CK database is exhaustive enough to encompass all the behaviours found in the wild but not necessarily in representative proportions.

The different behaviours that our dataset should include based on this first definition are displayed in Table 4.2.

Table 4.2: Criteria of completeness.

Criterion	Options
Windows library used to take screenshots	- Windows graphics device interface - Desktop duplication API
Screenshot-triggering	- Frequency - Punctual command - Application of interest - Mouse clicks/Keyboard presses - Unique screenshot upon infection
Captured area	- Whole screen - Window of interest - All overlapping windows - Around the mouse pointer - Configurable area
Place where the screenshots are stored	- HDD - Memory
Image file format	- JPG - PNG - BMP
Encryption	- Yes - No
Communication protocol	- TCP - HTTP/HTTPS - FTP/FTPS - SMTP/SMTPS
Screenshots sending triggering	- Directly after the screen capture - Frequency - Command - After a given number of screenshots

4.5.2 Proportional completeness

Table 4.3: Screenloggers behaviours (samples selected from [14]).

		Used Solution	Malware (Mitre) %
Screen capture	Screenshot capture library	GDI	59
		DD API	-
	Screenshot capture triggering	Frequency	35
		App. of interest	9
		Mouse clicks /User trig.	3
		Punctual command	38
		Unique capture upon infec.	5
	Frequency		2s to 15mn
Format and storage	Format	JPG	25
		PNG	19
		BMP	7
		Video	1
		Other	6
	Storage	Memory	5
		Disk	59
	Captured area	Full screen	37
		Coordinates	2
		Difference	-
		Other	6
Exfiltration	Screenshot sending trig.	Real-time flow	25
		Remote cmd	6
		Scheduled	10
		Other	13
	Encryption?	No	-
	Files sending?	No	-
	Comm. protocol	HTTP	36
		HTTPS/FTP/SMTP/RFB	36
		Proprietary	1
		TCP	18

The second definition we can give to the word 'completeness' is that our dataset will only be complete if the behaviours at the different stages of the screenlogger operating process are represented in the same proportions as in the security reports. These proportions can be found in Table 4.3.

The assumption made is strong as we assume that the security reports compiled by MITRE provide an accurate overview of the behaviours exhibited by screenloggers, with the same proportions.

4.5.3 Discussion

When assessing the completeness of our dataset, we had to choose between the two definitions given above.

On the one hand, the assumption made for the behavioural completeness definitions seems realistic as the MITRE ATT&CK database is renowned for documenting different kinds of attacks and is updated regularly.

On the other hand, the assumption made for the proportional completeness definition seems less likely to hold for two reasons. The first is that much information is missing. For example, only 45% of the security reports indicated what area of the screen was targeted by the screenshots. The second reason is that, even if the reports were complete, there are many more than 103 screenshot-taking malware programs in the wild, and most of them probably use the most naÃ¯ve and unsophisticated behaviours. However, one could argue that the attacks referenced in the MITRE ATT&CK database are the ones with the most critical consequences and that we precisely aim to detect the most stealthy and unusual screenlogger behaviours. Moreover, as the detection is based on ML algorithms, if the dataset is overwhelmingly composed of basic and naÃ¯ve behaviours, the features that

would be selected would not allow for the detection of more sophisticated variants, which will be under-represented.

Therefore, even if it might appear less realistic than the first definition, we chose to implement the proportional completeness definition in our dataset.

5 | Dataset Construction

To effectively detect spyware using the screen capture feature, it is essential to understand and identify the similarities and differences between their behaviour and that of legitimate applications. Thus, the first step of our approach aims to constitute a significant sample of legitimate applications and screenlogger-type spyware, with varied and representative behaviour.

In this chapter, we present our methodology to build the first dataset dedicated to screenshot-taking malware (Section 5.1) and legitimate screenshot-taking applications (Section 5.2).

The dataset is not only composed of malware and legitimate samples but also contains execution reports. These reports monitor two aspects of the samples' execution: API calls and network behaviour. They are used to train and test our detection models.

5.1 Malicious dataset

After presenting the methodology we used to collect and analyse the malware samples of our dataset (Section 5.1.1), we discuss the main challenges faced in the construction of the malicious dataset and how we proposed to address them (Sections 5.1.2 and 5.1.3).

5.1.1 Data collection and analysis

Our approach is based on collecting a significant number of known screenshot-taking malware samples, then executing them in a safe environment to analyse their behaviour.

The names and hashcodes of our samples were collected from MITRE [14], which has the advantage of centralising and listing 103 malware programs with the screenshot functionality. The hashcodes can be found in the indicators of compromise section of the security reports. Based on these hashcodes, we were able to collect 600 spyware samples from VirusShare [85], AVCaesar [51], Malshare [218] and VirusSign [219]. We also collected some screenlogger source codes on open archives, such as Github.

Given the nature of the software studied, the execution was carried out in a secure environment. Subsequently, their behaviour was analysed using dedicated software.

Spyware were executed in a Windows 10 virtual machine (INetSim internet configuration) [220]. After each execution, the machine was reinitialised from a snapshot of the non-infected initial state.

To analyse the spyware, we used Cuckoo sandbox [79], which allows generating reports on the API calls made by malware programs, and on their network traffic in pcap files. We also used Wireshark [221] as well as API Monitor [222], which provides more precise information than Cuckoo on API calls (log with each call and when they were made).

5.1.2 Challenge 1: Ensuring that screenshots are taken

As we target screenshot-taking malware, to propose a meaningful detection approach, it is necessary to ensure that the execution reports generated actually display screen capture operations.

While employing a novel technique based on the monitoring of API call sequences, we realised that our samples were not taking screenshots during ex-

ecution. We then investigated the reasons for this unexpected outcome, which allowed us to propose a solution based on widely used screenshot tools.

API calls hooking to detect screen-capturing actions

To ensure that the collected samples were taking screenshots, it was necessary to find a sequence of API calls that characterises screenshot-taking. This sequence must be precise enough that non-screenshot cases would be excluded and simultaneously not too precise to avoid false negatives (excluding cases where screenshots were actually taken). The difficulty lies in the fact that there is no unique screenshot function proposed by Windows APIs but rather a sequence of 5–6 functions that can be used alternatively with other functions. Moreover, each function in the sequence can individually be used in other contexts. Thus, it is impossible to deduce that a screenshot was taken by only looking at a unique function.

As explained in Section 4.1.3, two main libraries can be used to take a screenshot: the Windows GDI and the DD API, which replaced mirror drivers from Windows 8 onward. To define the criteria characterising a screen capture, screenloggers and legitimate screenshot-taking applications were analysed. Results showed that different API call sequences can be used for screenshot-taking. We therefore had to make flowcharts with different alternatives at each step (Figure 5.1). The charts were then transcribed into a script that takes the API call reports as an input and returns information such as the number of screen captures taken and their frequency (if the screenshots are taken at a regular time interval).

Using the flowcharts, for the GDI library, we identified a screen capture by the call of one function that retrieves the content of the screen (GetDC, GetWindowsDC, CreateDCA or CreateDCW), followed by the call of the BitBlt or the StretchBlt function. The script ensures that BitBlt's hdcSource parameter is the value

that was returned by the function capturing the screen's content. Sometimes, the GetDC function is called only one time at the beginning of the screenshot session and the subsequent functions are called multiple times with the same hdcSource parameter.

As the functions in the sequence take as parameters the return values of the previous functions, it is impossible for them to be called out-of-order. Moreover, as the return values are kept in memory until they are used as parameters, the screenshot is detected even if the API calls are spaced in time.

A rarer sequence using GetDIBits, CreateDIBitmap, SetDIBits and CreateBitmap functions was found in some applications (e.g., JoinMe [55] and AnyDesk). When this sequence is used, BitBlt is called with small 'size' parameter values, whereas, in the above sequence, it is called with the size of the screen.

Finally, a stealthier way of taking screenshots using GDI is to call only the getDC function for each screenshot, with the GdipCreateBitmapFromScan0 function, from the GDI+ API, in the callstack. Indeed, api calls made in the callstack are not recorded by API Monitor. However, we were able to make our scripts detect this way of taking screenshots by noticing that the getDC function is called by the gdiplus.dll module, which is not the case in other situations.

For the DD API, a screenshot is identified by the use of the AcquireNextFrame, GetFrameMoveRects and GetFrameDirtyRects functions, which allow for capturing only the part of the screen that changed compared to the previous frame. This library is mainly used by the applications that capture the screen continuously (screen sharing, remote control, screencasting).

Zhao et al. proposed a method for screenlogger detection and also identified criteria based on Windows API calls to characterise a screen capture operation [223]. Their solution was to examine the GetWindowsDC and BitBlt function calls. This

approach is similar to the criteria we identified but does not consider the other equivalent functions (StretchBlt, GetDC, etc.) nor the other libraries that can be used (DD API). Thus, examining only the GetWindowsDC and BitBlt function calls does not allow for the identification of all the screen-capturing applications. Moreover, the parameters of the function are not verified, particularly the hdcSource parameter of the BitBlt function, which must refer to the screen content. Otherwise, if the hdcSource parameter does not correspond to the DC of the screen, it means that BitBlt was used for a different purpose and some applications that do not capture the screen can be identified as taking screenshots, which is problematic as well.

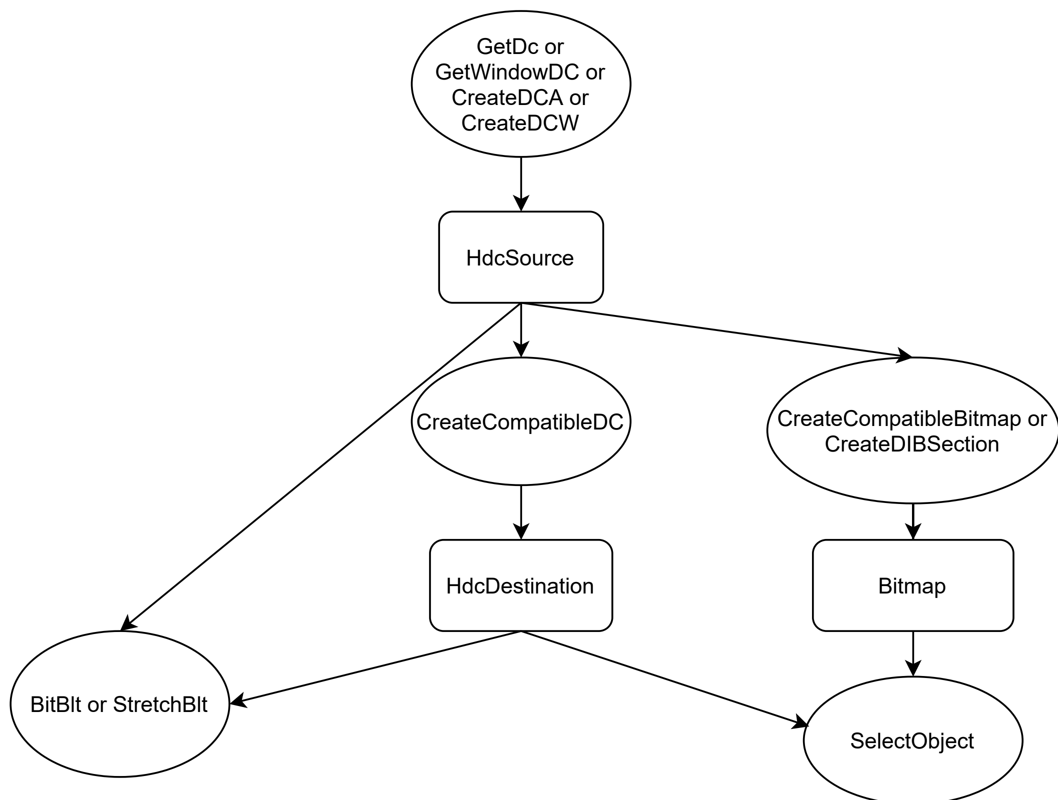


Figure 5.1: Windows screen-capturing functions.

Problem

By running our scripts on the found samples, we realised that no sample but one (over 600 samples) was actually taking screenshots. Different reasons can be put forward to explain this unexpected result.

- Many malware programs infect their targets through different steps. The screenshot module is often not present from the beginning and is rather downloaded from the server afterwards. For example, the Prikormka malware has a 'downloader module' that downloads other modules that are not present at the time of infection [202]. Silence downloads a dropper which will communicate with the server to obtain the screenshot plugin [224]. However, almost all the malware servers are currently dead. Even if they were not, we would not allow the samples to connect to the network for obvious security reasons.
- Some malware programs are fitted with anti-sandbox /anti-analysis capabilities. For example, ZeusPanda [209] looks for files showing the presence of Virtual Box, VMware or Wireshark among many other tools on the infected device. It also looks for these tools in the list of running processes. If ZeusPanda realises that it is within an analysis environment, it does not run.
- Sometimes the screenshot settings governing whether screenshots can be taken, when, and targeting what applications were defined by the attacker before infection and impossible to modify afterwards with only the sample at our disposal (particularly KeyBoy [225] and AgentTesla [226] malware).
- In most cases, the screenshot functionality is triggered by a command coming from the attacker. However, as pointed out earlier, we do not allow the

malware samples to receive network messages. Moreover, analysis reports rarely indicate the command used to trigger the screenshots. Even when they do, there is insufficient information to simulate the command (i.e., to what HTTP field does it correspond, what are its parameters, on which port should it be sent?). Finally, the samples available in current datasets only include the client part of the malware, which is run on the victim's machine and not the attacker's, which sends commands.

As the samples available on current malware datasets (VirusShare, VirusSign, AV-Caesar) were not taking screenshots during their execution, it was necessary to do more extensive research to find malware whose screenshot functions could be enabled.

Solution

The solution we propose to this challenge results from the observation that malware authors often do not bother implementing the screenshot functionality themselves. Instead, they reuse RATs or 'Pentest tools' widely available on the internet. The only element that varies between different malware is therefore the infection medium, which is not relevant for this study. Several well-known cybercriminals, having realised significant attacks, employed these kinds of tools. By way of illustration, and in a non-exhaustive manner, one could cite:

- The FIN7 group (particularly the GRIFFON and HALFBAKED malware) uses Carbanak for screen capture [34]
- The Group5 group uses njRAT and Nano Core RAT [227]
- The CopyKittens group uses CobaltStrike and Meterpreter [228]
- The Socksbot malware uses QuazarRAT and BADNEWS [229]

Analysing these malicious tools seems a reasonable attempt to cover the majority of screenshot-taking malware. As they are widely available, we were able to get working samples (with the client and server parts) and even source codes where available. We constructed a dataset dedicated specifically to screenshot-taking malware, containing 118 samples. Some were source codes that we had to compile and make operable. We ran each of them to ensure that the screen-capturing function was triggered. For that, it was necessary that each selected malware be composed of executable client and server parts to allow us to trigger the screen capture ourselves.

These malicious tools have then been analysed to gather the same type of information as from the MITRE security reports. When we had the source code, we directly looked there. If not, we ran the executable files. Given the dangerous nature of the studied software, their execution was carried out in a secure environment, with the client and server parts run on two different VirtualBox virtual machines. The network was configured by creating a virtual interface network named vboxnet0. The two machines were configured with host only network to isolate them from the real interface network and to allow communication between them.

5.1.3 Challenge 2: Representativeness of the dataset

This dataset is, to the best of our knowledge, the first one dedicated to screenshot-taking malware and is available on Github [230].

We used the source codes and execution reports at our disposal to analyse the malware samples of our dataset using the criteria presented in Section 4.1.2 (Sections 5.1.3.1, 5.1.3.2, 5.1.3.3).

Then, to assess the 'proportional' completeness of our dataset (defined in Section 4.5.2), we compared the obtained results with the proportions we obtained from the malware programs listed by MITRE (Section 5.1.3.4).

To address the observed lack of representativeness, our solution consisted of implementing a 'screenlogger generator', which was used to generate malware samples with the missing behaviours (Section 5.1.3.5).

Screen capturing

Used API The malware of our dataset exclusively uses the GDI library to capture screenshots. Therefore, this does not include the DD API that may be used by some malware programs, as seen in Section 4.2.1.

Screenshot-triggering All the malware samples composing the dataset must receive a command to start taking screenshots. This corresponds to the observations made in Section 4.2.1, as most malware had to wait for a command to capture the screen. However, the opposite behaviour (start taking screenshots automatically) is not represented.

Regarding the event triggering the screen-capturing functionality, the two most frequent behaviours encountered are the capture of the screen at a given frequency or on-demand upon receipt of a specific command allowing for one screenshot to be taken (Figure 5.2). Several screenloggers, such as Xtreme RAT, SpyNet and NetWire, offer the two possibilities.

Remcos constitutes a particular case in the dataset because its screenshots are triggered by the occurrences of some target keywords in the titles of the opened windows.

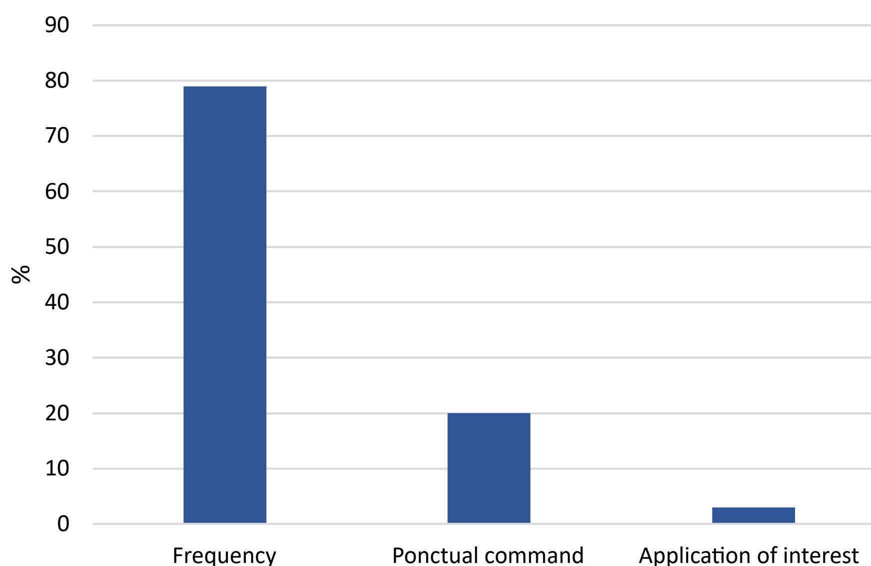


Figure 5.2: Screenshot-triggering.

Two behaviours observed in the previous section are not represented here: screenshots triggered by mouse clicks and unique screen capture during the whole execution for reconnaissance purposes.

Regarding the values of the screen-capturing frequencies, they ranged from 17 ms to 67 s, and several screenloggers had a configurable frequency (e.g., PowerShell-RAT, Powersploit). Interestingly, thanks to the source codes, we were able to observe that some malware use a random number between each screen capture and, therefore, do not take the screenshots at exactly equal time intervals. For instance, Carbanak draws a random number between 15,000 ms and 30,000 ms at each capture, and CtOSRAT draws a random number between 17 and 31 ms.

Note that all malware samples in our dataset take screenshots using hidden processes with no user interaction. Some of them even inject themselves in legitimate processes such as the default browser (Figure 5.3).

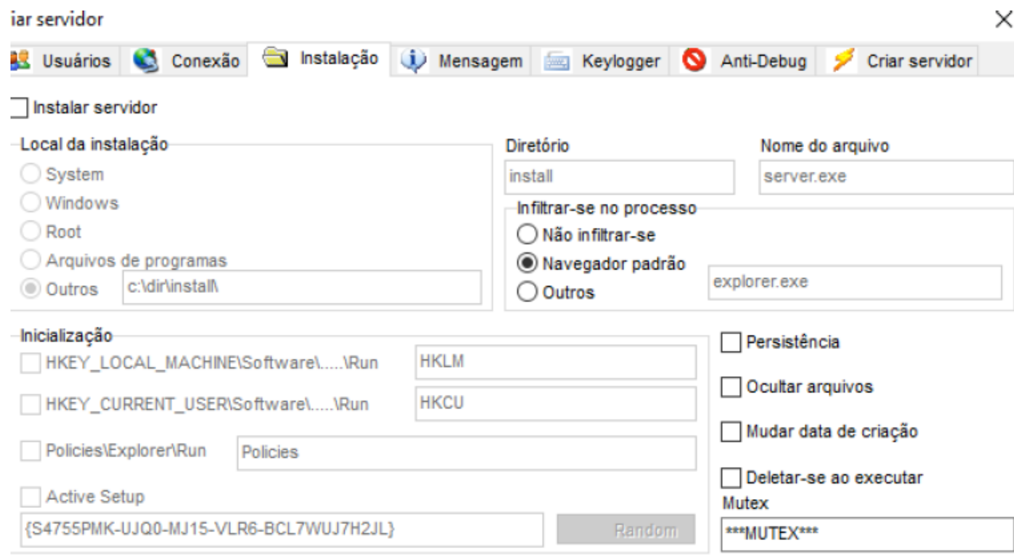


Figure 5.3: Malware infiltrating the default browser process (Navegador padrao).

Captured area Regarding the captured area, as shown in Figure 5.4 a majority of the selected malware (86%) captures the entire screen at each shot. This corresponds to the observations made in Section 4.2.3. Capturing the whole screen might be used to recover as much information as possible. Relatively advanced software can subsequently be used to extract relevant data after the transfer to the malicious servers.

However, 14% of malware have smarter and more optimised behaviours as they capture either a specified zone using its coordinates (7%, e.g., SpyNet, Xtreme RAT) or only the difference (changes) between two successive screens (Gh0st). Moreover, we found a behaviour that we did not observe in Section 4.2.3, which is the capture of the zone around the mouse click, an option proposed by Pupy. This enables sending smaller, and thus stealthier, packets, and can be useful in attacks targeting online banking users who enter their password using a virtual keyboard, as proposed by many banks [41].

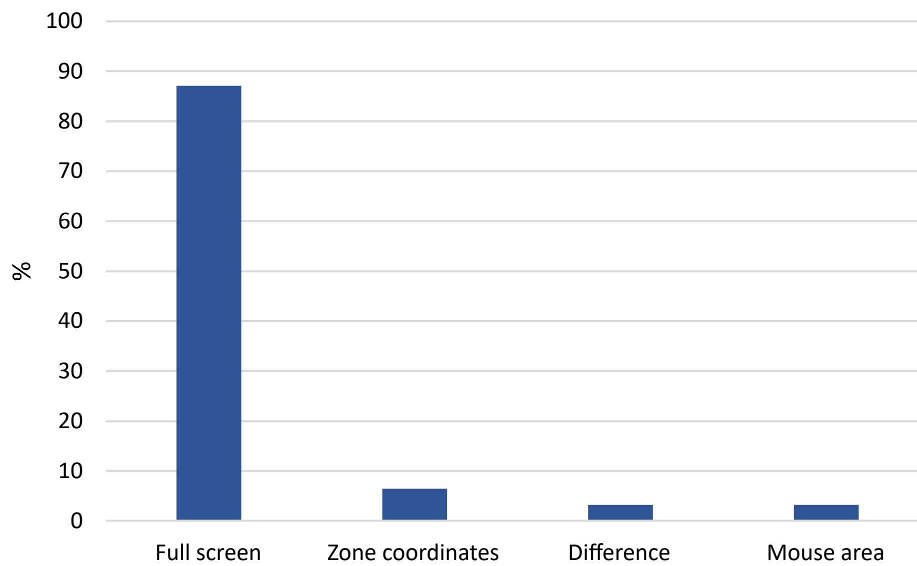


Figure 5.4: Captured area (in the malware of the dataset).

Screenshot storage

Image compression In the constituted dataset, 57% of malware use JPG coding to represent images. The remaining ones use BMP and PNG formats in equal proportions (Figure 5.5). These observations appear to correspond to those theorised in Section 4.3.1, with the exception that several formats found in the minority, such as AVI, RAR and WCRT, are missing.

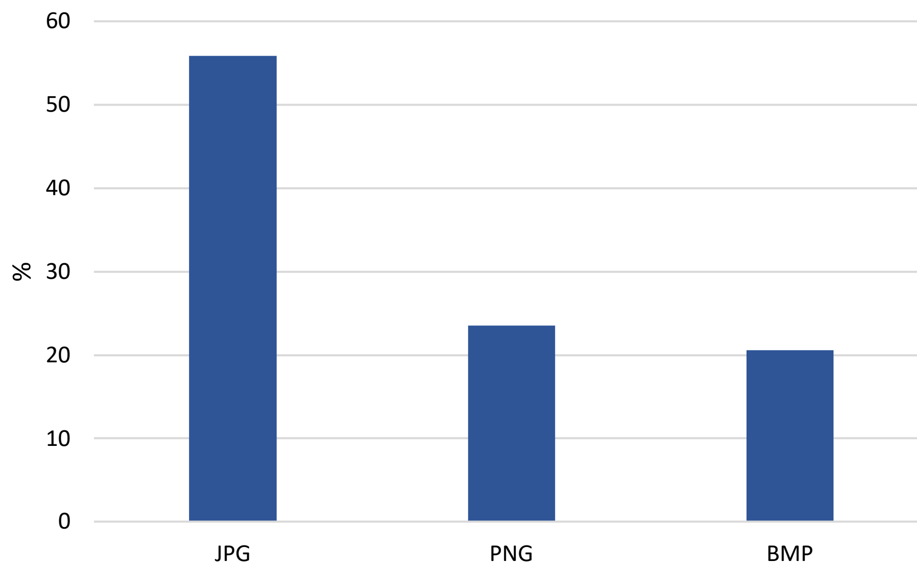


Figure 5.5: Image files format (in the malware of the dataset).

Storage media Regarding screenshot storage, as illustrated in Figure 5.6, in our dataset 66% of malware only use a memory representation of the captured image, while 34% have persistent storage strategies on disks. For this criterion, the results are the opposite of those found in Section 4.3.2, where most of the malware was using disk storage.

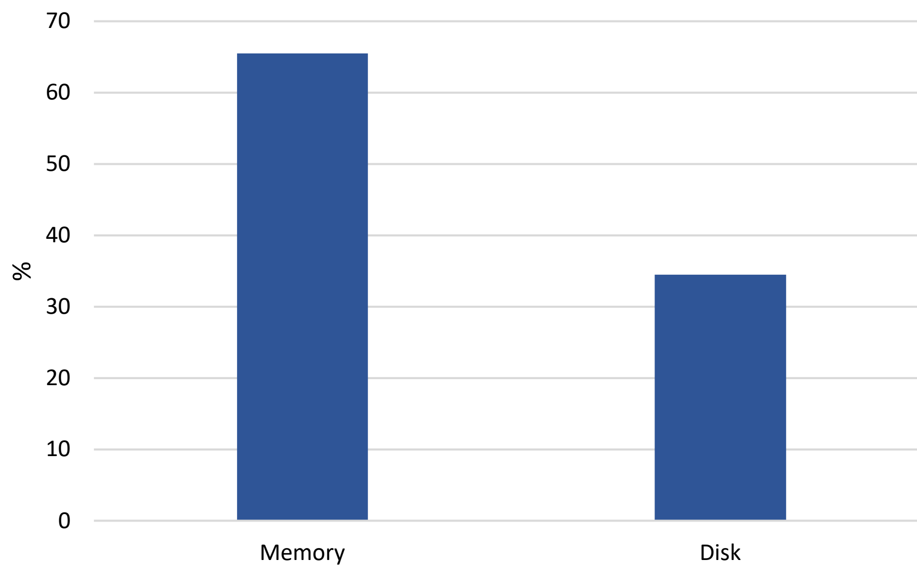


Figure 5.6: Image files storage (in the malware of the dataset).

Screenshot exfiltration

Communication protocol A large proportion of the studied malware does not have an application network layer. Specifically, 94% of the malware examined uses only the transport layer to exfiltrate data in the form of TCP packets. The remaining 6% of our samples use the HTTP protocol.

Encryption Regarding encryption, the majority of the analysed malware samples do not encrypt the sent data. Only a quarter of the malware samples encrypt the information sent, all using different encryption algorithms, namely AsyncRAT (SSL), Carbanak (RC2), CtOSRAT (DES), LimeRAT (AES), SpyNet (RC4) and pyRAT (SSH). Remcos and NetWire allow the use of a custom password for encryption over TCP.

Screenshots sending triggering Regarding the screenshots sending triggering, all malware programs in the dataset send captured images to a remote server im-

mediately after capturing the screen. This is unfortunately not sufficiently representative of the potential behaviours presented in Section 4.4.3. Indeed, all the exfiltration modes that are more advanced than the systematic sending after screen capture are not represented in this sample.

Problem

Even if 118 samples represents a significant size, it is insufficient to represent all the possible behaviours of screenshot-taking malware. Indeed, the analysis results have shown that some behaviours and characteristics that were observed in the security reports analysed in Chapter 4 are missing.

Moreover, Table 5.1 shows the differences between the proportions found in our dataset and the proportions found by analysing the security reports of MITRE.

Therefore, to ensure the completeness of our dataset, we implemented a 'screenlogger generator', as presented in the next section.

Solution: Multi-criteria behaviour generator

Presentation of the generator To complete the malicious dataset, our solution consists of developing a configurable tool that enriches the constructed dataset with all the missing behaviours and characteristics. In this generator, we implemented the functionalities mentioned in the analysed security reports but not found while analysing our dataset. Although not optimal (because some real malware simultaneously carry out other malicious functionalities), this solution allows for a complete toolset to study the screenshot-taking behaviours.

Table 5.1: Screenlogger behaviours in our dataset vs those found in security reports selected from Mitre [14].

		Used Solution	Malw. (Mitre)	Malw. Dataset %
Screen capture	Screenshot capt. lib.	GDI	59	100
		DD API	-	-
	Screenshot capt. trig.	Frequency	35	78
		App. of interest	9	3
		Mouse clicks /User trig.	3	-
		Punctual command	38	19
		Unique capture upon infec.	5	-
	Frequency		2s to 15mn	17ms to 67s
Format and storage	Format	JPG	25	56
		PNG	19	24
		BMP	7	21
		Video	1	-
		Other	6	-
	Storage	Memory	5	66
		Disk	59	34
	Captured area	Full screen	37	87
		Coordinates	2	6
		Difference	-	3
		Other	6	3
Exfiltration	Screenshot send. trig.	Real-time flow	25	73
		Remote cmd	6	-
		Scheduled	10	-
		Other	13	-
	Encryption?	No	-	75
	Files sending?	No	-	-
	Comm. protocol	HTTP	36	6
		HTTPS/FTP/SMTP/RFB	36	-
		Proprietary	1	-
		TCP	18	94

Taking the form of a 'screenlogger generator', our solution poses various questions to the user at the beginning of each execution. To generate a screenlogger with certain characteristics, questions include 'screenshots must be taken every ? clicks', 'should the malware start the screenshots right away or wait for a command?', 'what protocol should be used for exfiltration?' and others (Figure 5.7).

This screenlogger generator takes the form of a builder that generates a payload according to the specified parameters. Then, this payload is run on the victim's machine, whereas the server part waits for connections on the attacker's machine.

Measures were taken to prevent the tool from being used for malicious purposes (a 'malware' flag on network messages and a warning message constantly displayed to the user). The screenlogger generator will also be declared on open and dedicated databases.

As illustrated in Figures 5.7 and 5.8, two types of parameters can be specified when building the payload.

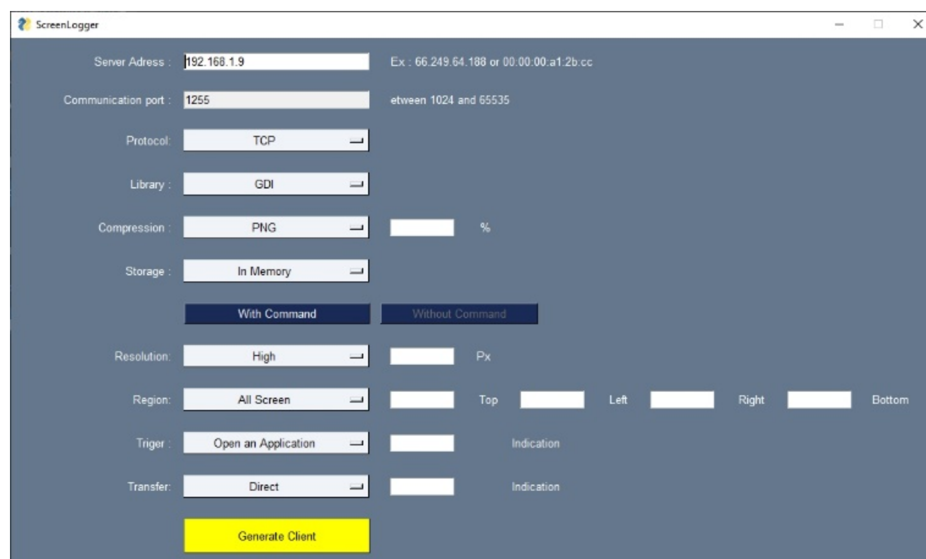


Figure 5.7: Screenlogger generator.

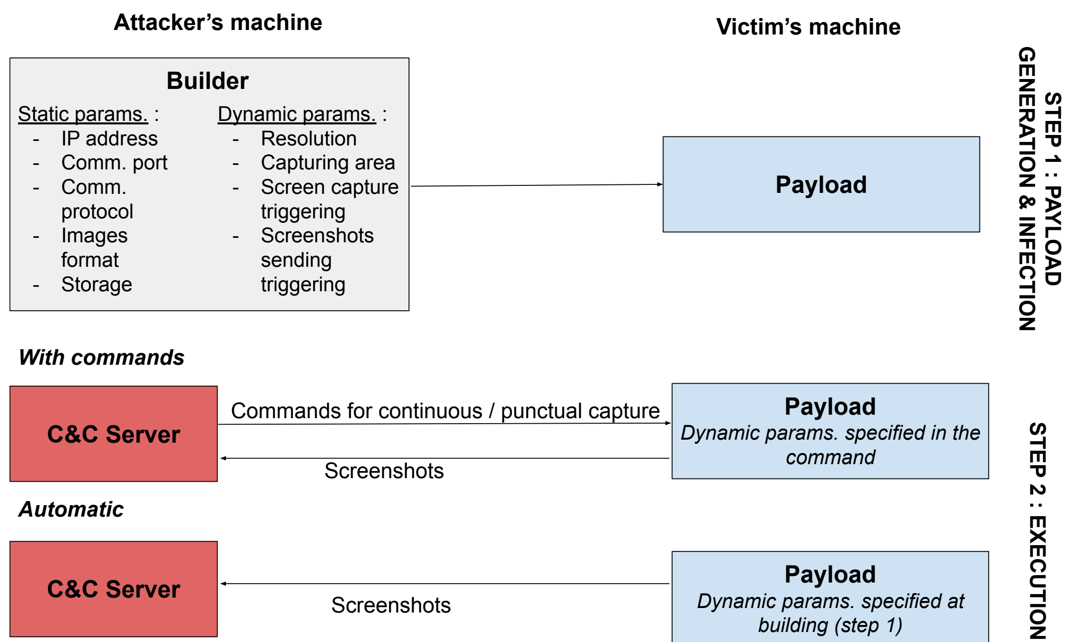


Figure 5.8: Screenlogger generator process.

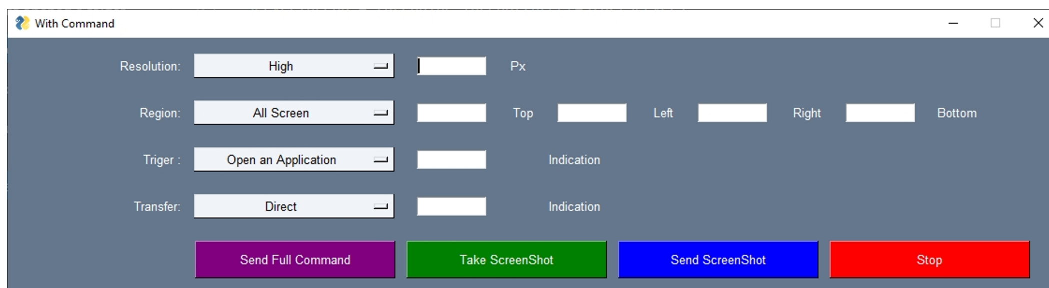
On the one hand, static parameters are predefined before execution and cannot be modified. These are the network parameters that will be used for the connection between the server and the client parts (IP address of the attacker's machine, port, communication protocol - TCP/HTTP/SMTP/FTP), but also the library that will be used to take screenshots (GDI/DD API), the compression format (PNG/JPEG/ZIP/BITMAP/AVI) and the location of storage (disk storage or memory-only).

On the other hand, dynamic parameters can be modified during execution using a command, if this option is chosen. These are:

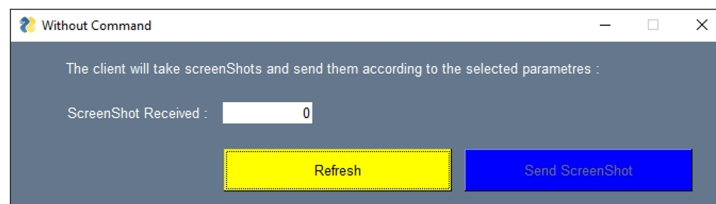
- Resolution of the screenshots
- Screen-capturing area: entire screen / active window / all overlapping windows / mouse area / specified coordinates

- Screenshots capture triggering: frequency / application of interest / each X mouse clicks or keyboard presses
- Screenshots sending triggering: immediate transfer / frequency / waiting for a command / each X screenshots / each time screenshots files reach a size of X bytes

If the option 'without command' is selected, all the parameters will be set at the building time and the payload will start taking screenshots according to these parameters directly upon infection of a new victim (Figure 5.9 a). If the option 'with command' is selected, the parameters cannot be specified at the building time and the screenlogger will wait for a command from the C2 server to start capturing the screen. This command may be used to take isolated or continuous screenshots (Figure 5.9 b).



(a)



(b)

Figure 5.9: Screenlogger generator during execution: (a) When the option 'with command' is chosen (b) When the option 'without command' is chosen.

Several choices, aspects and characteristics of our approach guarantee a certain completeness of the final dataset. Indeed, our screenlogger generator, in addition to almost exhaustively integrating different screenshot-taking behaviours, remains an extremely extensible and scalable tool due to the way it was constructed. For instance, any new screenshot-triggering option might easily be added.

Using the generator to make the dataset complete In addition to adding samples with the missing behaviours to meet the 'behavioural' completeness requirement defined in Section 4.5.1, we also had to match the proportions found in the security reports, as explained in Sections 4.5.2 and 4.5.3. For this, 31 generated samples have been added to our 118 existing samples.

The proportions to match had to account for data that was missing in security reports. For example, if for a given criterion we have 60% unknown, 20% of behaviour A and 20% of behaviour B, the proportions to match are 50% of behaviour A and 50% of behaviour B.

The generated screenloggers that make the dataset complete can be found along with the tools mentioned in Section 5.1.2.3 on the following Github repository: [230]. (For security reasons, the executable is not provided.)

5.2 Benign dataset

In this thesis, we define a legitimate screenshot-taking application as an application that takes screen captures with the knowledge and consent of the legitimate user of the device. The person considered as the legitimate user of the device can be different from the end user. For example, an employee monitoring application will be considered legitimate because the legitimate user (the employer) has given

their consent, even if the actual user of the device (the employee) did not agree to the screenshot-taking.

5.2.1 Screenshot-taking application categories

Benign uses of the screenshot functionality

Ever more legitimate applications use the screenshot functionality. We identified five main categories of such applications:

- Screen sharing: This first category of applications allows a user to have a real-time view of another user's screen. Well-known applications, such Skype [53], Screen Leap [54], Join.me [55], Show My PC [231] and MingleView [232] use the screensharing feature. This functionality is increasingly used due to the rapid development of online learning, telecommuting, remote presentations and conferencing tools.
- Remote control: These tools allow a user not only to see another computer's screen in real time but also to remotely control it through a given protocol. They are generally used by system administrators in companies to access and work on remote computers inside their information systems. Remote control can be used for after-sell or maintenance purposes as well. We could also mention the rise of remote computer fixing with operators asking the users to share their screens to help them resolve the problems they encounter [233]. Some examples of this category are TeamViewer [56], Netviewer [57], GoToMyPC [58], AnyDesk [234], Apple Remote Desktop [235] and Chrome Remote Desktop [236].
- Screencasting: As a digital video recording, screencasting captures actions taking place on a computer that may be transmitted to one or several remote

machines. As opposed to screensharing applications, the recording is not necessarily sent in real-time and may never be sent. Some applications in this category include Camtasia [59], CamStudio [60], Ezvid [61], TinyTake [237] and Telestream ScreenFlow [238].

- Parental control or employee control: These applications allow the legitimate user of a device to monitor the activities of other users of the device. These controls are characterised by the possibility of the current user of the device being unaware that screenshots are taken. The users whose activity is monitored may be children or employees, particularly with the recent development of homeworking [239]. Verity [62], Kidlogger [63], Norton Online Family7 [64], Teramind [201], Time Doctor [240], ActivTrak [241], InterGuard [242], Hubstaff [243] and StaffCop Enterprise [244] are some examples in this application category.
- Screenshot-taking and editing: These applications are used to punctually capture the screen once or a few times for different purposes (e.g., justify a payment, use in a document, etc.). PicPick [65], Snipping Tool [66] and FastStone Capture [67] are examples of this category.

Diverging behaviours

Each of the five categories presented above has specificities regarding the way they take and process screenshots:

- Screen sharing: Screenshots are often taken at a high frequency to allow for a smooth view of the screen activity. The real-time constraint mandates that screenshots must be sent right after they are taken.
- Remote control: The behaviours mentioned for screen sharing also apply to remote control. The main difference lies in the triggering of the screen-

shot operations. Indeed, screenshots start to be taken when the user of the remote-control application chooses to start the session. This behaviour is quite similar to malware programs that start taking screenshots when they receive a command from the malicious server.

- Screencasting: Once again, screenshots are taken at a high frequency. However, they can be grouped and sent at any moment or not sent at all.
- Parental control or employee control: Monitoring a person's activity the whole day by constantly sharing their screen consumes significant resources, especially in a company with thousands of employees. Therefore, these applications tend to take screenshots at an extremely low frequency (e.g., once an hour). The screenshots need not be sent in real-time. They can, for example, be sent together at the end of every working day.
- Screenshot-taking and editing: As they are used punctually, these applications have the most unpredictable behaviour. The screenshots are not taken at a given frequency and may not be sent over the network.

As we can see, legitimate screenshot-taking applications display a wide range of behaviours that sometimes present significant similarities with malicious behaviours. Our objective is to find relevant criteria to distinguish legitimate and malicious screenshot-taking behaviours.

5.2.2 Data collection and analysis

We assembled a dataset of 94 legitimate screenshot-taking applications, among the most commonly used. As explained in the previous section, the main types of applications requiring screenshots (screen sharing, remote control, screencasting, children and employee controls and screenshot-taking and editing) are represented in this dataset.

These applications have been analysed with the same tools used to assess malware: API Monitor for API calls and Wireshark for network activity. This analysis was focused on the criteria presented in Section 4.1.2. Information regarding the selected criteria was not always available. However, the objective was to obtain the maximum amount of information on all applications regarding the maximum number of criteria.

6 | Behavioural Comparison

In this chapter, we first present the results of our analysis of legitimate screenshot taking applications according to the criteria proposed in Chapter 4 (Sections 6.1, 6.2, 6.3).

Then, by comparing these results with those we obtained through the analysis of the security reports referenced on MITRE, we gather insights and formulate hypotheses about the promising criteria for screenlogger detection and differentiation from legitimate screenshot-taking applications (Section 6.4).

6.1 Screen capturing

6.1.1 Used API

Legitimate applications use the two main available libraries to take screenshots: GDI (76%) and the DD API (17%). The DD API is more represented than in the malware case because it is used by several real-time applications (screen sharing - e.g. Skype, remote control - e.g. TightVNC, ShowMyPC, UltraVNC, Ehorus). Indeed, this API is adapted to this type of applications: it allows taking of screenshots in a fast and robust way by sending only the difference between two consecutive screens. However, this is a quite recent API (Windows 8 and above) compared to GDI, and this could explain why it is not used in malware. Malware developers rarely develop all the modules themselves and, in many cases, they use stable and widely tested plugins. Most of those modules are implemented using native technologies like GDI. Another reason could be that this library is much more flexible, allowing to take screenshots at the desired frequency with the desired size.

6.1.2 Screenshot triggering

Screen captures are triggered only while the application is executed, and usually not by a command from a distant server, unlike screenloggers. However, this observation is unfortunately not systematic as some types of legitimate applications such as remote control respond to distant commands to establish the connection and start the screen sharing session.

Another observation is that some applications may exhibit a behaviour very similar to malware in the way screenshots are triggered. For example, Spyrix Free Keylogger, an application for monitoring children's activities, takes screenshots each time the user switches window or opens a new window. Hubstaff, an employee control application, takes screenshots irregularly, and we were not able to identify the triggering event. It may be linked to the "performance factor" of the employee, which is a function of several elements like mouse clicks, keystrokes, opened windows and more.

Finally, all legitimate screenshot-taking applications, except employee and children monitoring, are visible to the user and require user interaction.

6.1.3 Captured area

The capturing areas of the legitimate applications making up our sample are illustrated in Figure 6.1. The majority of applications take the full screen (37%), but some samples target more precise areas of the screen (18% use coordinates, 14% target specific windows). A quarter of the tested applications capture the whole screen but in an optimised way: only the areas of the screen that have been updated are captured. We can note that none of our legitimate applications target overlapping windows.

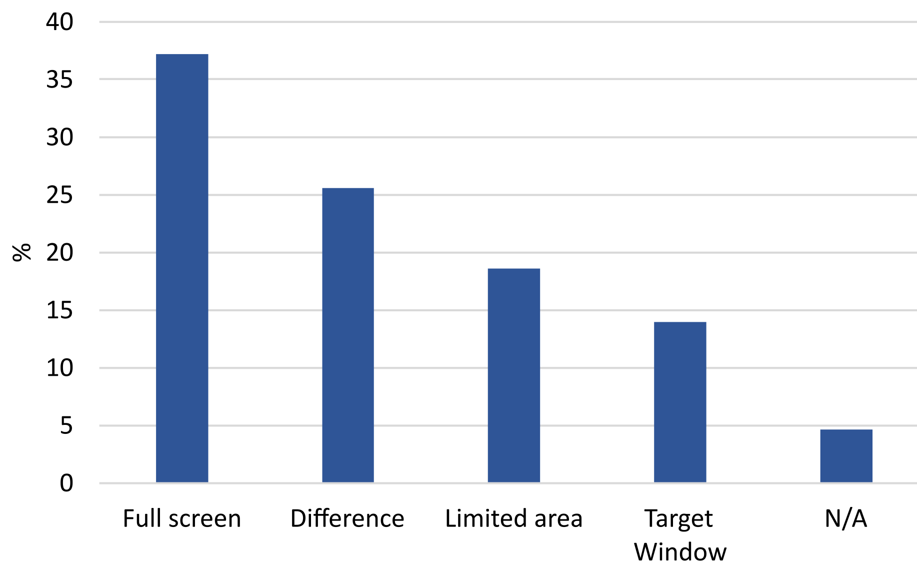


Figure 6.1: Screen capture size (legitimate applications).

6.2 Screenshots storage

6.2.1 Image compression

Determining which format is used for the screenshots captured by legitimate applications is not an easy task. It is often time-consuming and complex because of the encryption and because many samples only use a memory representation of the screenshot. However, it was possible to observe that, unsurprisingly, an important part of legitimate applications for which obtaining the information was possible use standard and portable compression formats (png and jpg) to save captured screenshots (Figure 6.2), with a large majority opting for the png format. Other applications use video formats such as AVI, MPEG4 and VMW to store images: these are the screencasting applications (e.g. VLC, TinyTake, CamStudio).

6.2.2 Storage media

Regarding the storage of screenshots, we can observe different behaviours depending on the type of application (Figure 6.3). 57% of the selected applications only use memory representations of the captured screenshots and do not generate disk persistent storage. These are the screen sharing and remote-control applications This is because these applications use temporary files which are erased after a real-time sending of the data. In contrast, screencasting, screenshot editing and employee control applications store images on the disk.

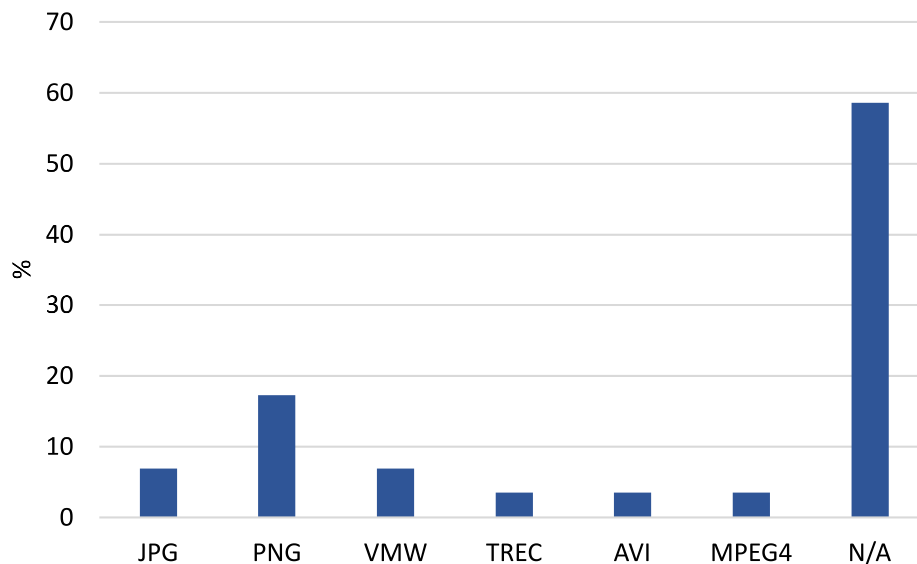


Figure 6.2: Screenshots files format (legitimate applications).

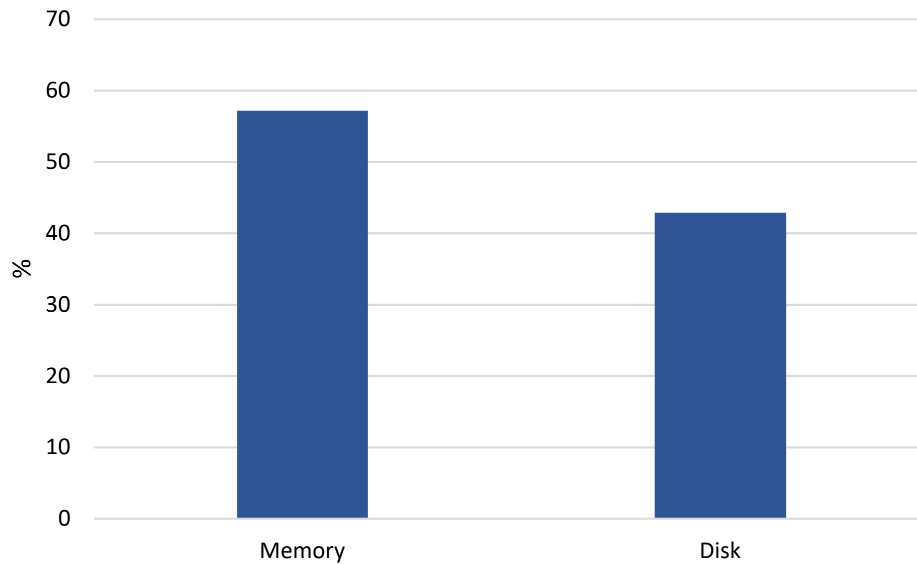


Figure 6.3: Screenshots storage (legitimate applications).

6.3 Screenshots exfiltration

6.3.1 Communication protocol

An important part of the legitimate applications does not send the screenshots over the network because this is not their purpose. Indeed, the screenshots editing, employee control and screen casting applications do not send the screenshots by default. The other applications often use proprietary protocols as shown in Figure 6.4. Few use standard known protocols such as RFB (Remote Frame Buffer protocol) or HTTPS.

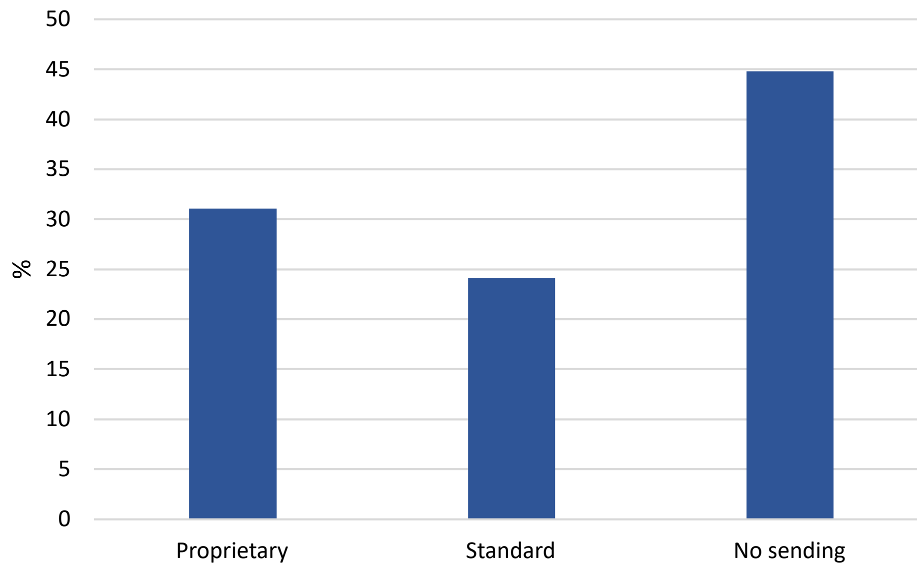


Figure 6.4: Communication protocol (legitimate applications).

6.3.2 Screenshot sending triggering

In real time contexts (screen sharing, remote control), the screenshots are sent immediately after they are taken. As expected, children/employee monitoring applications do not send their screenshots right away. As illustrated in Figure 6.4, the other 45% of legitimate applications never send the screenshots.

6.3.3 Encryption

Applications of our dataset that send the screenshots mainly use either SSL or TLS encryption before transmitting files. A few of them, such as TightVNC and ULTRAVNC do not encrypt files as shown in Figure 6.5.

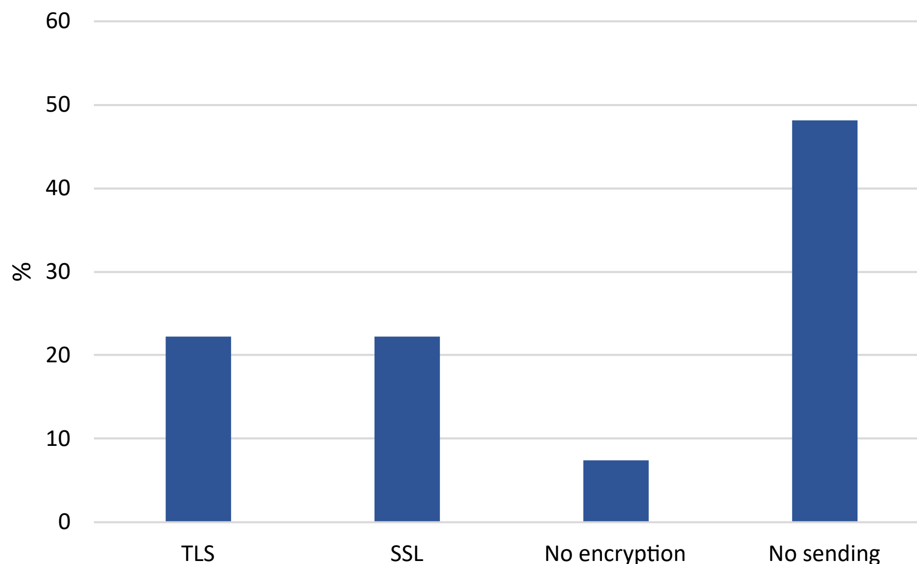


Figure 6.5: Image files encryption (legitimate applications).

6.4 Hypotheses for detection

The criteria on which our study of screenloggers and legitimate screenshot-taking applications (Chapter 4 and Chapter 5) was based allowed us to highlight some significant differences between the two classes of software. These differences are presented and analysed in the rest of this section.

We formulate hypotheses about which criteria seem to be the most suitable for screenlogger detection.

6.4.1 Screen capturing

Used API Regarding screenshot taking, it is possible to discern that the malware programs in our sample only use the GDI library to capture screenshots while 17% of legitimate applications use DD API. This might be an interesting criterion of differentiation of these software classes, but it is limited in the sense that even if it

does not seem to be the case nowadays, malware developers could choose to use the DD API in the future.

Screenshot triggering Our study revealed that most malware (67%) starts taking screenshots in response to commands coming from the C&C server, whereas in the case of legitimate applications, screen capturing is strongly correlated with the use of the application and most of the time cannot be done in response to distant commands or events unrelated to the current application.

However, this reasoning cannot be generalised because of remote control applications like Teamviewer which start taking screenshots in response to a distant connection. Moreover, children/employee control programs may take screenshots in response to user-related events not directly linked with the current applications (mouse clicks, keystrokes, windows opening...), which is a behaviour similar to some screenloggers.

Frequency does not seem to be an interesting criterion because, as with malware, legitimate applications display various behaviours: irregular screenshots (screenshot capture and editing), high frequency (screenshare, remote control) and low frequency (employee/children monitoring).

Finally, contrary to the observations made about our malware samples in Section 5.1.3.1, most legitimate applications do not try to conceal their screenshot-taking activity. This could be an interesting criterion for screenlogger detection.

Captured area The vast majority of malware captures the full screen whereas it is more diverse for legitimate applications: only 37% of them capture the full screen, and the others either capture only part of the screen, a target window or only the difference between two successive screens. This contrast may be explained by the fact that most screenshot-taking malware does not look for specific

information on the infected device, but rather spies on the user's activity in general.

6.4.2 Screenshots storage

Storage media Most legitimate applications use memory representations, for immediate processing on the host machine, without using persistent storage.

This trend is much less evident in the case of malware. Indeed, the theoretical study of malware security reports shows a strong inclination towards the use of the hard drive for storage. This might be because a significant proportion of screenloggers do not send the images right away but rather, wait for specific events (e.g. frequency, buffer size reached, command for screenshot sending).

Image compression Even if the format used by many legitimate applications for screenshot compression remains unknown, it is notable that the same formats are used by both malware programs and legitimate applications to store and send the images, the most common being JPG and PNG. Video formats such as AVI may be used in both cases.

6.4.3 Screenshots exfiltration

Communication protocol As malware programs are very often built by assembling a certain number of basic software blocks, they mainly use standard network protocols. Indeed, only one of the malware programs we analysed (Biscuit) took the trouble to implement and integrate a proprietary protocol, as opposed to 31% of legitimate applications. Thus, even if this cannot be considered the only differentiation criterion, the use of a proprietary protocol by software reduces the probability that it is malware.

Encryption Encryption protocols used by malware and legitimate applications to encrypt images are quite different. Indeed, although a minority of legitimate applications do not encrypt the data they send, most of them do by using the well-known standards SSL and TLS. On the other hand, 75% of malware in our dataset do not even encrypt their data, and in the theoretical study it was observed that, when they do, they use “home-made" and rudimentary encryption techniques combining XOR operations, base 64 encoding and symmetric encoding algorithms such as RC2 and RC4. Few of them use RSA, which is asymmetric, and SSL/TLS, which are a combination of symmetric and asymmetric encryptions. One of the possible explanations may be that asymmetric encryption often requires more computation and that is why it is used less by malware, which is not concerned with the confidentiality and authentication of its network traffic.

Screenshots sending triggering A large proportion of legitimate applications only use screenshots to process them locally without ever transmitting them over the network, unlike malware programs which systematically send captured files over the network to carry out their malicious intents. Indeed, as mentioned in Section 4.4.3, no screenlogger with the local screenshot exploitation feature was found.

Most of the legitimate applications which send the screenshots send them directly after the screen capture as they are real-time applications (remote control, screen sharing). A majority of malware programs exhibit the same behaviour, except that some of them may exhibit different sending patterns, such as sending the image files after a certain number of captures or upon reception of a command. However, the fact that the exfiltration of screenshots is delayed is not sufficient to conclude that the program is malware, because children/employee control may

behave similarly, by sending the screenshots at the end of the control session for example.

6.5 Discussion

Based on our experiments and execution analysis of screen-logging malware and legitimate applications, Table 6.1 summarises the difference levels for each criterion.

In this table, we mainly distinguish three classes of criteria according to the degree of differentiation between malware and legitimate applications.

Indeed, criteria like the compression format and screen capture frequency do not highlight enough differences between legitimate applications and malware to be used effectively in a detection methodology.

In the second class we find criteria of average importance in terms of differentiation between the two types of software. These are the screenshot function library, screenshot triggering, files sending triggering, storage and the captured area. These criteria allow quite significant differentiation but with some specific cases and exceptions which make generalisation difficult. This is for instance the case of screenshots sending triggering, for which legitimate applications and malware seem to use different strategies, except that legitimate applications like parental control behave, on this criterion, in a similar way to malware.

Table 6.1: Screenloggers behaviours in the malicious dataset vs those found in the legitimate dataset.

		Used Solution	Legitimate	Malicious	Dif. deg.
Screen capture	Scr. capt. lib.	GDI	76	59	++
		DD API	17	-	
	Scr. capt. trig.	Frequency	63	35	++
		App. of interest	3	9	
		Mouse/User trig.	23	3	
		Punctual command	-	38	
		Capt. upon infec.	-	5	
	Frequency		9ms to 1h	2s to 15mn	+
Format and storage	Format	JPG	7	25	+
		PNG	17	19	
		BMP	-	7	
		Video	17	1	
		Other	-	6	
	Storage	Memory	57	5	++
		Disk	43	59	
	Captured area	Full screen	37	37	++
		Coordinates	19	2	
		Difference	26	-	
		Other	14	6	
Exfiltration	Scr. send. trig.	Real time flow	43	25	++
		Remote cmd	-	6	
		Scheduled	3	10	
		Other	-	13	
	Encryption?	No	7	-	+++
	Files sending?	No	50	-	+++
	Comm. prot.	Http	-	36	+++
		Https/Ftp/Smtp	24	36	
		Proprietary	24	1	
		TCP	-	18	

The third and final class includes criteria with significant degrees of differentiation: sending of screenshot files, encryption and the communication protocol. Indeed, software that uses a proprietary protocol with TLS/SSL encryption for network communication or which does not send the screenshots over the network is likely to be a legitimate application, whereas one that does not encrypt data or just uses base64 encoding and that uses a standard protocol such as TCP, FTP or SMTP might be malware. However, a detection methodology limited only to these criteria cannot be effective because of the potential for false positives.

Thus, an effective malware screenshot detection approach should not only be based on the criteria of this third class but also integrate those of the second class, possibly with different weights.

7 | Behavioural Screenlogger Detection

Using the malicious and benign datasets constructed in Chapter 5, we were able to identify the features found in the literature which are the most performant for screenlogger detection. Moreover, we trained and tested a malware detection model with new features adapted to the specifics of screenlogger behaviour.

In this Chapter, we start by outlining our experimental setup for screenlogger detection (Section 7.1) and introducing the performance metrics we use (Section 7.2). Then, we present a detection model based on state of the art features (Section 7.3) and a novel detection model including features specific to the screenlogging behaviour (Section 7.4). Finally, the performances of the two models are compared and discussed (Section 7.5).

7.1 Experimental Setup

As presented in Section 5.1.2.3, our malicious samples were run in two Windows 10 virtual machines to allow the client and server parts to communicate and trigger the screenshot functionality. Legitimate applications were also run in two machines when it was required for screenshot-triggering.

During their execution, the behaviour of malicious and benign samples were monitored using API Monitor and Wireshark.

To implement and test our detection models, we used the Weka framework, which is a collection of ML algorithms for solving real-world data mining problems [245].

More precisely, we used it to process the run-time analysis reports, select the best detection features, select the classification algorithms, train and test the models, and visualise the detection results.

We used the python-weka-wrapper3 python library to access Weka features. This library was chosen to easily control the classification process flow (e.g. save feature selection, save detection results, control detection results).

7.2 Performance measurements

Malware detection is a binary classification problem with two classes: malware and legitimate application.

The measures used to assess the performances of our detection models are the following:

- True Positives (TP): Number of malware programs classified as malicious.
- False Positives (FP): Number of legitimate applications classified as malicious.
- True Negatives (TN): Number of legitimate applications classified as legitimate.
- False Negatives (FN): Number of malware programs classified as legitimate.
- Accuracy: Given by the formula $\frac{TP+TN}{TP+TN+FP+FN}$. Accuracy does not discriminate between false positives and false negatives.
- Precision: Given by the formula $\frac{TP}{TP+FP}$. Precision is inversely proportional to the number of false positives.

- Recall: Given by the formula $\frac{TP}{TP+FN}$. Recall is inversely proportional to the number of false negatives.
- F-score: Given by the formula $\frac{2*Precision*Recall}{Precision+Recall}$. Contrary to accuracy, F-score decreases more rapidly if false positives or false negatives are high (i.e. precision or recall are low).

In the case of malware detection, it is crucial that all malware programs be detected, to avoid them causing important damage. On the other hand, classifying a legitimate application as malware, even if it can be inconvenient for the user, might not be as critical. As a result, we give a particular importance to the false negatives and recall metrics.

7.3 Basic detection approach

To prove the effectiveness of our novel detection model, it was first necessary to construct a model based on features from the malware detection literature. We started by extracting these features (Section 7.3.1). Then a ML model (Section 7.3.2) was trained (Section 7.3.3) to select the most effective features (Section 7.3.4). The model was evaluated using our performance metrics (Section 7.3.5).

7.3.1 Feature extraction

As discussed in Section 2.2.4, the screenlogging process involves taking screenshots by using API calls and sending them over the network. Moreover, as seen in Section 2.2.3, recent detection works combining different types of dynamic features (such as API calls occurences and network features) achieve better results. This is why our state of the art detection model uses these two kinds of features.

API call features and network features were tested both independently and conjointly.

When running the samples from our malicious and benign datasets in a controlled environment, we collected reports on two aspects of their behaviours: API calls (API Monitor reports) and network activity (Wireshark reports).

API calls

This category of features was extracted from the reports produced by API Monitor.

The first feature we used consisted in counting the number of occurrences of each API call. For each malicious and benign API call report, the numbers of occurrences of the API calls it contains was extracted in a .csv file.

In the literature, we found that malware programs try to dissimulate their malicious functionality by introducing benign API calls to their API call sequences. A popular way of performing malware detection using API calls is to use the number of occurrences of API call sequences rather than API calls taken alone. For this, the concept of N-grams is used. N-grams are sequences of N API calls made successively by the studied program.

As a result, we also extracted features based on the number of occurrences of 2-gram and 3-gram API calls sequences. The values of N were intentionally kept low for two reasons: (1) the number of features increases exponentially with N, and (2) the detection performance decreases as N increases.

Network traffic

Using the .pcap files produced by Wireshark and the Argus tool to isolate network flows [246], we extracted 47 network features found in the literature. These features belong to four categories:

Behaviour-based features ([92, 247])

These features represent specific flow behaviours. They include:

- Duration of the flow
- Source IP address
- Destination IP address
- Source port
- Destination port

Byte-based features ([93])

These features use byte counts. They include:

- Total number of bytes from source to destination
- Total number of bytes from destination to source
- Average number of bytes from source to destination
- Average number of bytes from destination to source
- Variance of the number of bytes from source to destination
- Variance of the number of bytes from destination to source
- Number of bytes per second
- Total number of bytes in the flow over the number of packets in the flow

Packet-based features ([92, 93, 99])

These features are based on packet statistics. They include:

- Total number of bytes in the initial window from source to destination

- Total number of bytes in the initial window from destination to source
- Ratio between the number of incoming packets and the number of outgoing packets
- Number of small packets (length < 400 bytes) exchanged
- Percentage of small packets exchanged
- Number of packets per second
- Minimum flow packet length
- Maximum flow packet length
- Mean flow packet length
- Median flow packet length
- Standard deviation of flow packet length
- Number of FIN packets
- Total number of packets from source to destination
- Minimum size of packets from source to destination
- Maximum size of packets from source to destination
- Mean size of packets from source to destination
- Standard deviation of the size of packets from source to destination
- Total number of packets from destination to source
- Minimum size of packets from destination to source
- Maximum size of packets from destination to source
- Mean size of packets from destination to source

- Standard deviation of the size of packets from destination to source

Time-based features ([92, 93, 99])

These features depend on time. They include:

- Minimum time a flow was idle before becoming active
- Maximum time a flow was idle before becoming active
- Mean time a flow was idle before becoming active
- Standard deviation of the time a flow was idle before becoming active
- Minimum time a flow was active before becoming idle
- Maximum time a flow was active before becoming idle
- Mean time a flow was active before becoming idle
- Standard deviation of the time a flow was active before becoming idle
- Total PUSH packets
- Total SYN packets
- Total ACK packets
- Total Urgent packets

7.3.2 Detection algorithm

Our detection model uses the RF algorithm [248]. This algorithm trains several DTs and uses a majority vote to classify observations. Each DT is trained on a random subset of the training dataset using a random subset of features.

The main shortcoming of DTs is that they are highly dependant on the order in which features are used to split the dataset. RF addresses this issue by using multiple trees using different features.

We tested several parameters to improve the performances of the model:

- Number of trees in the forest (by default 100).
- Number of randomly selected features for each tree.
- Maximum depth of the trees (by default unlimited).
- Minimum number of instance per leaf (by default 1 but can be raised to prevent overfitting).

It emerged from our experiments that the default parameters described above provided the best restults. Bootstrapping was used.

Other algorithms than RF were tested: KNN (K=7; LinearNNSearch) and SVM (polynomial kernel, normalised training data). As shown in Section 7.3.5, RF yielded the best results.

Using an Artificial Neural Network was also considered, but we did not have a sufficient number of samples in our dataset. Indeed, neural network require a large amount of data for training.

7.3.3 Model training and testing

To train and test our model, we used the k-fold cross-validation method (with $k = 10$). This value of k allowed to reach the best tradeoff between bias and variance. As seen in Section 7.3.5, other values were tested but led to degraded performance.

This method consists in dividing our dataset into k blocks of the same size. The blocks all have the same proportions of malware and legitimate applications. For each block, we train the model on the $k-1$ other blocks and test it on the current block. The final detection results are obtained by adding the results of each block.

Using cross-validation, we trained and tested our model using first API call features only, then network features only, and, finally, using both categories of features.

7.3.4 Feature selection

Due to the high number of features used, to avoid overfitting, it was necessary to select the most useful ones. A features is useful if it is informative enough for our classification task, that is, if it enables to effectively distinguish between malicious and benign behaviours.

For this task we used the Recursive Feature Elimination (RFE) method [249]. Given a number of features to select, this method iteratively trains our RF model using cross-validation and removes the least important features at each iteration. The importance of a feature is given by the average of its Gini impurity score for each DT in which it is used.

The Gini impurity of a feature that splits the samples at a node of a DT reflects how 'pure' are the subsets produced by the split. In our case, a subset is purer if it contains mostly screenloggers or mostly legitimate screenshot-taking applications. For instance, a subset containing 75% malware and 25% legitimate applications is purer than a subset that contains 50% malware and 50% legitimate applications.

The impurity of a subset is given by the formula:

$p(malware) * (1 - p(malware)) + p(legitimate) * (1 - p(legitimate))$.

That is: $2 * p(malware) * p(legitimate)$.

The Gini impurity of a feature is the weighted average of the impurity scores of the subset it produces. The weights are computed using the number of samples contained in each subset.

When the features are numerical values (which is our case), instead of computing the impurity of the subsets produced by each single value, intervals are used. More precisely, the Gini impurity of the feature is obtained through the following steps:

- Step 1: The values of the feature are sorted.
- Step 2: The averages of each adjacent values are computed.
- Step 3: For each average value from Step 2, the Gini impurity of the feature if the samples were split using this value is computed.
- Step 4: The Gini impurity of the feature is the minimum among the Gini impurities from Step 3.

7.3.5 Evaluation

Table 7.1 contains the results we obtained for the first detection approach using features found in the literature with the RF algorithm and k=10.

Table 7.1: Detection results for the basic approach using features from the literature with the RF algorithm (k=10).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	94.301%	0.038	0.080	0.936	0.962	0.949
1-gram	92.228%	0.038	0.126	0.903	0.962	0.932
2-gram	88.601%	0.104	0.126	0.896	0.896	0.896
3-gram	83.938%	0.123	0.207	0.838	0.877	0.857
(1+2)-gram +network	94.301%	0.038	0.08	0.936	0.962	0.949
1-gram + network	94.301%	0.028	0.092	0.928	0.972	0.949

Table 7.2 contains the results we obtained for the first detection approach using features found in the literature with the KNN algorithm and k=10.

Table 7.2: Detection results for the basic approach using features from the literature with the KNN algorithm (k=10).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	91.192%	0.195	0.000	0.862	1.000	0.926
1-gram	84.974%	0.276	0.047	0.808	0.953	0.874
2-gram	82.383%	0.333	0.047	0.777	0.953	0.856
3-gram	88.601%	0.184	0.057	0.862	0.943	0.901
(1+2)-gram +network	88.601%	0.253	0.000	0.828	1.000	0.906
1-gram + network	88.601%	0.253	0.000	0.828	1.000	0.906

Table 7.3 contains the results we obtained for the first detection approach using features found in the literature with the SVM algorithm and k=10.

Table 7.3: Detection results for the basic approach using features from the literature with the SVM algorithm (k=10).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	91.710%	0.172	0.009	0.875	0.991	0.929
1-gram	81.347%	0.069	0.283	0.927	0.717	0.809
2-gram	80.289%	0.011	0.340	0.986	0.660	0.791
3-gram	79.793%	0.000	0.368	1.000	0.632	0.775
(1+2)-gram +network	93.264%	0.138	0.009	0.897	0.991	0.942
1-gram + network	92.228%	0.161	0.009	0.882	0.991	0.933

For all categories of feature except 3-gram API calls, RF outperforms KNN and SVM.

We can observe that network features seem to give better results overall than API call features. Regarding API calls, using sequences of two and three calls significantly decreases the performances of the model, with more than 10% of malware classified as legitimate (vs 3.8% when individual API calls are used).

Combining network features and API call features does not improve the results compared to using network features alone.

Tables 7.4, 7.5, 7.6, 7.7 and 7.8 show the detection results for respectively k=3, k=5, k=7, k=12 and k=15 and the RF algorithm. We can see that k=10 yields the best accuracy (see Table 7.1). With k=10, the variance is 2.111% and the training time is 0.03sec.

Table 7.4: Detection results for the basic approach using features from the literature with the RF algorithm (k=3).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	94.301%	0.069	0.047	0.944	0.953	0.948
1-gram	91.192%	0.138	0.047	0.894	0.953	0.922
2-gram	89.119%	0.149	0.075	0.883	0.925	0.903
3-gram	86.010%	0.149	0.132	0.876	0.868	0.872
(1+2)-gram +network	93.264%	0.092	0.047	0.927	0.953	0.940
1-gram + network	93.783%	0.103	0.028	0.920	0.972	0.945

Table 7.5: Detection results for the basic approach using features from the literature with the RF algorithm (k=5).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	90.674%	0.115	0.075	0.907	0.925	0.916
1-gram	92.746%	0.092	0.057	0.926	0.943	0.935
2-gram	90.155%	0.149	0.057	0.885	0.943	0.913
3-gram	89.119%	0.138	0.085	0.890	0.915	0.902
(1+2)-gram +network	94.819%	0.092	0.019	0.929	0.981	0.954
1-gram + network	95.337%	0.092	0.009	0.929	0.991	0.959

Table 7.6: Detection results for the basic approach using features from the literature with the RF algorithm (k=7).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	92.228%	0.092	0.066	0.925	0.934	0.930
1-gram	93.782%	0.080	0.047	0.935	0.953	0.944
2-gram	89.637%	0.138	0.075	0.891	0.925	0.907
3-gram	89.119%	0.161	0.066	0.876	0.934	0.904
(1+2)-gram +network	94.301%	0.069	0.047	0.944	0.953	0.948
1-gram + network	96.891%	0.057	0.009	0.955	0.991	0.972

Table 7.7: Detection results for the basic approach using features from the literature with the RF algorithm (k=12).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	93.782%	0.0092	0.038	0.927	0.962	0.944
1-gram	93.782%	0.092	0.038	0.927	0.962	0.944
2-gram	89.637%	0.149	0.066	0.884	0.934	0.908
3-gram	88.083%	0.161	0.085	0.874	0.915	0.894
(1+2)-gram +network	93.264%	0.092	0.047	0.927	0.953	0.940
1-gram + network	94.301%	0.080	0.038	0.936	0.962	0.949

Table 7.8: Detection results for the basic approach using features from the literature with the RF algorithm (k=15).

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
network	93.782%	0.080	0.047	0.935	0.953	0.944
1-gram	92.230%	0.103	0.057	0.917	0.943	0.930
2-gram	90.155%	0.149	0.057	0.885	0.943	0.913
3-gram	89.119%	0.149	0.075	0.883	0.925	0.903
(1+2)-gram +network	93.264%	0.103	0.038	0.919	0.962	0.940
1-gram + network	94.819%	0.080	0.028	0.936	0.972	0.954

Finally, using RFE, we identified the most relevant API calls for screenlogger detection:

- strcpy_s (Visual C++ Run Time Library)
- ntreleasemutant (NT Native API)

- _isnan (Visual C++ Run Time Library)
- getobjectw (Graphics and Gaming)
- rtltimetotimefields (NT Native API)

We also identified the most relevant state of the art network features:

- Bytes per packet
- Total number of bytes in the initial window from source to destination
- Total number of bytes in the initial window from destination to source
- Total number of bytes from source to destination
- Average number of bytes in a subflow from source to destination

7.4 Optimised detection approach

As discussed in Section 2.2.4, when the amount of data at our disposal is low, such as in the case of screenloggers, a classical machine learning model using thousands of features is very likely to fall into overfitting.

The novel detection approach we propose is based on new features specific to the screenlogger behaviour (Section 7.4.1). Our model was tested using the same metrics as the model that uses only the features found in the literature (Section 7.4.2).

7.4.1 New features

Thanks to the comparison made in Chapter 6 between malicious and legitimate screenshot-taking behaviours, we were able to identify promising features for

screenlogger detection. These features target specific behaviours that can allow screenloggers and legitimate screenshot-taking behaviours to be distinguished.

The first novelty of this approach is that, instead of using hundreds of features and trusting a ML model to select the most discriminating ones, we use features that reflect a specific behaviour. The advantage is that malware authors will not be able to misguide the detection system without changing their core functionality. Indeed, as seen in Section 2.2, existing detection models are prone to overfitting and can easily be misguided by malware authors by acting on features unrelated to the malicious functionalities of their programs.

The second novelty lies in the way our features are collected. Indeed, in the malware detection field, it is commonplace to automatically run millions of malware samples in a controlled environment to collect features without interacting with the samples. Such an approach would unfortunately not work for screenloggers, as their malicious functionality needs to be triggered at run time through interaction with the malware program. To collect our features, we paid a particular care to ensure that each malicious or legitimate program worked as intended.

A prerequisite to the following features is that the studied application has been identified as having a screenshot-taking activity (using the API call sequences from Section 5.1.2.1).

Interaction with the user

As observed in Section 6.4.1, contrary to screenloggers, a majority of legitimate screenshot-taking applications require an interaction with the user to start taking screenshots.

To extract this feature, we had to identify the API calls which result from user interaction.

We found that, on Windows, some API calls involved in user interaction can be called on other applications' windows. As such, they could easily be called by a malware program pretending to interact with the user, whereas in fact, it does not even have a window.

Other API calls, mainly those involved in drawing on the window can only be called by the application that created the window. If they are called by another application, their return value is $false$. Therefore, we monitor this second category of functions and, even if they are called, we verify their return value.

Visibility of the screenshot-taking process

We saw in Section 6.4.1 that, unless they infiltrate themselves in legitimate processes, all the malicious samples of our dataset take screenshots through background processes hidden to the user. Legitimate screenshot-taking applications, apart from children/employee monitoring and some applications that create a background process for the screenshot-taking (e.g. TeamViewer), use foreground processes.

Thus, the fact that the screenshots are taken by a background process increases the probability of malicious activity.

Image sending

As pointed out in Section 6.4.3, a major part of legitimate screenshot-taking applications do not send screenshots over the network, contrary to our malware samples (no malware with the local OCR exploitation feature was found in the security reports and our dataset). However, due to the limited monitoring time (... minutes), we cannot tell for sure that the screenshots taken by a given application will never be sent. Indeed, as observed in Section 4.4.3, some malware can for instance

schedule the sending of screenshots. In such a case, even if image packets are not sent during the monitoring time, it can be that these packets will be sent later.

Therefore, our 'Image sending' feature only reflects whether or not screenshots are sent during the monitoring time, and cannot be used to affirm that an application does not send the screenshots it takes.

Moreover, determining whether a network packet contains an image is only possible when the packet is not encrypted.

Remote command triggering

As discussed in Section 6.4.1, an important characteristic shared by almost all screenloggers is that their screenshot-taking activity is triggered by command received from their C&C server.

In Section 4.2.2, we presented two kinds of screenshot-triggering commands: commands for continuous capture of the screen and punctual commands for a single screenshot. In the first case, only one command is received at the beginning of the screenshot session, whereas in the second case, a command is received before every screenshot event. To cover both cases, we chose to consider that the screenshot-taking activity is triggered by a command even if only one screenshot is preceded by the reception of a network packet.

We had to determine an adequate duration between the reception of the command and the screenshot. Indeed, we only consider that the screenshot was triggered by the network packet if this packet is received within a given time-window T before the screenshot API call sequence.

Concretely, for each screenshot taken, we control if:

$t(screenshot) - t(lastNetworkMessage) < T.$

Note that, to measure this feature accurately, it was necessary that the API calls and network reports be generated at the exact same time.

By analysing our samples, we found that the maximum duration between the command and the screenshot is 46 772 ms, the minimum duration is 0.0059 ms, the average duration is 83.044 ms and the median duration is 33.115 ms. We conducted experiments with these different values for T.

Even if this was not found in our dataset, we account for the case where the process receiving the command is different from the process taking the screenshots.

To the best of our knowledge, our detection model, through this feature, is the first to make a correlation between two kinds of events (reception of a command and screenshot API call sequences) for malware detection.

Asymmetric traffic

As mentioned in Section 7.3.1.2, one of the packet-based network feature we found in the literature is the ratio between the number of incoming packets and the number of outgoing packets. This feature fails to capture the asymmetric traffic displayed by most screenloggers as opposed to legitimate screenshot-taking applications (e.g. video call with screen sharing). Indeed, in the case of screenloggers, the asymmetry lies in the quantity of data exchanged, and not necessarily in the number of packets. It may be that the number of incoming and outgoing packets are equal, for example in the case of punctual screenshot commands. In such a case, however, the quantity of data received from the C&C server is significantly lower than the quantity of data sent by the victim machine.

Therefore, instead of measuring the ratio between the number of incoming and outgoing packets, we use the ratio between the numbers of bytes exchanged in both directions.

Captured area

In Section 6.4.1, we saw that almost all malware capture the full screen as opposed to legitimate applications which may target more specific areas of the screen depending on their purpose. As a result, we implemented a 'captured area' feature which takes three values: full screen, coordinates and target window.

We had to identify, in our screenshot API call sequences, the elements that show what area of the screen is captured. However, as discussed in Section 5.1.2.1 for the screenshot sequences, there is not only one way to capture a given area of the screen, but several. For instance, to capture a zone with given coordinates, one might get a cropped DC from the beginning using the GetDC function with the desired coordinates as parameters, or take the whole DC and do the cropping afterwards when copying the content of the screen in the destination bitmap using BitBlt's arguments.

Therefore, for each of the three values of the 'captured area' feature, we listed the possible API call sequences which might be used.

Note that we consider that an application capturing more than the three quarters of the screen's area captures the full screen. This is to avoid malware programs pretending that they capture a precise area when, in fact, only few pixels are removed from the whole screen.

Screenshot frequency

The last screenlogger-specific feature we created is the frequency of screenshots. We consider that the studied application takes screenshots at a given frequency if we find the same time interval between ten screenshots. Indeed, some malware programs offer to take punctual screenshot as well as continuous screen capture.

Therefore, it is possible that not all the screenshots be taken at the same time interval.

Each time a screenshot API call sequence is found, we record its time stamp. Then, we subtract the timestamps of consecutive sequences and compare the intervals obtained. If more than ten intervals are found to be equal, the feature takes the value of this interval. Else, it takes the value 'no frequency'. Note that screenshots taken using different sequences are accounted for in this frequency calculation.

As seen in Section 5.1.3.1, some malware programs may try to evade detection by dynamically changing the screenshot frequency using random numbers. To cover this case, we consider that the intervals are equal if they are within 15s of each other.

7.4.2 Evaluation

Table 7.9 contains the results we obtained for the second detection approach using the screenlogger-specific features we implemented.

Table 7.9: Detection results for the optimised approach using our specific features.

Features	Accuracy	FN	FP	Precision	Recall	F-Measure
Specific features	97.409%	0.009	0.046	0.963	0.991	0.977

We can see that the detection performance is improved on all metrics: with only 7 features, our model outperforms the first model based on hundreds of standard features. That is because our features capture specific malicious behaviours.

Moreover, a malware author would not be able to act on these features to mislead the classifier without changing the malicious functionality. Indeed, to mislead

traditional classifiers based on numerous features, malware authors leverage over-fitting by acting on features unrelated the core functionality of their programs. When all the features target a specific behaviour, as in our case, this cannot be done.

7.5 Discussion

We built a first RF detection model using only API calls and network features from the literature. This model was trained and tested using our malicious and benign datasets. Using RFE with Gini importance, we identified the most informative existing features for screenlogger detection.

Then, we built a second model including novel features adapted to the screenlogging behaviour. These features were collected using novel techniques. Particularly, we can cite:

- Using API call sequences to identify specific behaviours. Contrary to existing works which only look at API called in a direct succession using the notion of n-grams, we wrote scripts which keep track of the API calls return values and arguments to characterise some behaviours even if the calls are not made directly one after the other. Numerous different sequences involved in the screenshot-taking process were identified by analysing malware and legitimate applications. These sequences were also divided into three categories depending on the captured area.
- Making a correlations between API calls made by an application and its network activity. During their execution, the API calls and network activity of our samples were simultaneously monitored. This allowed us to extract fea-

tures such as the reception of a network packet before starting the screenshot activity or the sending of taken screenshots over the network.

When adding these novel features to the detection model, the detection accuracy increased by at least 3.108%. Indeed, it is well known that, when there is few data, a detection model based on less features is less likely to fall into overfitting. Moreover, a detection model based on features which have a logical meaning and reflect specific behaviours, is less prone to evasion techniques often used by malware authors.

More generally, our results show that, for some categories of malware, a tailored detection approach might be more effective and difficult to mislead than a generalist approach relying on a great number of seemingly meaningless features fed to a ML model.

8 | Leveraging Retinal Persistence for a Usable Countermeasure to Malicious Screenshot Exploitation

Although the detection approach presented in Chapter 7 gives promising results, some malware will inevitably remain undetected. In such a case, we believe it is necessary to prevent the malicious exploitation of the screenshots, or at least to make it more difficult for the attacker. As presented in our threat model (Section 3.4), the stolen screenshots can either be exploited automatically using OCR algorithms, or manually by the human attacker.

As discussed in Section 2.3, today, there is no scalable solution aimed at protecting the screen's content from malicious screen capture in the general case (i.e. consumer protection).

It is important to stress that, in our view, any white-listing solution relying on the user to allow or deny screenshot-taking by a given application is doomed to fail. Indeed, there are multiple ways in which malware authors can bypass such a solution, for example by misguiding the user. To illustrate this point, three concrete examples of screenshot-taking malware programs found in the wild have been selected:

- The RTM malware mimics a registry check by using the regedit icon and a design similar to Window's progress bars. Then, a fake error message is shown to the user. Clicking on one of the two options runs a process with administrator privileges [197].

- The Prikormka malware records Skype calls using the Skype Desktop API. The use of this API by a third-party application causes Skype to display a warning asking the user to allow or deny access. The malware then creates a thread that attempts to find the window and click the “Allow access" button programmatically, without human interaction [202].
- The Janicab malware uses the right-to-left override character in its file’s name. This way, a .fdp.app file becomes a .ppa.pdf one and the users think that they are opening a PDF instead of an executable ([250]).

All of these examples demonstrate that malware authors can deploy countless strategies to mystify users or completely bypass them by simulating events.

For this reason, in this Chapter, we propose an approach to mitigate malicious screenshot exploitation while trying to address the following challenges:

- Not requiring the user’s intervention for configuration or parameterisation, contrary to existing solutions: Many users have little-to-no culture when it comes to computer security and they are often reluctant to change their habits and practices when using devices. Thus, any effective method targeting large-scale use must, as much as possible, avoid changes to user practices.
- Being user-friendly: The solution must be transparent to users in the sense that it must minimise the impact on the user’s visual comfort and not overly alter the functioning of legitimate screenshot-taking applications.
- Being real-time: This is an implication of the fact that the solution must be user-friendly. Indeed, if the solution requires heavy treatments on each screenshot, it will not allow screenshot-taking applications such as screen-

sharing applications to take screenshots at a high-enough frequency to function normally.

After discussing the underlying foundations of our approach (Section 8.1), we present how we implement it by leveraging the visual persistence property of the human eye (Section 8.2). Then, hypotheses are formulated regarding the performance of the system for different variations of its parameters (Section 8.3).

8.1 Proposed model: On-the-fly screenshot alteration

As discussed earlier, we have chosen not to involve the user in any way, such as to distinguish between malicious and legitimate uses of the screenshot functionality. This implies that, to prevent malicious screenshot exploitation, we have to affect the functioning of legitimate screenshot-taking applications as well as malware.

Concretely, the proposed approach does not aim to change the general way in which information is displayed on the screen for the user's day-to-day activity, but only alters the screenshot function.

The idea, instead of returning a screenshot containing the whole information displayed on the screen in response to a screenshot API call, is to return an altered image (Figure 8.2). On Windows devices, screenshots are usually taken using the GDI API. The returned bitmap is generated from a DC obtained with functions such as GetDC.

Using hooking techniques such as the one offered by the Microsoft Detours library [251], we can intercept the API call used to take the screenshot (e.g. BitBlt) and modify its return value. More precisely, the functions' definitions are modi-

fied by inserting jump instructions. The jump instructions modify the program's execution flow by redirecting it to our code (figure 8.1).

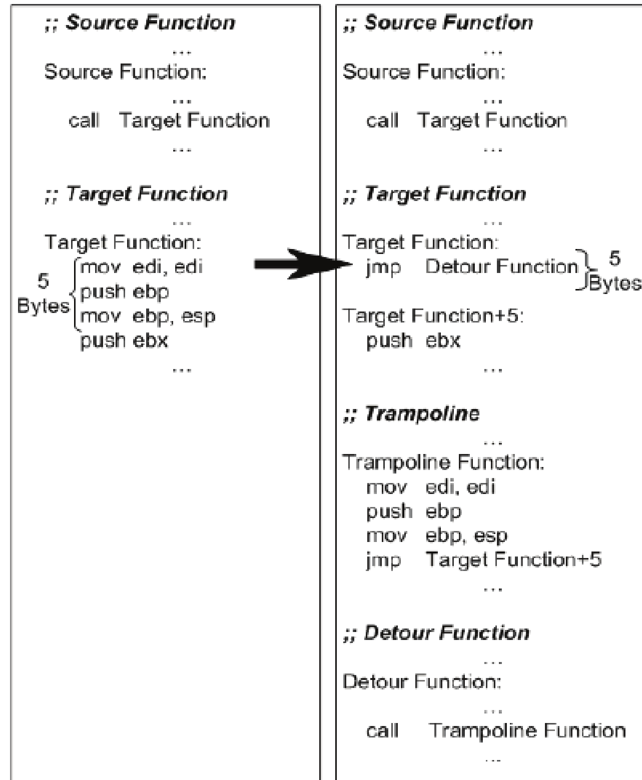
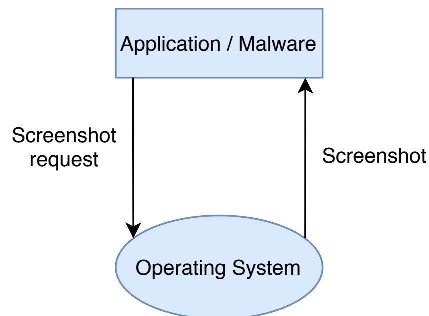


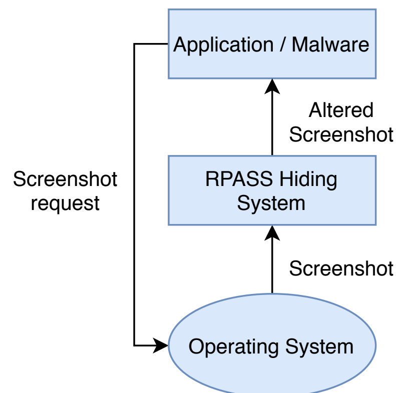
Figure 8.1: Normal function-images vs. Microsoft Detours function-images ([15]).

The difficulty, in our case, is that the API calls that compose the screenshot-taking sequence are not used for the sole purpose of taking screenshots. For example, the BitBlt function, which copies the content of a source DC into a Bitmap object, can perfectly be called by an application that does not take screenshots. Therefore, we have to make sure that the intercepted function is indeed part of a screenshot sequence before redirecting it to our image-altering code (Algorithm 1). To the best of our knowledge, this way of using the hooking functionality (i.e. by redi-

recting the program to our code only if the function is called in a specific context) is novel.



a. Classic way of requesting and receiving a screenshot



b. Screenshot interception in order to apply hiding algorithms before file delivery to the application

Figure 8.2: RPASS

Algorithm 1 Image-altering algorithm.

```
1: function SCREENSHOT
2:     if the parameters indicate that the function is called for screenshot taking then
3:         Jump to Alteration function
4:     else
5:         The normal behaviour of the function
6:     end if
7: end function
8: function ALTERATION
9:     The normal behaviour of the function
10:    Apply a hiding algorithm to the obtained screenshot
11:    Return the altered image
12: end function
```

8.2 Developing several retinal persistence based algorithms

Retinal persistence (also known as "afterimage effect") is a visual illusion in which retinal impressions persist after the removal of a stimulus. It is believed to be caused by the continued activation of the visual system [252].When an image is replaced by another image during a period of less than one-fifteenth of a second, there is an illusion of continuity [150]. This implies that if subparts of an image are displayed at high speed, the human viewer can see the whole image with no changes, though one frame alone only contains a subpart of the original information.

The mechanism we propose to alter the images returned by screenshot functions introduces dynamically hidden areas on the image.

Note that this solution does not completely prevent malicious screenshot exploitation, but only makes it more difficult. Indeed, with a sufficient number of screen-

shots, it is possible to see the whole screen's content. This limitation comes from the important requirement that users should not be involved and that, at the same time, legitimate screenshot-taking applications must continue working as intended.

The process used to alter the image must not require too complex and important computations to be able to display the frames at a high enough rate. This allows legitimate screenshot-taking application already taking screenshots at a high frequency (such as Skype screenshare) to keep working thanks to retinal persistence.

However, this solution prevents legitimate applications that are taking screenshots punctually from having a complete and clean screenshot, as they cannot leverage retinal persistence. Two solutions have been studied to remedy this challenge:

- Solution 1: Offering another function that returns an unaltered screenshot. To avoid it being abused by malware, we can restrict the frequency at which such a function can be called (e.g. $f = 0.1$ frame per second (FPS)). This time limit applies to all processes which means that only one process can take a screenshot during any given $1/f$ seconds. Malware that want a real time view of the victim's activity (e.g. in the case of a password input using a virtual keyboard) will have to use the altered screenshots. This solution is illustrated in Figure 8.3.
- Solution 2: Allowing a clean screenshot to be taken only when the user presses the "Print Screen" physical key. Measures should be taken at the hardware level to avoid the key-press being simulated.

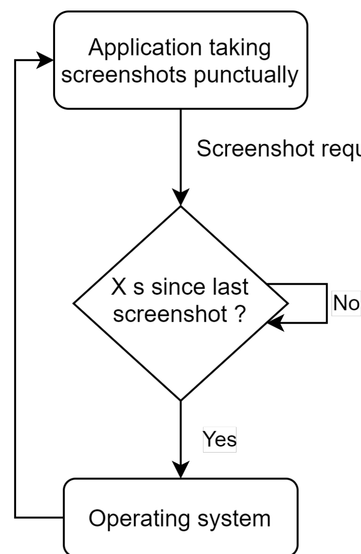


Figure 8.3: Punctual screenshot: solution 1.

On the one hand, Solution 1 does not require any hardware change to be applied and can therefore be deployed immediately on old devices. However, a malware that takes screenshots at a frequency lower than f will be unaffected by our countermeasure.

On the other hand, Solution 2, although it requires hardware modifications to ensure that the "Print Screen" keypress cannot be simulated, appears to be the most secure solution in the long run.

To be used widely for general screen content protection, our approach must respect three main constraints: security (preventing the screenshot exploitation by the attacker, be it automatic or manual), usability (as users of legitimate applications must not be overly disturbed) and real time (so that incomplete images are generated and displayed at a high frequency to leverage retinal persistence).

However, these constraints can be contradictory. This is, for instance, the case with security and usability. Indeed, to achieve better security, an important portion of the screen must be hidden; this is done at the expense of usability.

To achieve the best trade-off between these constraints, three main parameters can vary: the pattern used in the hidden areas (Section 8.2.1), the way areas to be hidden are determined (Section 8.2.2) and the frequency at which incomplete images are displayed (Section 8.2.3).

We implement several algorithms that offer different options regarding these three parameters.

8.2.1 Pattern used inside hidden areas

We propose three patterns to hide areas on altered screenshots: uniform colour (Section 8.2.1.1), gaussian blur (Section 8.2.1.2) and a hierarchical pattern (Section 8.2.1.3).

Each one of the proposed patterns has strengths and weaknesses with regards to the three constraints evoked earlier (usability, security, and real-time).

Uniform patterns

The simplest pattern that can be used to hide a part of the screen is a uniform colour, as illustrated in figure 8.4.

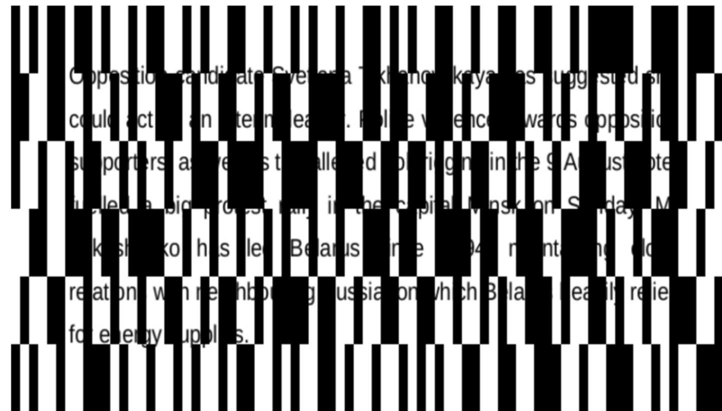


Figure 8.4: Example of uniform pattern.

This pattern has the advantage of not requiring any additional computation. Indeed, it is independent of the hidden content. This makes it the most suitable pattern for real-time use.

However, hiding the screen's content with a colour that is independent from it is not an optimal solution for user comfort [13].

Gaussian blur

We implement a second filter that uses blur, as illustrated in figure 8.5 Blur allows us to minimise the differences between hidden and visible areas.

Figure 8.5: Example of Gaussian blur pattern.

More precisely, we use Gaussian blur, which is known to be one of the best blurring algorithms for preserving edges [253]. Thanks to this property, it is harder to detect by an algorithm.

The level of blur can be set to obtain the best trade-off between usability and security. The radius of the blur defines the value of the standard deviation to the Gaussian function, i.e. how many pixels on the screen blend into each other; thus, a larger value will create more blur while value of 0 leaves the input unchanged.

However, contrary to the uniform pattern, the blur pattern varies according to what is displayed inside the hidden area. As a result, it requires to apply a filter on each pixel of the image, which can be detrimental to real-time display. Moreover, even if the blur level is low, blur detection algorithms can detect the parts of the image that are blurred as the contrast is much lower than in other areas (e.g. in the case of text). This can allow malware programs to automatically discard the blurred zones and “superimpose" the different incomplete screenshots, which would make the solution insecure.

Hierarchical patterns

The third pattern we implement tries to address this issue by hiding parts of the screen while keeping a high contrast, as illustrated in figure 8.6. We introduce a level of hierarchy by dividing each hidden area into a given number of columns. These columns are then randomly mixed. This allows us to keep a high contrast in the hidden areas, which makes it harder for attackers to automatically know which parts of the screen are hidden. However, the computation required at runtime might reduce the frequency at which incomplete screenshots can be displayed.

Figure 8.6: Example of hierarchical pattern.

8.2.2 Determining hidden areas

Two categories of algorithms have been implemented to determine which areas of the screen should be hidden at each iteration: periodical algorithms (Section 8.2.2.1) and random algorithms (Section 8.2.2.2).

Periodical algorithms

For the sake of comparison with the approach proposed by Park et al. [13], we implemented vertical bars that are periodically sliding on the screen (figure 8.7). The space between the bars is equal to the bars' width. This value can be chosen ranging from ten pixels to one hundred pixels. We also implement horizontal bars with configurable height, diagonal bars with configurable dimensions, and concentric circles (figure 8.8).

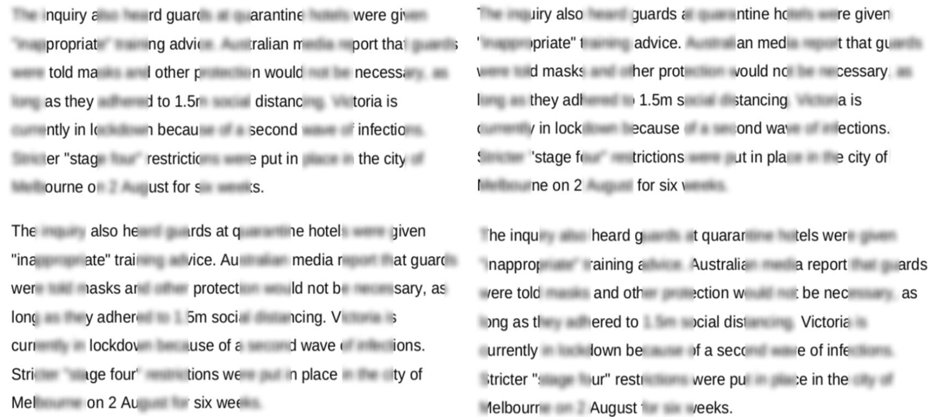


Figure 8.7: Examples of text screenshots altered with the periodical vertical sliding bars algorithm (captured at different times)

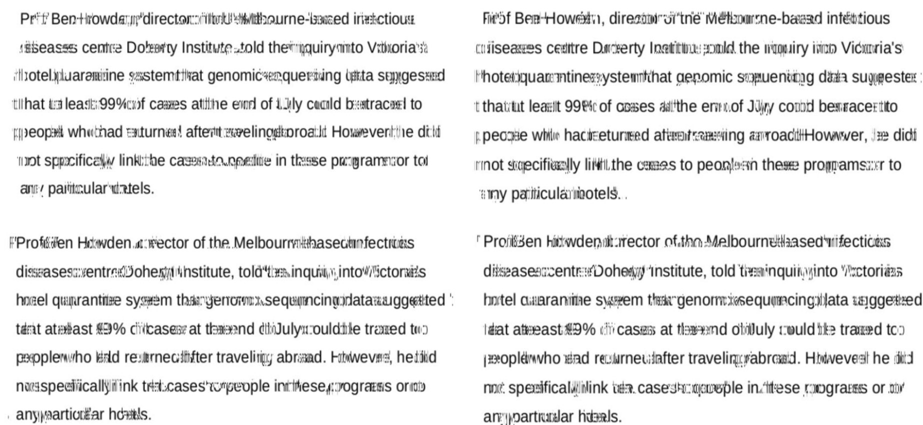


Figure 8.8: Examples of text screenshots altered with the periodical concentric circles algorithm (captured at different times)

Random algorithms

The first algorithm we propose is highly configurable by means of the following parameters.

Number of patterns The screen (with dimensions W and H) is divided into identical rectangles with configurable height (h) and width (w).

The total number of rectangles is $\frac{W*H}{w*h}$.

The smaller the w and h parameters, the more calculations there will be in the algorithm and the more the execution time will increase. However small values for these two parameters allow us to hide numerous, small portions of the screen.

Probability of being hidden Each rectangle with coordinates (i, j) has a probability $P_{ij}(k)$ of being hidden at iteration k, with i ranging from 1 to $I = \frac{W}{w}$ and j ranging from 1 to $J = \frac{H}{h}$.

The probabilities are initialised to 0.5:

$\forall\ (i, j)$, $1 < i < I$, $1 < j < J$, $P_{ij}(0) = 0.5$.

Probability variation The probabilities are then dynamically modified depending on whether the rectangle was hidden in the previous iteration or not. Let $H_{ij}(k)$ be a Boolean that represents whether the block with coordinates (i, j) was hidden at iteration k. When the rectangle is hidden in iteration k ($H_{ij}(k) = 1$), we subtract a number E from its probability of being hidden in the next iteration, with $0 < E < 1$. When it is not hidden ($H_{ij}(k) = 0$), we add this number E to the probability:

$P_{ij}(k + 1) = P_{ij}(k) - E$, if $H_{ij}(k) = 1$,

$P_{ij}(k + 1) = P_{ij}(k) + E$, if $H_{ij}(k) = 0$.

This number E is configurable. When it is high, rectangles have a high probability of alternating between hidden and visible. When E has a low value, there is more randomness because the probabilities will stay at around 0.5.

Maximum numbers of hidden and visible neighbours Two constraints have been added to limit the randomness. The first one states that no more than U

neighbouring rectangles can be hidden in the same iteration. This first constraint aims at making the solution more usable by limiting the maximum area of the screen that can be hidden. The second constraint states that no more than V neighbouring rectangles can be visible in the same iteration. This second constraint aims at making the solution more secure by limiting the maximum area of the screen that can be visible on a given screenshot.

Algorithm's principles After an initialisation phase, in which all parameters' values (lines 1 to 10) are fixed, the algorithm starts processing the first pattern (top left of the screenshot). For each pattern, a random number is drawn to determine whether it will be hidden in the current iteration. Depending on the result, the probability for the next iteration is updated. The probabilities of the neighbouring rectangles for the current iteration are modified if the maximum numbers of hidden or visible areas are reached (Algorithm 2). Drawing a random number for each area prevents anticipating whether an area will be hidden or visible most of the time, which, in turn, prevents screenshot reconstruction using a majority rule.

Examples of screenshots altered using this algorithm can be found in figures 8.9 and 8.10.

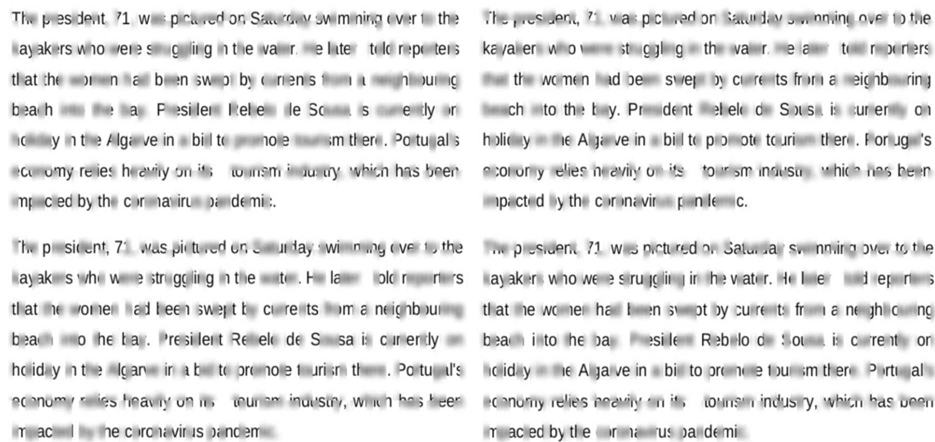


Figure 8.9: Examples of text screenshots altered with the random vertical rectangles algorithm (captured at different times)

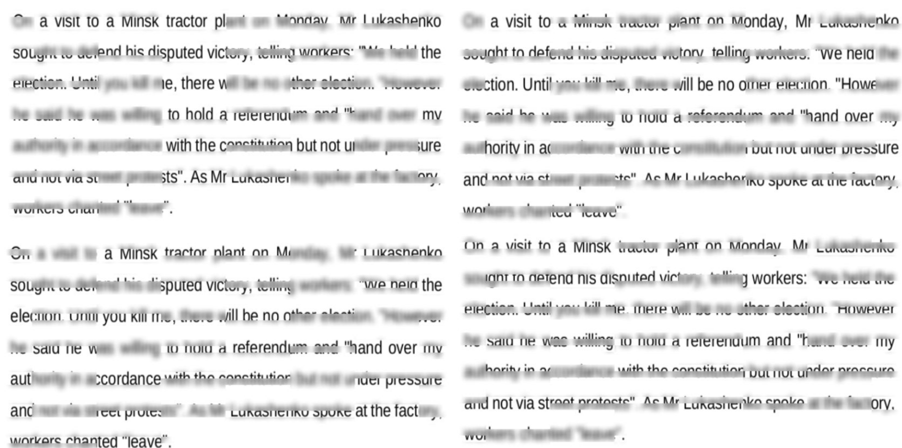


Figure 8.10: Examples of text screenshots altered with the random horizontal rectangles algorithm (captured at different times)

Algorithm's complexity Let n be the number of rectangle patterns. The number of elementary operations at each step of the algorithm is as follows:

- line 12: $n()$.
- line 13: 1.

- line 14: 1.
- line 15: 5.
- line 16 to line 26: 9.
- line 28 to line 40: 9.
- line 41: 1.

Algorithm 2 Hidden patterns in successive images.

```
1:  U: is the maximum number of successive hidden patterns on the same row or
    in the same column.
2:  V: is the maximum number of successive visible patterns on the same row or
    in the same column.
3:  ur: is the number of successive hidden neighbouring patterns on the same row
    as the current pattern.
4:  vr: is the number of successive visible neighbouring patterns on the same row
    as the current pattern.
5:  uc: same as ur but for columns.
6:  vc: same as vr but for columns.
7:  P_ij: probability that pattern ij is hidden.
8:  E: parameter indicating probability increasing or decreasing step.
9:  Initialisation:
10: P_ij = 0.5 ∀i, j
11: CurrentPattern = Pattern_00
12: for CurrentPattern do
13:     Generate a = randomNumber in [0;1]
14:     if a < P_ij then
15:         if ur < (U − 1) and uc < (U − 1) then
16:             hide CurrentPattern
17:         else
18:             hide CurrentPattern
19:             if uc = U − 1 then
20:                 P_ij+1 = 0
21:             end if
22:             if ur = U − 1 then
23:                 P_i+1j = 0
24:             end if
25:             P_ij = P_ij − E
26:         end if
27:     else
28:         if vr < (V − 1) and vc < (V − 1) then
29:             show CurrentPattern
30:         else
31:             show CurrentPattern
32:             if vc = V − 1 then
33:                 P_ij+1 ← 1
34:             end if
35:             if vr = V − 1 then
36:                 P_i+1j = 1
37:             end if
38:             P_ij = P_ij + E
39:         end if
40:     end if
41:     CurrentPattern = NextPattern
42: end for
```

The maximum number of elementary operations is $n*(1+1+5+9+9+1) = 26n$. Thus the complexity of our algorithm is linear and equal to *O(n)*.

Circle patterns We implement a second algorithm where the shapes are circles instead of rectangles. The radius and the minimum distance between two centres are given as parameters. At each iteration coordinates pairs are randomly chosen. Then, circles with the specified radius are drawn with these coordinates pairs as their centres. These circles can overlap if the distance between two centres is smaller than two times the radius. Here, the advantage is that the transition between hidden and visible areas is smoother, which could improve usability. However, choosing random numbers and drawing circles at each iteration implies important computations, which limits the frequency at which images can be displayed. Examples of screenshots altered using this algorithm can be found in Figure 8.11.

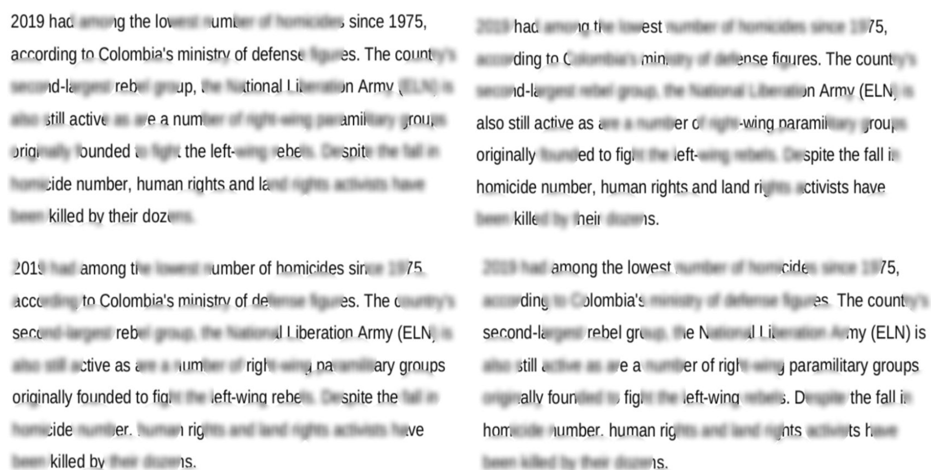


Figure 8.11: Examples of text screenshots altered with the random circles algorithm (captured at different times).

8.2.3 Frequency of display

The frequency at which images are displayed is a key factor. Indeed, retinal persistence precisely relies on displaying images at a high frequency. It has already been shown that greater frequency implies a higher usability [13].

However, it seems that when a certain threshold is reached, increasing the frequency does not increase the usability much anymore [13]. This can be explained by the fact that, for the retina to keep an image "in memory", it must be shown for a minimum duration.

Imposing a very high frequency ($f < 25ms$ i.e. $f > 40$FPS) is not ideal either. Indeed, it limits the computations possible to be performed on each screenshot because of the time constraint, which could be detrimental to security. Moreover, the number of incomplete screenshots sent over the network would explode, without a guarantee of improving usability.

To test these different implications of frequency, each algorithm can have a frequency ranging from 1ms to 1000ms.

8.3 Hypotheses

As discussed in Section 8.2, for our approach against screenloggers to be effective and usable by a wide spectrum of users, it must reach the best trade-off between three main constraints: security, usability and real-time.

Before thoroughly testing the various screenshot alteration algorithms that we are proposing, we can already make a number of hypotheses about the performance of some parameter configurations on the three criteria. These hypotheses arise from the characteristics of our different algorithms and their behaviours:

- Security: When black and blur patterns are used, we believe it will be possible to automatically reconstitute a complete screenshot from incomplete ones, with more or less computation involved. In contrast, when the hierarchical pattern is used, it is less certain that reconstruction will be possible. Experiments with state of the art OCR algorithms will be conducted to assess this hypothesis.

  Besides the pattern used, the way hidden areas are chosen also has an influence on security. Indeed, predictability is the main obstacle to the effectiveness of periodical patterns. In the same way, the more randomness there is to random patterns, the more we believe the solution will be secure. As a result, a low probability update (E) between successive iterations should yield the best security. Moreover, to minimise the limits posed by the U (respectively, V) parameter on security, this parameter should be maximised (respectively, minimised).

- Usability: The blur and hierarchical patterns should give the best results because there will be less contrast with visible areas than when the black pattern is used.

  The size of the hidden area should also have an impact on usability. Indeed, we can suppose that wide hidden areas will be more visible to the user and thus, more detrimental to usability. In the case of the random rectangles algorithm, the parameters that have an incidence on the size of the hidden areas are w, h, and U.

  Regarding frequency, up to a certain limit, a higher frequency should yield greater usability.

- Real-time: Large black rectangles or vertical bars should allow image files generation at the highest speed. On the other hand, the use of blur or hierar-

chical small rectangles or circle patterns, due to the volume of calculations that they require, should be the slowest configurations.

The hypotheses made in this section have been assessed through a rigorous and in-depth testing phase which is the subject of Chapter 9.

9 | Evaluation and Security Analysis of the Proposed Countermeasures

Different tests were conducted to compare the proposed algorithms based on four key criteria (Section 9.1). After presenting our methodology for evaluating our algorithms using these criteria (Section 9.2), we show the results we obtained (Section 9.3) and discuss them (Section 9.4).

9.1 Evaluation criteria

The four criteria we used for the evaluation of our algorithms are: security, usability, real-time and network bandwidth.

9.1.1 Security analysis

As explained in Section 8.2, our approach does not aim at making malicious screenshot exploitation impossible because it is possible to reconstitute the original screen content from incomplete screenshots.

Therefore, the security of our model depends on the number of screenshots the adversary would need to reach their objective.

Two kinds of attacker's goal can be distinguished: getting all the content displayed on the screen and getting particular sensitive information displayed on the screen.

In the first case, we look for the number of screenshots needed to get the whole screen content. Due to the random nature of our algorithms, this number will vary from one execution to the other. Let $P1_n(A)$ be the probability that the attacker

will need n screenshots or less to read the whole screen content when algorithm A is used. Our objective is to find the maximum N such that:

$P1_N(A) < \epsilon$, with ϵ a small positive number, $0 < \epsilon < 1$.

In the second case, we look for the number of screenshots needed to get a particular sensitive information displayed on the screen. Let $P2_n(A)$ be the probability that the attacker will need n screenshots or less to read the sensitive information when algorithm A is used. Our objective is to find the maximum N such that:

$P2_N(A) < \epsilon$, with ϵ a small positive number, $0 < \epsilon < 1$.

9.1.2 Usability

In order for our approach to be scalable and usable in a general way, legitimate screenshot-taking applications must remain usable even with altered screenshots.

Contrary to existing works, we concretely measure usability with actual users and not theoretical measures such as Peak signal-to-noise ratio (PSNR).

Three metrics were used to measure usability:

- Reading time: An objective measure representing the supplementary time $\Delta t(A)$ the users need to read a text displayed using algorithm A compared to a text displayed normally. With T the average time needed to read a clear text and $t(A)$ the average time needed to read a text altered using algorithm A, $\Delta t(A)$ is given by the formula:

  $\Delta t(A) = (\frac{t(A)}{T} - 1) * 100.$

  The use of reading time to measure usability is inspired by the CAPTCHA literature, which frequently uses solving time as a measure of the model's usability

- Reading accuracy: An objective measure that stems from the requirement that the different algorithms we implemented must not prevent the user from getting valid information from the screen. Tor each algorithm A, reading accuracy is measured using the error rate $e(A)$, which is a function of the number of errors made by the user when completing a task. As the solving time, the error rate is frequently used to assess the usablity of CAPTCHA models
- Usability score: A subjective feedback given by the the user regarding reading comfort. For each algorithm A, it consists in a grade $s(A)$ ranging from 0 (very difficult and unpleasant) to 10 (very easy and pleasant).

These metrics were designed to measure the usability of our algorithms on text. We chose not to include images in this usability test because we think it would require further optimisation and a specific usability test which could be conducted as future work.

9.1.3 Real-time

We have seen in section 8.2.3 that frequency is a crucial aspect for the usability of our approach.

As explained before, increasing frequency of display up to a certain threshold increases usability.

However, important on the fly computations can limit the maximum frequency that can be reached with a given algorithm. To ensure a certain comfort of use, it is essential that the algorithms used to hide parts of the screenshots allow real-time execution. Indeed, the fluidity of the images succession on the screen should not constitute an obstacle to the use of countermeasures.

9.1.4 Network bandwidth

We have seen in section 8.2.3 that frequency is a crucial aspect for the usability of our approach.

Still, as much as increasing frequency improves usability, sending more screenshots can result in a congestion of the network if used in a general context as we intend to do.

However, since the screenshots are altered with many parts hidden, with good compression algorithms, one could consistently reduce their size. We would therefore be able to send more screenshots but with the same network bandwidth consumption that when no countermeasure is used.

The objective is to determine, for each algorithm, to what extent we can increase the frequency of screenshots without increasing network use.

In order to do this, we first had to determine the compression ratio r that we can obtain with each algorithm (r= size of the compressed altered screenshot/size of the compressed normal screenshot).

Then we did the following computation: (FPS with normal screenshots) x 1/r.

9.2 Methodology

In this section, we present the methodology used to assess our models using the criteria explained in Section 9.1.

Some aspects of our approach (security against a human adversary, usability) required user tests to be conducted.

We gathered 119 users and made them pass a 20 minutes test on our 'Persistest' website [254]. As this study involved human participants, we obtained the approval of the Departmental Ethics Committee (reference number CS_C1A_20_022). Before undertaking the test, the users gave their informed consent [255].

9.2.1 Security analysis

As explained in our threat model (Section 3.4), regarding the malicious exploitation of the stolen screenshots, two types of adversaries with different capabilities can be found. The first kind of adversary exploits the screenshots manually and the second kind uses algorithms to automatically exploit the screenshots. As manual exploitation involves a human, it is more suited to targeted attacks whereas automatic exploitation allows for large-scale attacks.

We have evaluated the security of our approach both against a human adversary (Section 9.2.1.1) and against an automatic adversary (Section 9.2.1.2). Then, we measured the impact of having to take more screenshots on malware detection (Section 9.2.1.3).

Against human exploitation

In this section, the objective is to determine the number of screenshots the human attacker would need to reach their objective.

Six algorithms have been tested to determine the impact of some parameters on security:

- Determination of the hidden area: random blur rectangles (width=10px, height=75px, U=7, V=3, E=0.2), vertical sliding blur bars (width=80px),

blur snowfall (radius=35px, distance between centers=50px) and shuffled periodical concentric circles.

- Parameter E in the random rectangles algorithm: random shuffled rectangles (width=100px, height=10px, U=6, V=4, E=0.05) and random shuffled rectangles (width=100px, height=10px, U=6, V=4, E=0.4). To ensure interpretability of the results, only E varies between the two tested algorithms.

In the case of human users, it was not necessary to test the impact of the hiding pattern. Indeed, be it black, blur or column shuffling, the user will not be able to read the hidden area.

Frequency was also irrelevant regarding security. Indeed, once the human attacker has the screenshots, they can display them in any way they like to try to exploit them, regardless of the frequency at which they were taken.

As explained in Section 9.1.1, we distinguish between adversaries which goal is to obtain the whole screen content and adversaries looking for a particular sensitive piece of information.

To simulate the first kind of adversary(Step 3 of the user tests), users were faced with an incomplete screenshot of a text (the first one generated by the current algorithm). Before starting Step 3, users received detailed instructions (Figure 9.1). The users had to click on 'Unreadable' if they were not able to read or infer the whole text (Figure 9.2). This would add one incomplete screenshot which is alternated with the first screenshot at a frequency of 25ms. The user has to repeat the operation until they are able to read or infer the whole text. In this case, they click on 'Readable'. This allows us to count the number of incomplete screenshots they needed to read or infer the whole text.

Instructions for step 3

The objective of this step is to determine the number of incomplete images you need to read the whole text (or infer its content with certainty).

For each text, you will first see an incomplete image. To try reading the text, click on “Unreadable” : this will add another incomplete image which will alternate with the first one at a high frequency. While you are not able to read the whole text, click on “Unreadable”. When you manage to read the whole text (or infer its content with certainty), click on “Readable” and go to the next text.

If you click on unreadable 20 times you'll automatically go to the next image.

Once you will have do that with all texts, click on “Proceed to the next step”.

Figure 9.1: Step 3 instructions

Figure 9.2: Step 3 of the user test : determine the number of incomplete screenshots necessary for the user to be able to read the text

To simulate the second kind of adversary(Step 4 of the user tests), the functioning is the same, but the user clicks on ‘Unreadable’ until they are able to see a partic-

ular information (and not the whole screen content). Before starting Step 4, users received detailed instructions (Figure 9.3). To confirm that the user has indeed acquired the desired information, they have to enter it instead of just clicking on a button (Figure 9.4). Their input is compared to the expected result and they cannot proceed to the next step until they enter the right information.

Instructions for step 4

Here, the objective is to determine the number of images you need, not to read a whole text, but only sensitive information on a hotel reservation (arrival date, departure date, city).

As in the previous step, click on “Unreadable” until you successfully read the needed information. Then, enter this information in the three fields bellow each video.

If you click on unreadable 20 times you'll automatically go to the next image.

Once you have done this for each image, click on “Proceed to the next step”.

Figure 9.3: Step 4 instructions

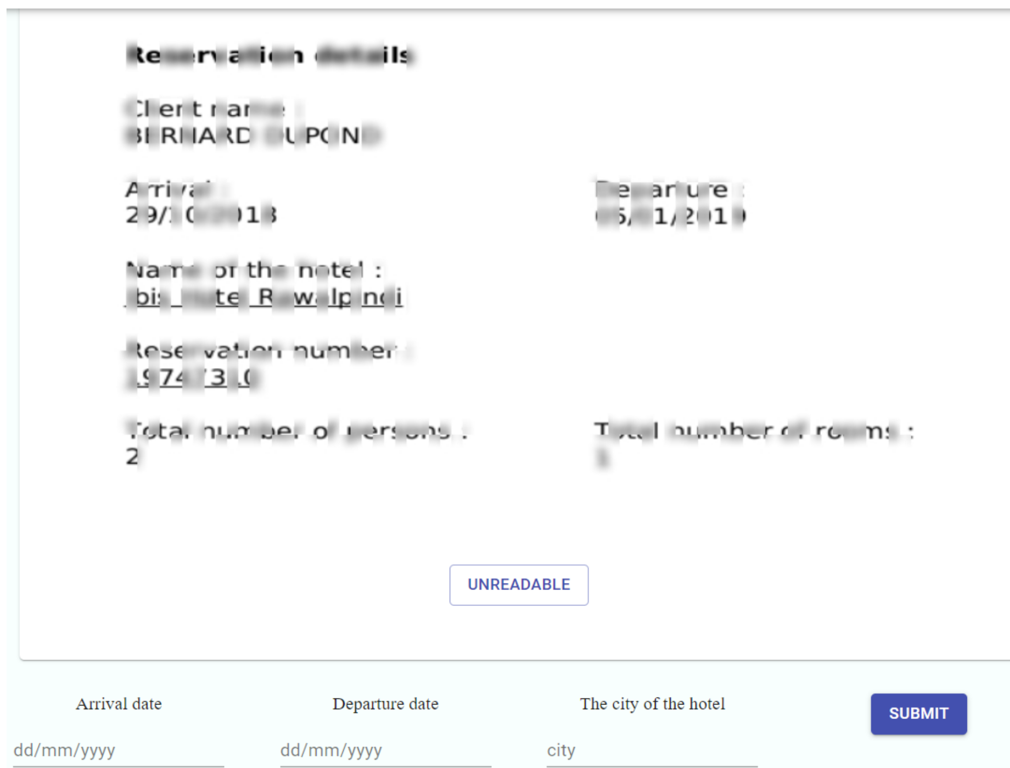


Figure 9.4: Step 4 of the user test : determine the number of incomplete screenshots necessary for the user to read a specific information on the screen (example of a hotel reservation where the user must read the arrival date, departure date and city)

Against automatic exploitation

As presented in our threat model (Section 3.4), we assume that the attacker has at their disposal state of the art OCR tools to exploit the stolen screenshots.

Once again, security is measured by the number of incomplete screenshots needed to obtain the desired information. The difference is that the screenshots are exploited by an algorithm as opposed to a human.

However, counting the number of incomplete screenshots needed would only be meaningful if the malware program is able to identify, on an incomplete screen-

shot, what information to discard and what information to keep . Indeed, if it is not able to do so, it will not be possible for it to merge incomplete screenshots into a complete exploitable screenshot.

Note that a simple majority or minority rule would not allow the hidden areas to be discarded. Indeed, when the pattern is periodical, the hidden zones have the exact same area as the visible ones. This implies that each given zone on the screen is visible 50% of the time and visible 50% of the time. When the pattern is random, the probability of each area of being hidden is initialised at 0.5 and changed at each iteration. It is impossible to predict in advance whether a given area of the screen will be visible or hidden most of the time. For example, in our experiments, one area was found to be visible 8 times out of 20 screenshots whereas another one was visible 13 times out of 20 screenshots.

For the cases in which screenshot merging is possible, as presented in Section 9.1.1, we measured the maximum number of incomplete screenshots N which minimizes the probability of extracting the desired information in two scenarios: full text and sensitive information. For each scenario, we ran the algorithm 100 times on 5 different images. The Tesseract OCR was used for these experiments.

The results have been used to iteratively improve our solution until we found the best possible solution to force the attacker taking more screenshots while at the same time letting the human eye read what is displayed on the screen.

Impact on malware detection

We have conducted experiments to demonstrate that an attacker forced to take more screenshots is more likely to be detected by our detection system (from Chapter 7). The goal is to show that the proposed solution does not only make

the attacker's task harder, but also makes them more visible and thus more easily detectable.

To show this, we have simulated an increase in the number of screenshots taken by the malware programs of our dataset. For each malware program, we modified the values of the relevant features (e.g. screenshot frequency, number of packets sent, median time between packets). Three categories of malware programs were distinguished (with N the number of screenshots the attacker is forced to take determined with previous experiments):

- Frequency superior to N screenshots per second: no modification.
- Frequency inferior to N screenshots per second: modification of the features to reach a frequency of N screenshots per second.
- No frequency (screenshots triggered by specific non-periodical events): features are modified so that each screenshot taken is replaced by N screenshots.

Even if the malware program tries to hide by injecting itself in other running processes or dividing between multiple processes, we will see an abnormally high number of screenshots taken in background.

9.2.2 Usability

The objective of our usability study was to measure the performance of several obfuscation algorithms and parameters combinations according to the metrics presented in Section 9.1.2.

Design

The independent variables of our experiment are the pattern used inside hidden areas, the algorithm used to determine areas to be hidden (with its different parameters) and the frequency at which altered screenshots are shown. All these variables are within-participants: each user was presented with all test cases. Even if this choice reduced the number of test cases, we chose to have the maximum amount of data for each case for the sake of statistical significance.

The dependant variables are the metrics presented in Section 9.1.2: $\Delta t(A)$, $e(A)$ and $s(A)$.

In order to test the impact of each independent variable on the dependant variables, we tested several parameter combinations. Parameters vary one at a time to observe the individual impact of each. For instance, when the hiding pattern varies, the determination of hidden areas and the frequency of display remain constant.

Pattern used inside hidden areas As seen in Section 8.2.1, we implemented three ways of hiding areas on the screen: colouring in black, blurring and shuffling columns. To compare these three solutions, we tested algorithms where the only varying parameter was the pattern used to hide determined areas on each incomplete screenshot (same determination of hidden areas and same frequency). We compared black and blur level 4 using the random rectangles algorithm with the following parameters: (width=10px, height=75px, U=7, V=3, E=0.2, frequency=50ms). We compared blur level 4 and shuffling columns using the random rectangles algorithm with the following parameters: (width=100px, height=10px, U=6, V=4, E=0.2, frequency=25ms).

Determining hidden areas As described in Section 8.2.2, we implemented six ways of determining which part of the screen should be hidden at each moment.

Here again, we compared these methods by only varying the identification of hidden parts (same hiding pattern and same frequency).

We compared random rectangles with parameters (width=10px, height=75px, U=7, V=3, E=0,2), vertical sliding bars with parameter (width=80px) and snowfall with parameters (radius=35px, distance between centers=50px) using blurred hidden areas and a frequency of 50ms. We compared random rectangles with parameters (width=100px, height=10px, U=6, V=4, E=0.2) and periodical concentric circles using shuffled hidden areas and a frequency of 50ms.

We chose not to test periodical horizontal bars because it turned out that, as tests are in the form of lines, hiding complete lines was to detrimental to usability, the user having to wait until the bar passes to resume the reading of the current line.

Frequency of display It is well established that frequency of display is a important parameter to improve usability when using retinal persistence based approaches [13].

However, in order to be able to deploy our solution in a general context, it is important to take into account the frequencies currently used by major legitimate screenshot-taking applications. Indeed, our objective is that these applications should remain usable even with altered screenshots.

Therefore, we measured the FPS of 50 legitimate screenshot-taking applications used in different contexts: screensharing, remote control, screen casting, employee control, children control. To this end, we ran these applications and looked for the frequency at which screenshot API calls were made using API Monitor execution reports. We found that the screenshot frequency ranges from 1 FPS to 45 FPS.

As a result, we compared a frequency of 25ms (40 FPS) and a frequency of 100ms (10 FPS) using the following algorithm: random rectangles with parameters (width=100px, height=10px, U=6, V=4, E=0.2, hiding pattern=blur level 4)

Materials

We used 10 texts of 350 characters to test our algorithms. These texts were extracted from 10 different BBC news articles. Each time a user took the test, the texts were randomly assigned to the tested algorithms: different users tested algorithms on different texts.

Our usability test did not require the participant to be physically present. The test is deployed on a publicly deployed website [254]. As a result, participants took the test in different physical conditions, either on mobile or desktop devices.

Participants

A first subset of 20 participants was recruited among Oxford students using snowball sampling.

The rest of the users (99) were recruited through the SurveyCircle website.

Due to the GDPR and ethics requirements, we did not collect any personal information on users (e.g. name, age, address) other than the fact that they had a normal vision and that they were wearing glasses when needed.

Procedure

The participants were first introduced to the objectives of the experiment and gave their consent for participating.

Two scenarios were used to measure usability according to the three dependant variables:

- In the first scenario (Step 1 of the user tests), the user is passive and simply has to read different texts with the same number of characters (350). On each text, a different algorithm is applied. One of the texts is displayed normally for the sake of comparison.

  Before starting Step 1, users received detailed instructions (Figure 9.5).

Instructions for step 1

In this first step, the objective is to measure the time you will need to read nine texts.

The first text will be displayed normally to have a reference. The eight other texts will be displayed in the form of videos in which incomplete images of the text alternate at a high frequency.

For each of these texts, you will first have to click on "Start reading" to start reading it. As soon as you finish reading, click on "I have read the whole text".

Then you will have to grade the ease of reading from one to ten using the sliding bar under each text.

Once this will be done for the nine texts, click on "Proceed to the next step".

Figure 9.5: Step 1 instructions

Then, for each text, they had to click the 'Start Reading' button to see the text (Figure 9.6).

When you click on the "start reading" button, a text will be displayed. You will need to read this entire text and your reading time will be timed. Once you are done reading, click the "I finished reading the entire text" button

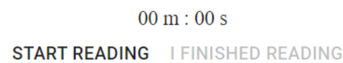
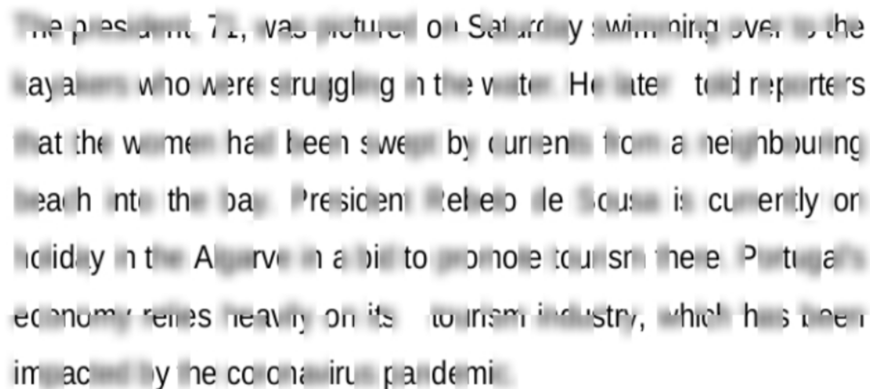


Figure 9.6: Step 1 before clicking on ‘Start Reading’ button

This triggers a timer that stops when users click on the ‘I finished reading’ button (Figure 9.7).



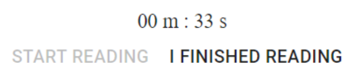


Figure 9.7: Step 1 after clicking on ‘Start Reading’ button

After clicking on this button, users had to rate the visual comfort for the text they just read (Figure 9.8).

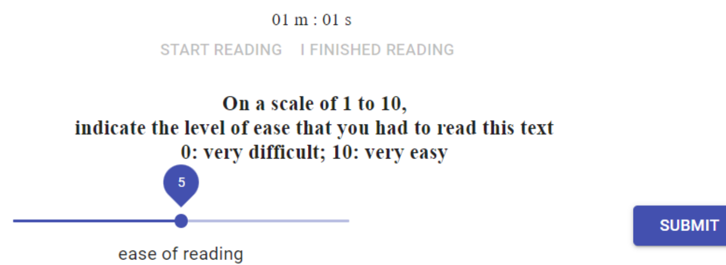


Figure 9.8: Step 1 after clicking on 'I finished reading' button

At the end of Step 1, after reading all texts, users were given the possibility to adjust all the grades (Figure 9.9).

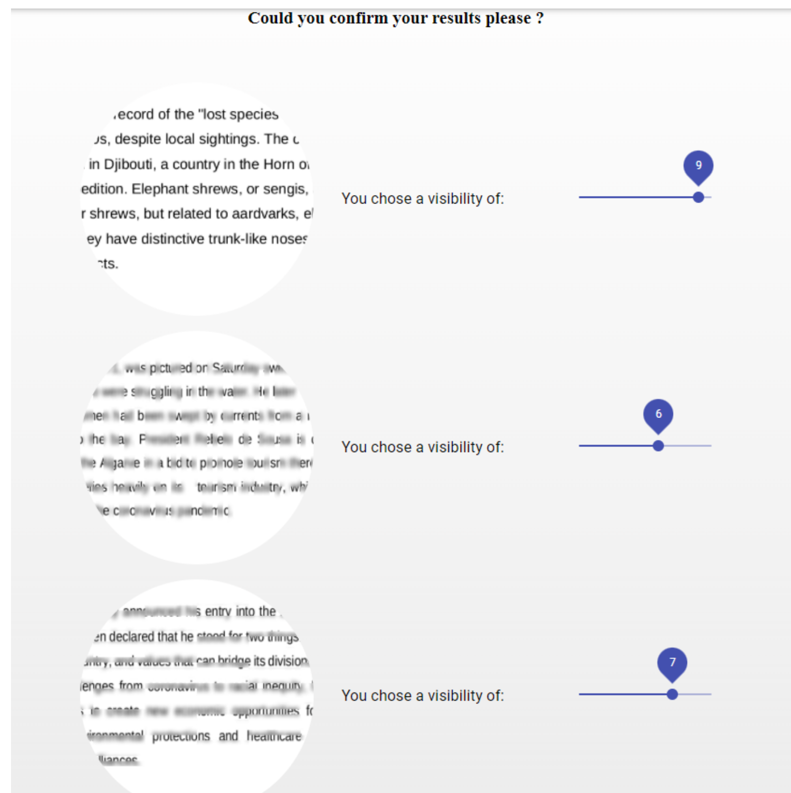


Figure 9.9: Step 1 final screen (confirm usability scores)

This first scenario was designed to measure two metrics: reading time and usability score.

- In the second scenario (Step 2 of the user tests), the user is active and must enter different 5-digits codes displayed for 3.5 seconds with different algorithms and one code displayed normally for comparison. This duration was chosen because it seems to correspond to a reasonable time necessary for an average person to read and memorise a 5-digit code.

Before starting Step 2, users received detailed instructions (Figure 9.10).

Instructions for step 2

The objective is the same as in the previous step: measure the impact of the particular display on your capacity to exploit the information displayed on the screen.

In this step, you will be asked to look at nine images on which five digits are displayed: eight images are displayed in a "weird" way and one is displayed normally to serve as a reference.

For each image, click on "Show the code". You will see a five digits code for 3.5 seconds. You will have to memorise this code, and, when the 3.5 seconds expire, enter it in the corresponding field under the image.

Once this will be done for the nine images, click on "Proceed to the next step".

Figure 9.10: Step 2 instructions

Then, for each algorithm, they had to click the 'Show the Code' button to see a 5-digits code for 3.5 seconds (Figure 9.12, Figure 9.11).

When you click on the "show code" button, a 5 digit code will be displayed for 3.5 seconds. You must copy this code in the box below.

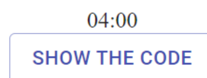


Figure 9.11: Step 2 before clicking on the ‘Show the Code’ button

Figure 9.12: Step 2 after clicking on the ‘Show the Code’ button

After the 3.5 seconds, the code was hidden and users had to enter it to proceed to the next algorithm (Figure 9.13).

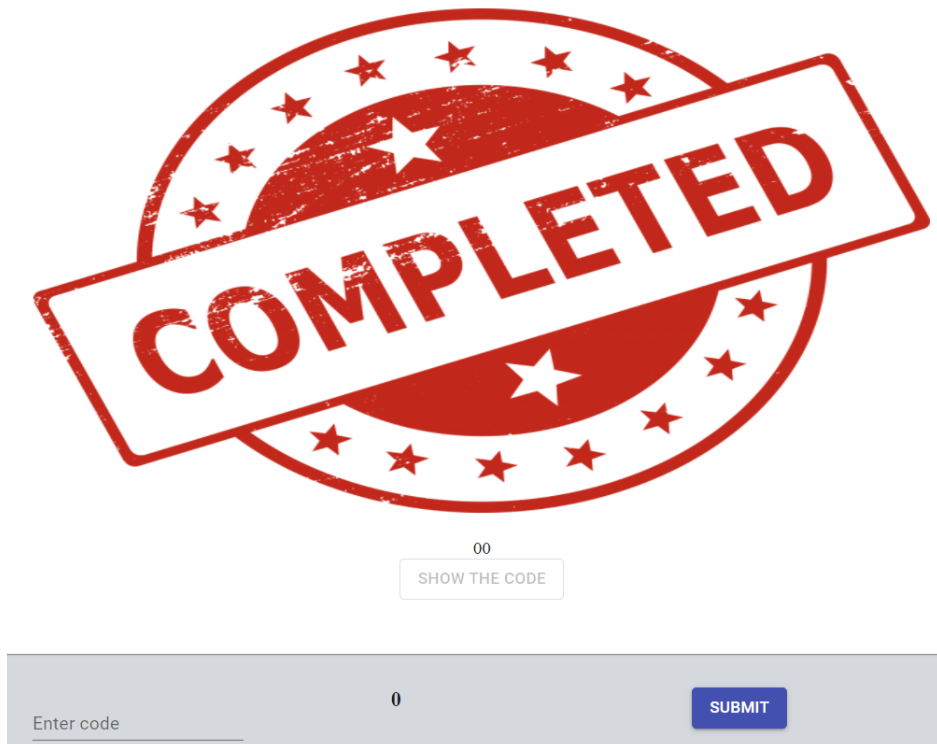


Figure 9.13: Step 2 after 3.5 seconds

This second scenario was used to measure the error rate. Let $r(A)$ be an Integer ranging from 0 to 5 which represents the average number of digits incorrectly entered by the users for algorithm A. The error rate is given by the formula:

$e(A) = \frac{r(A)}{5}$.

9.2.3 Real-time

Our objective was to see how many frames per second can be generated with each of our algorithms.

For each algorithm, we varied the parameters and calculated the number of images generated per second.

For these experiments we used a common PC with the following characteristics:

- Processor: Intel Core i5-5300.
- RAM: 8 Gb.
- OS: Ubuntu 64-bit.
- Disk: 256 Gb SSD.

In addition to the width of the patterns, we varied the set of parameters specific to the random rectangles algorithm (pattern height, number of maximum successive hidden patterns in a row, number of maximum successive shown patterns in a raw, probability step) in order to explore the efficiency of all possible configurations.

9.2.4 Network bandwidth

The objective of this part in our experiments is to figure out how the various algorithms that we studied affect screenshot files sizes.

Indeed, the increase (respectively decrease) in the images sizes after an algorithm is applied could be a disadvantage (respectively advantage) to its use.

To carry out this part of the experiment, we selected five images with different properties in terms of colours, text and background.

- anime.jpg: an image of a cartoon;
- black.jpg: an image of a black background with short blue text;
- color.jpg: an image containing a combination of colours;
- person.jpg: an image of a person;

- text.jpg: an image of black text on a white background.

The images correspond to screenshots resized to (800px * 450px), and the TAR and ZIP standards were used for compression.

9.3 Results

9.3.1 Security analysis

Against human exploitation

From a security point of view, that is, the difficulty for a human to visually extract sensitive information from a screenshot, several interesting observations can be made from Table 9.1. The first column represents the average number of screenshots necessary for a user to read a text, to which a protection algorithm has been applied beforehand. The second column shows the number of screenshots the users needed to obtain some specific piece of information contained in an image to which algorithms were applied.

First, one can notice that, paradoxically, when looking for a specific piece of information, users systematically need more screenshots than when reading a whole text. This can be explained by the fact that, when reading a meaningful text, human users can infer a substantial amount of hidden information, whereas, when having to recopy a date or an exotic city name (column 2), they have to see all the characters without being able to infer them and therefore need more screenshots.

Table 9.1: Number of successive screenshots needed by users.

	Images NB - full text	Confidence interval (p=0.05)	Images NB - sensitive information	Confidence interval (p=0.05)
Algo1 - Random rectangles (10px, 75px, 7, 3, 0.2)	2.85	2.49 - 3.21	5.74	4.91 - 6.57
Algo2- Random rectangles (100px, 10px, 6, 4, 0.05)	8.22	7.35 - 9.09	11.44	10.41 - 12.47
Algo3 - Random rectangles (100px, 10px, 6, 4, 0.4)	4.59	3.82 - 5.36	9.11	7.9 - 10.32
Algo4 - Vertical sliding bars (80px)	7.11	6.41 - 7.81	10.07	9,4 - 10,74
Algo5 - Periodical concentric circles	9.18	7.94 - 10.42	15.37	14.18 - 16.56
Algo6 - Snowfall (35px, 50px)	6.22	5.52 - 6.92	6.70	5.4 - 8

At a first glance, it would seem that algorithms 4 and 5 give very good performances in terms of number of screenshots needed by the user. However, this is a result to be taken with great care because in fact, the periodic pattern moves slowly in these algorithms. Actually, it would suffice to have the first screenshot and the one where the pattern has completely moved (completely left the areas hidden on the first screenshot). Thus, two well selected screenshots would be enough to extract the data.

An interesting observation can be made when comparing algorithms 1, 2 and 3 regarding the orientation of the rectangles used to hide parts of the screenshots. It can be seen, on the results of the tests, that this orientation of the rectangles has an impact on the number of screenshots necessary for reading a text and extracting sensitive information. Indeed, when the rectangles are vertical (algorithm 1), they do not hide a lot of text so users have an easier time inferring the hidden

information and need fewer screenshots, unlike the case of horizontal rectangles (algorithms 2 and 3). When comparing more specifically algorithms 1 and 3, we see that even with a lower E, a lower maximum number of contiguous visible rectangles and a higher maximum number of contiguous hidden rectangles, vertical rectangles are less effective than horizontal ones.

Algorithms 2 and 3 are identical from all points of view except the parameter E (the probability step). From the test results, we can see that Algorithm 3 requires fewer screenshots. This is quite logical because, in Algorithm 3, $E = 0.4$, which concretely means that if a pattern is hidden in screenshot number *I*, its probability of being displayed in screenshot number *I + 1* will increase by 0.4 instead of 0.05. Therefore, the number of screenshots required for all areas to be visible at least once is significantly lower.

Against automatic exploitation

Table 9.2 reflects the number of screenshots needed by an automatic attacker to read a full text of 350 characters (the same texts as the ones used for human exploitation). For each algorithm, the values represent the probability for the OCR of reading the whole text with a given number of screenshots (the probability is 0 for the numbers below 6). For instance, the first value means that, in 8% of cases where algorithm 1 was applied, the OCR needed six screenshots or less to read the text.

We can see that, contrary to human attackers, OCR algorithms need more images when vertical rectangles are used than when horizontal rectangles are used. This might be because vertical rectangles are to thin to disturb human inference but are sufficient to mislead OCR algorithms.

Table 9.2: Probability that a state of the art OCR will read the full text (350 characters) with different numbers of incomplete screenshots.

	6	7	8	9	10
Algo1 - Random rectangles (10px, 75px, 7, 3, 0.05)	8%	16%	20%	48%	100%
Algo2- Random rectangles (100px, 10px, 6, 4, 0.05)	12%	48%	100%	100%	100%

Table 9.3 focuses on the case where the attacker's objective is to get a specific piece of information from the screen (in this case from the same hotel bookings as for human adversaries).

Table 9.3: Probability that a state of the art OCR will extract a specific piece of information (350 characters) with different numbers of incomplete screenshots.

	2	3	4	5	6	7
Algo1 - Random rectangles (10px, 75px, 7, 3, 0.05)	14%	22%	58%	76%	86%	100%

Impact on malware detection

Table 9.4 shows that, even if the improvement is relatively small, forcing malware programs to take more screenshots makes the false negative rate drop to 0 (all malware programs are detected as malicious). In parallel, the number of legitimate applications classified as malware is lowered as well.

Table 9.4: Detection results when we simulate an increase in screenshot frequency.

	Accuracy	FN	FP	Precision	Recall	F-Measure
Regular frequency	97.409%	0.009	0.046	0.963	0.991	0.977
Increased frequency	98.446%	0.000	0.034	0.972	1.000	0.986

9.3.2 Usability

As described in the previous sections, usability was measured in two scenarios.

- Passive user: The results obtained are summarised in Table 9.5. The first column shows the reading time increase for each algorithm compared to an unaltered text ($\Delta t(A)$). The second column is the average usability score given by the users ($s(A)$).

Table 9.5: Passive user usability evaluation.

	$\Delta t(A)$	$\Delta t(A)$ confidence interval (p=0.05)	$s(A)$	$s(A)$ confidence interval (p=0.05)
Static image	0%	-7% - 7%	8.63	8.23 - 9.03
Algo1 - Random blur rectangles (10px, 75px, 7, 3, 0.2, 50ms)	14%	4% - 24%	5.07	4.64 - 5.5
Algo2 -Random blur rectangles (100px, 10px, 6, 4, 0.2, 25ms)	-4%	-11% - 3%	5.74	5.32 - 6.16
Algo3 -Random blur rectangles (100px, 10px, 6, 4, 0.2, 100ms)	4%	-5% - 13%	4.81	4.4 - 5.22
Algo4 - Vertical sliding blur bars (80px)	34%	24% - 44%	4.22	3.85 - 4.59
Algo5 - Blur Snowfall (25px, 50px, 50ms)	15%	2% - 28%,	4.37	3.97 - 4.74
Algo6 - Random black rectangles (10px, 75px, 7, 3, 0.2, 50ms)	24%	11% - 37%	2.59	2.22 - 2.96
Algo7 - Random mixed rectangles (100px, 10px, 6, 4, 0.2, 25ms))	38%	24% - 52%	2.15	1.81 - 2.49
Algo8 - Periodical mixed concentric circles	29%	13% - 45%	2.89	2.52 - 3.26

First, it is interesting to notice that users have generally underestimated the usability of the various algorithms even if the reading time in each case is only between 0.96 and 1.38 times the time required to read the static text. Indeed the usability scores oscillate from 2.44 to 6.44. Even the normally displayed text has received an average grade of 8.63.

We can also notice that a shorter reading time is not always correlated with a better usability (algorithms 4 and 6).

The parameters of the algorithms appearing in Table 9.5 have been chosen to be able to accurately compare the impact of certain factors on the usability.

In algorithms 2 and 3, we used random blur rectangles with identical parameters except the display frequency of successive images on the screen. It emerges from the results of the tests that the reading time of the text is 96% and 104% of the reading time of a clear text for respectively algorithms 2 and 3. That is to say an increase of 8% whereas the frequency was divided by 4. This increase is accompanied by a decrease in the subjective usability score of only 0.93 points. We can thus say that the increase in the display frequency (greedy in resource use) has only a limited impact on the usability of the tested algorithms.

The results of algorithms 1, 2, 3 and 5 show that blurring areas with random overlapping circles instead of random rectangles produces a lower usability on both measures whereas we anticipated that overlapping patterns would provide smoother transitions and therefore better usability. This may be due to the fact that the snowfall algorithm is completely random and does not use constraints to make sure that a given area will be more likely to appear if it is hidden in a given iteration. Algorithms 1, 2, 3 and 5 also confirm that, even if areas are randomly hidden, and thus not necessarily visible more

than half of the time, the reading time augmentation compared to a clear text only ranges from -4% to 15%.

Algorithms 1 and 6 are the same in all respects except the use of blurring in Algorithm 1 and a simple black colour in Algorithm 6 to hide parts of the image. We see that the use of the black colour increases the reading time by about 10% but at the same time generates the division of the usability score by 2, which therefore marks a significant discomfort caused to users due to the use of black colour.

In the same way, we can see, by comparing the results of algorithms 2 and 7 that the use of mixed rectangles instead of blurred ones significantly impacts the usability by increasing the reading time and reducing the usability score in a significant way. This poor usability of the mixed pattern can be explained by the fact that it is harder to distinguish from real text than blur. This can cause some confusion for the user and increase their reading time.

Regarding periodical patterns (algorithms 4 and 8), we notice that they produce among the highest increases in reading time, second only to random mixed rectangles. This is due to the fact that the user must wait for the pattern to move to be able to read the text under it. We can notice that the subjective grade given by the user is also lower than algorithms 1, 2, 3 and 5. Moreover, contrary to the random algorithms, the vertical bars algorithm allows each area to be visible half of the time. This does not correlate with an improved usability.

- Active user: All of the 119 users correctly returned the five characters for each algorithm. This shows a certain relevance of the parameters selected for each algorithm and also that over a reasonable duration (3.5 seconds),

human users have no trouble reading the masked data using all the tested algorithms.

9.3.3 Real-time

The number of images generated per second for each algorithm are given in Table 9.6.

Table 9.6: Number of images generated per second.

Algorithm	Filter	Pattern width	Images per sec
Random rectangles	Blur	10	16 - 19
		25	13 - 20
		50	17 - 20
		75	14 - 20
	Mixed	10	17 - 20
		25	19 - 20
		50	19 - 20
		75	20
	Black	10	20
		25	20
		50	20
		75	20
Periodical vertical bars	Blur	25	17
		50	18
		80	18
		100	17
	Mixed	25	19
		50	18
		80	18
		100	17
	Black	25	20
		50	20
		80	20
		100	20
Periodical concentric circles	Blur	10	22
		20	20
		25	22
		50	22
	Mixed	10	4
		20	6
		25	6
		50	5
	Black	10	27
		20	28
		25	26
		50	28
Snowfall (*, 50px)	Blur	10	4
		20	2
		25	3
		50	1
	Mixed	10	4
		20	2
		25	1
		50	1
	Black	10	8
		20	3
		25	3
		50	1

As expected, the number of generated images is always higher when the black pattern is used, as it requires less computations. The two other patterns produce similar results except for the concentric circles algorithm for which the use of the mixed pattern drastically decreases performance.

The results that oscillate around 20 images per second are higher than the average frequency currently used by legitimate screenshot-taking applications (11 FPS). It corresponds to a frequency of 50ms which is compatible with the usability tests presented earlier.

Finally, regarding the snowfall algorithm, as anticipated, the drawing of circles takes too much time to allow for a real time display of screenshots using retinal persistence.

9.3.4 Network bandwidth

We applied each pair (hiding algorithm/pattern) to each one of the five images listed in Section 9.2.4. The file sizes obtained allowed calculating the compression rates shown in Table 9.7.

Table 9.7: Images compression rates.

	Blur	Mixed	Black
Random rectangles	0.83	1.28	0.79
Vertical bars	0.78	1.17	0.52
Circles	0.93	1.19	1
Snow fall	0.87	1.42	0.83

It appears quite clearly in the Table 9.7 that the black pattern produces, almost in all cases and with all the algorithms, the best compression rate compared to the other patterns. This is a relatively predictable finding.

We can note that, apart from those involving the concentric circles algorithm and the mixed pattern, all the other combinations give interesting compression rates. These rates are even particularly good with the use of the black filter reaching 52%, which means that the size of the returned screenshot is half the size of the original image.

As legitimate applications take between 1 and 45 screenshots per second (11 on average), it is easy to see the impact of the tested algorithms on outgoing network traffic. Indeed, for the same duration (1 second) and using an identical bandwidth, most of the pairs (algorithm / pattern) allow sending between 20% and 100% more image files. This makes it possible to use higher display frequencies compatible with the use of retinal persistence.

However, in the case of video formats, our approach would increase the amount of data to be sent over the network. Indeed, in such formats, individual frames are not compressed separately. Instead, only the changes between successive frames are encoded. Therefore, with different parts of the screen being hidden in each frame, there would be more changes to record.

9.4 Discussion

Our experiment was carried out on four criteria which seem to have an extreme importance for any effective widespread countermeasure against screenshot-taking spyware. Each of these criteria has brought out a subset of efficient (algorithm / pattern) pairs and, often, another subset leading to degraded performances.

From a security point of view, the random rectangles and the snowfall algorithms have proven to achieve the best performance in terms of the number of screenshots required for the reading of a text or the extraction of a piece of sensitive

information. More specifically, for the random rectangles algorithm, in the case of a human reader, horizontal rectangles were more effective whereas, in the case of an OCR, vertical rectangles consistently yielded better results.

On the usability criterion, the random blur rectangles, vertical blur bars and blur snowfall algorithms obtained the best scores from the panel of testers. This score should nevertheless be put in perspective for the vertical bars algorithm as it significantly increases reading time.

Regarding real time, our experiments allowed to generate altered screenshot at a rate oscillating around 20 images per second when using the random rectangles or vertical bars algorithms. It is an acceptable result compared to the average frequency of legitimate screenshot-taking applications (11 FPS). Furthermore, the usability of a frequency of 50ms (20 FPS) has been shown by the usability study. This part of the experiment also showed the limit of the snowfall algorithm, which produces images at a particularly low rate.

Regarding the bandwidth, the vertical bars algorithm offers the best compression rate. The random rectangles and snowfall algorithms present interesting compression rates as well, allowing to potentially increase the number of images sent by 20%.

The random rectangles algorithm with the blur pattern seems to produce the best results when considering all the criteria. It is therefore a relevant choice as a countermeasure to spyware taking screenshots. However, our multiple experiments, with various parameters configurations of this algorithm, show that the choice of the parameters values must be done with the greatest care.

In the future, we could make our approach more robust by preventing automatic screenshot merging. This would involve using techniques to raise the OCR confidence level of hidden areas, while lowering the confidence level of visible areas.

If this can be achieved, the only way for an attacker to exploit stolen screenshots would be to manually analyse them, which would drastically reduce the scope of screenlogging attacks.

10 | Conclusion

After recapitulating the contributions brought about in this thesis (Section 10.1), we acknowledge the limitations of our work (Section 10.2), and conclude with final remarks (Section 10.3).

10.1 Contributions

10.1.1 Threat analysis (Chapters 4 and 6)

We proposed an in-depth analysis of screen capture functionality of malware, which is quite unknown whereas it is one of the most prejudicial to privacy. This analysis was conducted by analysing over one hundred security reports on screenloggers, emanating from various cybersecurity firms.

This study showed that there are several reasons explaining the lack of knowledge on screenloggers. The main one is that in the majority of cases, a specific event such as the reception of a command, the opening of an application of interest, or a number of mouse clicks, is needed to trigger the screen capture functionality, which explains why it is neglected in the malware detection works.

Across the security reports, we identified recurring steps in the screenlogger operating process. This allowed us to define criteria for establishing a novel screenlogger taxonomy.

By gathering statistics on these criteria, completeness requirements for a screenlogger dataset were defined.

Using the same criteria, 94 legitimate screenshot-taking applications were analysed and statistics were gathered. This enabled to identify promising criteria for screenlogger detection.

10.1.2 Dataset construction (Chapter 5)

Malicious dataset The samples we collected from existing malware datasets were useless for various reasons, including anti-sandboxing techniques and specific screenshot triggering mechanisms which prevented us from studying the screen capture functionality. The reason why this issue was not raised in previous generalist malware detection works is that the datasets they use often contain thousands of malware samples. To extract the training data, the samples are usually automatically run for a given duration, without any interaction.

As a result, we had to build our own dataset by collecting malicious screenlogging tools and source codes and making sure that they were taking screenshots at runtime.

To achieve completeness of our dataset as per the definition we gave, it had to match the behavioural statistics obtained from the security reports. We implemented the first screenlogger generator encompassing all the behaviours found in the security reports at the different stages of the screenlogger operating process. This generator was used to create samples to make our malicious dataset complete by reaching the statistics from the security reports on all criteria.

Benign dataset We created the first dataset of legitimate screenshot-taking applications by gathering applications belonging to five categories, each category having its specificities.

10.1.3 Screenlogger detection (Chapter 7)

We built a first RF detection model using only API calls and network features from the literature. This model was trained and tested using our malicious and benign datasets. Using RFE with Gini importance, we identified the most informative existing features for screenlogger detection.

Then, we built a second model including novel features adapted to the screenlogging behaviour. These features were collected using novel techniques. Particularly, we can cite:

- Using API call sequences to identify specific behaviours. Contrary to existing works which only look at API called in a direct succession using the notion of n-grams, we wrote scripts which keep track of the API calls return values and arguments to characterise some behaviours even if the calls are not made directly one after the other. Numerous different sequences involved in the screenshot-taking process were identified by analysing malware and legitimate applications. These sequences were also divided into three categories depending on the captured area.
- Making a correlations between API calls made by an application and its network activity. During their execution, the API calls and network activity of our samples were simultaneously monitored. This allowed us to extract features such as the reception of a network packet before starting the screenshot activity or the sending of taken screenshots over the network.

When adding these novel features to the detection model, the detection accuracy increased by 3.108%. Indeed, it is well known that a detection model based on less features is less likely to fall into overfitting. Moreover, a detection model

based on features which have a logical meaning and reflect specific behaviours, is less prone to evasion techniques often used by malware authors.

10.1.4 Retinal persistence-based mitigation technique (Chapters 8, 9 and 10)

To account for the cases in which a screenlogger might not be detected, we designed and implemented a novel technique to mitigate malicious screenshot exploitation, with the strong constraint of not requiring any action from the user. This work is the first to impose such a constraint, existing works focusing on preventing screenshot exploitation in sensitive scenarios using intrusive techniques impossible to deploy at a large scale.

We propose to take advantage of the retinal persistence property of the human eye by displaying incomplete screenshots in a quick succession. Instead of using periodical patterns, which predictability is detrimental to security, we introduce randomness to determine which areas of the screen should be hidden at a given iteration. Several parameters are used to limit this randomness to achieve the best trade-off between security and usability.

Furthermore, we also propose a concrete implementation our approach on the Windows operating system by using a novel hooking technique. Indeed, as the API calls involved in the screenshot-taking process can individually be used for other purposes, we had to verify that they are indeed called as part of a screenshot sequence before modifying their functionality.

We analysed the security of our approach against a human adversary and against an automatic adversary. For the latter case, we implemented an adversarial algorithm dedicated to merging incomplete screenshots depending on the pattern used to hide parts of the screen. Then, we counted the number of altered screen-

shots necessary for an adversary, with state of the art OCR tools at their disposal, to reach their goal in two scenarios: a whole text and a piece of sensitive information. Using these results, we conducted experiments to demonstrate that an attacker forced to take more screenshots is more likely to be detected by our detection system.

The usability of our approach was tested with a panel of 119 users with three metrics: reading time, reading accuracy and a subjective usability score. To the best of our knowledge, this is the first time an approach based on retinal persistence was tested with actual users.

Finally, the performance of our approach was tested on two additional criteria: real time and network bandwidth.

10.2 Limitations

The work proposed in this thesis presents five main limitations:

- The size of our dataset: As explained before, the numerous screenlogger executable files found in malware datasets were useless to us because of the impossibility of triggering the screenshot functionality. The malicious tools with an available server part were, unfortunately limited in number.

  Moreover, another constraint on the size of the dataset was the necessity to interact with the samples at run time, which prevented their automatic execution. Indeed, even if we had millions of samples, it would not have been possible to run them all and generate their API calls and network reports because of the time-consuming constraint of interacting at run time to trigger the screenshots.

However, we have established that it is common, in malware detection works targeting a specific malware category, to use datasets with a size similar to ours or even smaller. We also made sure that our detection results are statistically significant.

- The generated samples: Because of the limited number of samples in our dataset, some behaviours were missing or under-represented. We added generated samples to remedy this issue. Because they only have the screenlogging functionality, these samples might not accurately reflect the complete behaviour of screenshot-taking malware in the wild. However, the generated samples only represent a quarter of the total number of malicious samples. Moreover, their screenlogging functionality was implemented on the basis of actual screenlogger source codes.
- The ineffectiveness against attackers with root privileges: As we use API calls to detect screenshot-taking operations, our approach is blind to an attacker with root privileges, which could, for example, access the screen content through the frame buffer. However, we found no records of such an attack and our system still covers the vast majority of screenlogger attacks.
- The lack of usability for images and videos: Currently, our model can generate 20 images per second which is less than the rate of some legitimate screenshot taking applications (up to 45 FPS). This rate of 20 FPS would be insufficient when screen-sharing a video. Moreover, the usability of our algorithms has only been tested on textual content.
- The insufficient security in sensitive contexts: With a sufficient number of screenshots and a dedicated algorithm, an adversary can reach their goal even if screenshots are altered using our approach. Even if taking more screenshots makes detection easier, in very sensitive contexts, more radical

measure (such as permanently using the setWindowsDisplayAffinity flag or disabling screenshots) should be more adapted.

10.3 Final remarks and future work

The work presented in this thesis can serve as a reference for future researchers aiming at countering the screenlogger threat. Our malicious and benign datasets are available on a public repository and examples of our altering algorithms can be found on the usability test website.

More generally, some of the conclusions we have drawn can be extended to other fields. For instance, our study has shown the need to pay a particular care to the functionalities triggered by malware samples while they are executed in a secure environment. Moreover, the performances of our detection system show that a tailored approach might, in some cases, avoid overfitting and make detection models more robust to evasion techniques. One last example is the usability results we obtained regarding retinal persistence, which can be of use to anyone interested in this property of the HVS.

In the future, it would be useful to collect more data on screenshot-taking spyware, for example through collaboration with an antivirus firm. A sufficient number of samples would allow the training of an effective neural network and lead to improved detection results.

Our prevention mechanism could also be optimised to be usable on non-textual images and allow for a better frame rate. Moreover, finding a way to prevent automatic screenshot merging would improve the security of the model.

Finally, as mentioned in the previous section, our screenshot altering approach might not give a 100% security guarantee in highly sensitive contexts. How-

ever, even in sensitive contexts, it might not be possible to completely prevent the screenshot functionality, particularly with the recent rise of telecommuting and screen sharing. Therefore, in the future, one solution might be to make the screen content impossible to exploit unless a specific device with a shared secret, such as programmable glasses or a screen overlay, is used to view the screenshots.

References

[1] Drozhzhin, "The greatest heist of the century: hackers stole $1 bln." [Online]. Available: https://www.kaspersky.com/blog/billion-dollar-apt-carbanak/7519/

[2] I. Jermyn, A. Mayer, F. Monrose, M. K. Reiter, and A. D. Rubin, "The design and analysis of graphical passwords," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, ser. SSYM'99. USA: USENIX Association, 1999, p. 1.

[3] A. Bianchi, I. Oakley, and D. S. Kwon, "The secure haptic keypad: A tactile password system," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 1089–1092.

[4] A. De Luca, M. Harbach, E. von Zezschwitz, M.-E. Maurer, B. E. Slawik, H. Hussmann, and M. Smith, "Now you see me, now you don't: Protecting smartphone authentication from shoulder surfers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 2937–2946.

[5] J. Lim, "Defeat spyware with anti-screen capture technology using visual persistence," in *Proceedings of the 3rd Symposium on Usable Privacy and Security*, ser. SOUPS '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 147–148.

[6] D. Nyang, A. Mohaisen, and J. Kang, "Keylogging-resistant visual authentication protocols," *IEEE Transactions on Mobile Computing*, vol. 13,

no. 11, pp. 2566–2579, 2014.

[7] H. Kim, H. Kim, and J. W. Yoon, "A new technique using a shuffling method to protect confidential documents from shoulder surfers," in *2015 1st International Conference on Software Security and Assurance (ICSSA)*, 2015, pp. 7–12.

[8] M. Mitchell, A.-I. A. Wang, and P. Reiher, "Cashtags: Protecting the input and display of sensitive data," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15. USA: USENIX Association, 2015, pp. 961–976.

[9] E. von Zezschwitz, S. Ebbinghaus, H. Hussmann, and A. De Luca, "You can't watch this! privacy-respectful photo browsing on smartphones," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 4320–4324.

[10] J.-U. Hou, D. Kim, H.-J. Song, and H.-K. Lee, "Secure image display through visual cryptography: Exploiting temporal responsibilities of the human eye," in *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, ser. IH&MMSec '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 169–174.

[11] G. Eric, "Defeat print screen with visual cryptography." [Online]. Available: https://www.delphitools.info/2012/08/01/defeat-print-screen-with-visual-cryptography/

[12] A. Y. Chia, U. Bandara, X. Wang, and H. Hirano, "Protecting against screenshots: An image processing approach," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1437–1445.

[13] S. Park and S.-U. Kang, "Visual quality optimization for privacy protection bar-based secure image display technique," *KSII Transactions on Internet and Information Systems*, vol. 11, pp. 3664–3677, 07 2017.

[14] Mitre, "Screen capture." [Online]. Available: https://attack.mitre.org/techniques/T1113/

[15] A. O. A. El-Mal, M. A. Sobh, and A. M. B. Eldin, "Hard-detours: A new technique for dynamic code analysis," in *Eurocon 2013*, 2013, pp. 46–51.

[16] R. Shahzad, S. Haider, and N. Lavesson, "Detection of spyware by mining executable files," 02 2010, pp. 295–302.

[17] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting apt malware infections based on malicious dns and traffic analysis," *Access, IEEE*, vol. 3, pp. 1132–1142, 01 2015.

[18] S. Bahtiyar, "Anatomy of targeted attacks with smart malware: Targeted attacks with smart malware," *Security and Communication Networks*, vol. 9, 02 2017.

[19] X. Ecular and G. Grey, "Cyberespionage campaign sphinx goes mobile with anubisspy." [Online]. Available: https://www.trendmicro.com/en_us/research/17/l/cyberespionage-campaign-sphinx-goes-mobile-anubisspy.html

[20] R. Shahzad, S. Haider, and N. Lavesson, "Detection of spyware by mining executable files," 02 2010, pp. 295–302.

[21] K. L. G. Research and A. Team, "The great bank robbery: Carbanak cybergang steals $1bn from 100 financial institutions worldwide." [Online]. Available: https://www.kaspersky.com/about/press-releases/

2015_the-great-bank-robbery-carbanak-cybergang-steals--1bn-from-100-financial-institutions-worldwide

[22] S. David, E. and P. Nicole, “Bank hackers steal millions via malware.” [Online]. Available: https://www.nytimes.com/2015/02/15/world/bank-hackers-steal-millions-via-malware.html

[23] S. S. Response, “Regin: Top-tier espionage tool enables stealthy surveillance.” [Online]. Available: https://www.databreaches.net/regin-top-tier-espionage-tool-enables-stealthy-surveillance/

[24] Z. Charline, “Viruses and malware: Research strikes back.” [Online]. Available: https://news.cnrs.fr/articles/viruses-and-malware-research-strikes-back

[25] U. Roman, “A new era in mobile banking trojans, securelist for kaspersky lab.” [Online]. Available: https://securelist.com/a-new-era-in-mobile-banking-trojans/79198/

[26] S. Lukas, “New telegram-abusing android rat discovered in the wild, welivesecurity by eset.” [Online]. Available: https://www.welivesecurity.com/2018/06/18/new-telegram-abusing-android-rat/

[27] G. Josh, L. Brandon, W. Kyle, and L. Pat, “Squirtdanger: The swiss army knife malware from veteran malware author thebottle.” [Online]. Available: https://unit42.paloaltonetworks.com/unit42-squirtdanger-swiss-army-knife-malware-veteran-malware-author-thebottle/

[28] B. Bogdan, “Six years and counting: Inside the complex zacinlo ad fraud operation, bitdefender.” [Online]. Available: https://labs.bitdefender.com/2018/06/six-years-and-counting-inside-the-complex-zacinlo-ad-fraud-operation/

[29] C. Mikey, "Xagent malware arrives on mac, steals passwords, screenshots, iphone backups." [Online]. Available: https://appleinsider.com/articles/17/02/14/xagent-malware-arrives-on-mac-steals-passwords-screenshots-iphone-backups

[30] O. Stefan, "The missing piece - sophisticated os x backdoor discovered, securelist by kaspersky lab." [Online]. Available: https://securelist.com/the-missing-piece-sophisticated-os-x-backdoor-discovered/75990/

[31] N. J. Cybersecurity and C. I. Cell, "Zbot/zeus." [Online]. Available: https://www.cyber.nj.gov/threat-center/threat-profiles/trojan-variants/zbot-zues

[32] T. N. Y. T. Tom Zeller Jr, "Cyberthieves silently copy your passwords as you type." [Online]. Available: https://www.nytimes.com/2006/02/27/technology/cyberthieves-silently-copy-your-passwords-as-you-type.html

[33] E. Pan, J. Ren, M. Lindorfer, C. Wilson, and D. Choffnes, "Panoptispy: Characterizing audio and video exfiltration from android applications," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, pp. 33–50, 10 2018.

[34] K. L. G. Research and A. Team, "The great bank robbery: Carbanak cybergang steals $1bn from 100 financial institutions worldwide." [Online]. Available: https://www.kaspersky.com/about/press-releases/2015_the-great-bank-robbery-carbanak-cybergang-steals--1bn-from-100-financial-institutions-worldwide

[35] Y. Fratantonio, C. Qian, S. Chung, and W. Lee, "Cloak and dagger: From two permissions to complete control of the ui feedback loop," 05 2017, pp. 1041–1057.

[36] A. Aviv, B. Sapp, M. Blaze, and J. Smith, “Practicality of accelerometer side channels on smartphones,” 12 2012, pp. 41–50.

[37] PandaSecurity, “Watch out for chrome and firefox web extensions that access browser history and rob passwords.” [Online]. Available: https://www.pandasecurity.com/en/mediacenter/malware/malicious-web-extensions/

[38] S. Heule, D. Rifkin, A. Russo, and D. Stefan, “The most dangerous code in the browser,” in *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, ser. HOTOS’15. USA: USENIX Association, 2015, p. 23.

[39] L. Bauer, S. Cai, L. Jia, T. Passaro, and Y. Tian, “Analyzing the dangers posed by chrome extensions,” 10 2014.

[40] I. Qabajeh, F. Thabtah, and F. Chiclana, “A recent review of conventional vs. automated cybersecurity anti-phishing techniques,” *Computer Science Review*, vol. 29, pp. 44–55, 08 2018.

[41] H. Sbai, M. Goldsmith, S. Meftali, and J. Happa, *A Survey of Keylogger and Screenlogger Attacks in the Banking Sector and Countermeasures to Them: 10th International Symposium, CSS 2018, Amalfi, Italy, October 29-31, 2018, Proceedings*, 01 2018, pp. 18–32.

[42] H. Sbaï, J. Happa, M. Goldsmith, and S. Meftali, “Dataset construction and analysis of screenshot malware,” *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 646–655, 2020.

[43] H. Sbai, J. Happa, and M. Goldsmith, *A Novel Behavioural Screenlogger Detection System*, 11 2021, pp. 279–295.

[44] C. Lin, H. Li, X. Zhou, and X. Wang, “Screenmilker: How to milk your android screen for secrets,” in *NDSS*, 2014.

[45] M. El-Serngawy and C. Talhi, “Captureme: Attacking the user credential in mobile banking applications,” *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 924–933, 2015.

[46] S. M. Muzammal, M. A. Shah, H. A. Khattak, S. Jabbar, G. Ahmed, S. Khalid, S. Hussain, and K. Han, “Counter measuring conceivable security threats on smart healthcare devices,” *IEEE Access*, vol. 6, pp. 20 722–20 733, 2018.

[47] S. Hwang, S. Lee, Y. Kim, and S. Ryu, “Bittersweet adb: Attacks and defenses,” in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS ’15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 579–584.

[48] C. Bacara, V. Lefils, J. Iguchi-Cartigny, G. Grimaud, and J. Wary, “Virtual keyboard logging counter-measures using human vision properties,” in *17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015*. IEEE, 2015, pp. 1230–1235.

[49] E. Nicolas, C. Bacara, J. Iguchi-Cartigny, G. Grimaud, and J. Wary, “Virtual keyboard logging counter-measures using common fate’s law,” in *Proceedings of the International Conference on Security and Management*, 2017, pp. 188–194.

[50] VirusShare, “A repository of malware samples.” [Online]. Available:

https://virusshare.com/

[51] AVCaesar, “A repository of malware samples.” [Online]. Available: https://avcaesar.malware.lu/

[52] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” 01 2018, pp. 108–116.

[53] “Skype.” [Online]. Available: https://www.skype.com/en/

[54] “Screenleap.” [Online]. Available: https://www.screenleap.com//

[55] “Join.me.” [Online]. Available: https://www.join.me/fr

[56] “Teamviewer.” [Online]. Available: https://www.teamviewer.com/fr/

[57] “Netwiewer.” [Online]. Available: http://www.tucows.com/preview/613992/Netviewer-Support

[58] “Gotomypc.” [Online]. Available: https://get.gotomypc.com/

[59] “Camtasia.” [Online]. Available: https://www.techsmith.fr/camtasia.html

[60] “Camstudio.” [Online]. Available: https://camstudio.org/

[61] “Ezvid.” [Online]. Available: https://www.ezvid.com/

[62] “Verity.” [Online]. Available: https://www.nchsoftware.com/childmonitoring/index.htm

[63] “Kidlogger.” [Online]. Available: https://kidlogger.net/

[64] “Norton online family 7.” [Online]. Available: https://family.norton.com/web/

[65] “Picpick.” [Online]. Available: https://picpick.app/fr/

[66] “Snipping tool.” [Online]. Available: https://support.microsoft.com/en-us/help/13776/windows-10-use-snipping-tool-to-capture-screenshots

[67] “Faststone capture.” [Online]. Available: https://www.faststone.org/FSCaptureDetail.htm

[68] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, “A survey on heuristic malware detection techniques,” in *The 5th Conference on Information and Knowledge Technology*, 2013, pp. 113–120.

[69] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010, pp. 297–300.

[70] A. Tang, S. Sethumadhavan, and S. J. Stolfo, “Unsupervised anomaly-based malware detection using hardware features,” in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Cham: Springer International Publishing, 2014, pp. 109–129.

[71] A. Souri and R. Hosseini, “A state-of-the-art survey of malware detection approaches using data mining techniques,” vol. 8, pp. 1–22, 12 2018.

[72] M. Eskandari and S. Hashemi, “A graph mining approach for detecting unknown malwares,” *Journal of Visual Languages & Computing*, vol. 23, pp. 154–162, 06 2012.

[73] I. Obaidat, M. Sridhar, K. M. Pham, and P. H. Phung, “Jadeite: A novel image-behavior-based approach for java malware detection using deep learning,” *Computers and Security*, vol. 113, p. 102547, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003710

[74] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. Bringas, "Using opcode sequences in single-class learning to detect unknown malware," *IET Information Security*, vol. 5, pp. 220–227, 12 2011.

[75] M. Nar, A. Kakisim, N. Carkaci, M. Yavuz, and I. Sogukpinar, "Analysis and comparison of opcode-based malware detection approaches," 09 2018, pp. 498–503.

[76] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 12 2015.

[77] W. Han, J. XUE, Y. Wang, L. Huang, Z. Kong, and L. Mao, "Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *Computers & Security*, vol. 83, 02 2019.

[78] D. Al-Anezi, "Generic packing detection using several complexity analysis for accurate malware detection," *International Journal of Advanced Computer Science and Applications*, vol. 5, 01 2014.

[79] C. Sandbox, "Automated malware analysis." [Online]. Available: https://cuckoosandbox.org

[80] CWSandbox, "Cwsandbox." [Online]. Available: https://cwsandbox.org

[81] S. Megira, A. Pangesti, and F. Wibowo, "Malware analysis and detection using reverse engineering technique," *Journal of Physics: Conference Series*, vol. 1140, p. 012042, 12 2018.

[82] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "Malinsight: A systematic profiling based malware detection framework," *Journal of Network and Computer Applications*, vol. 125, 11 2018.

[83] D. Mohaisen, O. Alrawi, and M. Mohaisen, "Amal: High-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, 04 2015.

[84] E. Amer, I. Zelinka, and S. El-Sappagh, "A multi-perspective malware detection approach through behavioral fusion of api call sequence," *Computers and Security*, vol. 110, p. 102449, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740482100273X

[85] VirusShare, "Virusshare." [Online]. Available: https://virusshare.com/

[86] Y. Ye, T. Li, K. Huang, Q. Jiang, and Y. Chen, "Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list," *Journal of Intelligent Information Systems*, vol. 35, pp. 1–20, 2009.

[87] Y. Ding, X. Yuan, K. Tang, X. Xiao, and Y. Zhang, "Malware detection based on objective-oriented association mining," *Computers & Security*, vol. 39, pp. 315–324, 11 2013.

[88] C. Jing, Y. Wu, and C. Cui, "Ensemble dynamic behavior detection method for adversarial malware," *Future Generation Computer Systems*, vol. 130, pp. 193–206, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004945

[89] D. Javaheri, M. Hosseinzadeh, and A. Rahmani, "Detection and elimination of spyware and ransomware by intercepting kernel-level system routines," *IEEE Access*, vol. PP, pp. 1–1, 12 2018.

[90] B. AsSadhan, A. Bashaiwth, J. Al-Muhtadi, and S. Alshebeili, "Analysis of p2p, irc and http traffic for botnets detection," *Peer-to-Peer Networking and Applications*, 09 2018.

[91] B. AsSadhan, A. Bashaiwth, J. Al-Muhtadi, and S. AlShebeili, "Analysis of p2p, irc and http traffic for botnets detection," *Peer-to-Peer Networking and Applications*, vol. 11, pp. 848–861, 2018.

[92] A. Boukhtouta, S. Mokhov, N.-E. Lakhdari, M. Debbabi, and J. Paquet, "Network malware classification comparison using dpi and flow packet headers," *Journal of Computer Virology and Hacking Techniques*, vol. 11, pp. 1–32, 07 2015.

[93] A. H. Lashkari, A. F. A.Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017, pp. 233–23 309.

[94] S. Nari and A. Ghorbani, "Automated malware classification based on network behavior," 01 2013, pp. 642–647.

[95] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–7.

[96] UNB, "Intrusion detection evaluation dataset (cic-ids2017)." [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[97] ——, "Cse-cic-ids2018 on aws." [Online]. Available: https://www.unb.ca/cic/datasets/ids-2018.html

[98] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting p2p botnets through network behavior analysis and machine learning," 07 2011.

[99] E. Beigi, H. Jazi, N. Stakhanova, and A. Ghorbani, “Towards effective feature selection in machine learning-based botnet detection approaches,” *2014 IEEE Conference on Communications and Network Security, CNS 2014*, pp. 247–255, 12 2014.

[100] W. Mao, Z. Cai, D. Towsley, Q. Feng, and X. Guan, “Security importance assessment for system objects and malware detection,” *Computers & Security*, vol. 68, 03 2017.

[101] S. Banin, A. Shalaginov, and K. Franke, “Memory access patterns for malware detection,” 2016.

[102] G. Dini, F. Martinelli, A. Saracino, and a. Sgandurra, “Madam: a multi-level anomaly detector for android malware,” vol. 7531, 10 2012.

[103] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “"andromaly": A behavioral malware detection framework for android devices,” *J. Intell. Inf. Syst.*, vol. 38, pp. 161–190, 02 2012.

[104] A. Ferrante, E. Medvet, F. Mercaldo, J. Milosevic, and C. A. Visaggio, “Spotting the malicious moment: Characterizing malware behavior using dynamic features,” 08 2016, pp. 372–381.

[105] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, and D. Ucci, “Android malware family classification based on resource consumption over time,” 10 2017, pp. 31–38.

[106] J. Milosevic, M. Malek, and A. Ferrante, “A friend or a foe? detecting malware using memory and cpu features,” ser. ICETE 2016. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2016, pp. 73–84.

[107] Yara, "The pattern matching swiss knife for malware." [Online]. Available: https://virustotal.github.io/yara/

[108] T. Liu, X. Guan, Y. Qu, and Y. Sun, "A layered classification for malicious function identification and malware detection," *Concurrency and Computation: Practice & Experience*, vol. 24, pp. 1169–1179, 08 2012.

[109] A. Hellal and L. Romdhane, "Minimal contrast frequent pattern mining for malware detection," *Computers & Security*, vol. 62, 06 2016.

[110] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," 08 2017, pp. 62–79.

[111] D. Arp, M. Spreitzenbarth, M. HÃ¼bner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," 02 2014.

[112] U. Urooj, B. A. S. Al-rimy, A. Zainal, F. A. Ghaleb, and M. A. Rassam, "Ransomware detection using the dynamic analysis and machine learning: A survey and research directions," *Applied Sciences*, vol. 12, no. 1, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/1/172

[113] J. C. Kimmel, A. D. Mcdole, M. Abdelsalam, M. Gupta, and R. Sandhu, "Recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure," *IEEE Access*, vol. 9, pp. 68 066–68 080, 2021.

[114] M. Agarwal, M. Mehra, R. Pawar, and D. Shah, "Secure authentication using dynamic virtual keyboard layout," in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ser. ICWET '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 288–291.

[115] A. Parekh, A. Pawar, P. Munot, and P. Mantri, "Secure authentication using anti-screenshot virtual keyboard," *International Journal of Computer Science Issues*, vol. 8, 09 2011.

[116] D. S. Tan, P. Keyani, and M. Czerwinski, "Spy-resistant keyboard: More secure password entry on public touch screen displays," in *Proceedings of the 17th Australia Conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future*, ser. OZCHI '05. Narrabundah, AUS: Computer-Human Interaction Special Interest Group (CHISIG) of Australia, 2005, pp. 1–10.

[117] D. Mehdi and J. Mohammad, "Secure payment in e-commerce: Deal with keyloggers and phishings," in *International Journal of Electronics Communication and Computer Engineering*, 2014, pp. 656–660.

[118] R. Srinivasan, K. Maheswari, R. Hemapriya, and S. Sriharilakshmi, "Shoulder surfing resistant virtual keyboard for internet banking," *World Appl. Sci. J*, vol. 31, 04 2014.

[119] V. Roth and K. Richter, "How to fend off shoulder surfing," *Journal of Banking & Finance*, vol. 30, pp. 1727–1751, 02 2006.

[120] D. Suraj, G. V, and D. H, "Secure authentication using session based password with virtual keyboard," in *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 4, 2016.

[121] H. Zhao and X. Li, "S3pas: A scalable shoulder-surfing resistant textual-graphical password authentication scheme," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, vol. 2, 2007, pp. 467–472.

[122] V. Roth, K. Richter, and R. Freidinger, "A pin-entry method resilient against shoulder surfing," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, ser. CCS '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 236–245.

[123] P. Nand, P. K. Singh, J. Aneja, and Y. Dhingra, "Prevention of shoulder surfing attack using randomized square matrix virtual keyboard," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 916–920.

[124] A. Surjushe, H. N. Sheikh, K. Bajar, P. Dhage, and P. P. Sambare, "Anti-screenshot , shoulder surfing resistant virtual keypad for online platform," vol. 3, 2017, pp. 491–496.

[125] S. Wiedenbeck, J. Waters, J.-C. Birget, A. Brodskiy, and N. Memon, "Pass-points: Design and longitudinal evaluation of a graphical password system," *Int. J. Hum.-Comput. Stud.*, vol. 63, no. 1-2, pp. 102–127, Jul. 2005.

[126] H. Sun, S. Chen, J. Yeh, and C. Cheng, "A shoulder surfing resistant graphical authentication system," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 2, pp. 180–193, 2018.

[127] D. Kim, P. Dunphy, P. Briggs, J. Hook, J. W. Nicholson, J. Nicholson, and P. Olivier, "Multi-touch authentication on tabletops," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 1093–1102.

[128] M. Martinez-Diaz, J. Fierrez, and J. Galbally, "Graphical password-based user authentication with free-form doodles," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 4, pp. 607–614, 2016.

[129] A. Bianchi, I. Oakley, V. Kostakos, and D. S. Kwon, "The phone lock: Audio and haptic shoulder-surfing resistant pin entry methods for mobile devices," in *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, ser. TEI '11. New York, NY, USA: Association for Computing Machinery, 2010, pp. 197–200.

[130] S. E. Palmer, "Modern theories of gestalt perception," *Mind and Language*, vol. 5, no. 4, pp. 289–323, 1990.

[131] F. Brudy, D. Ledo, and S. Greenberg, "Is anyone looking? mediating shoulder surfing on public displays (the video)," in *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 159–160.

[132] M. Khamis, M. Eiband, M. ZÃ¼rn, and H. Hussmann, "Eyespot: Leveraging gaze to protect private text content on mobile devices from shoulder surfing," *Multimodal Technologies and Interaction*, vol. 2, p. 45, 08 2018.

[133] M. Eiband, E. von Zezschwitz, D. Buschek, and H. Hußmann, "My scrawl hides it all: Protecting text messages against shoulder surfing with handwritten fonts," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 2041–2048.

[134] E. Bursztein, A. Moscicki, C. Fabry, S. Bethard, J. C. Mitchell, and D. Jurafsky, "Easy does it: More usable captchas," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 2637–2646.

[135] N. Roshanbin and J. Miller, “Adamas,” *Future Gener. Comput. Syst.*, vol. 55, no. C, pp. 289–310, Feb. 2016.

[136] C. Tangmanee, “Effects of text rotation, string length, and letter format on text-based captcha robustness,” *Journal of Applied Security Research*, vol. 11, pp. 349–361, 07 2016.

[137] E. Bursztein, M. Martin, and J. Mitchell, “Text-based captcha strengths and weaknesses,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS ’11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 125–138.

[138] E. Bursztein, J. Aigrain, A. Moscicki, and J. Mitchell, “The end is nigh: Generic solving of text-based captchas,” in *WOOT*, 2014.

[139] S. Geman and D. Geman, “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[140] S. team, “Spyshelter: Anti-keylogger software.” [Online]. Available: https://www.spyshelter.com/

[141] Raymond, “What is the best anti keylogger and anti screen capture software?” [Online]. Available: https://www.raymond.cc/blog/what-is-the-best-anti-keylogger-and-anti-screen-capture-software/2/

[142] W. report, “3 best anti-screenshot software for windows 10.” [Online]. Available: https://windowsreport.com/anti-screen-capture-software/

[143] W. Security, “2018 user risk report: Results of an international cybersecurity awareness survey.” [Online]. Available: https://info.wombatsecurity.com/hubfs/WombatProofpoint-UserRiskSurveyReport2018_US.pdf

[144] Microsoft, "Microsoft documentation, setwindowdisplayaffinity function." [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowdisplayaffinity

[145] S. C. Sarin Sumit Manmohan, "U.s. patent 8,844,059 b1." [Online]. Available: https://patents.justia.com/inventor/sumit-sarin

[146] S. Zeki, J. Watson, C. Lueck, K. Friston, C. Kennard, and R. Frackowiak, "A direct demonstration of functional specialization in human visual cortex," *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 11, pp. 641–9, 04 1991.

[147] H. Quigley, A. Brown, J. Morrison, and S. Drance, "The size and shape of the optic disc in normal human eyes." *Archives of ophthalmology*, vol. 108 1, pp. 51–7, 1990.

[148] D. Weintraub, "Rectangle discriminability: perceptual relativity and the law of prägnanz." *Journal of experimental psychology*, vol. 88 1, pp. 1–11, 1971.

[149] I. Phillips, "Afterimages and sensation," *Philosophy and Phenomenological Research*, vol. 87, pp. 417–453, 2013.

[150] H. Hawkins and G. Shulman, "Two definitions of persistence in visual perception," *Perception & Psychophysics*, vol. 25, pp. 348–350, 1979.

[151] A. Briand, S. Zacharie, L. Jean-Louis, and M.-J. Meurs, *Identification of Sensitive Content in Data Repositories to Support Personal Information Protection*, 01 2018, pp. 898–910.

[152] S. Lehar, "The world in your head : A gestalt view of the mechanism of conscious experience," 2003.

[153] A. Desolneux, L. Moisan, and J. Morel, "Gestalt theory and computer vision," 2004, pp. 71–101.

[154] Nightwatchcybersecurity, "Research: Securing android applications from screen capture." [Online]. Available: https://wwws.nightwatchcybersecurity.com/2016/04/13/research-securing-android-applications-from-screen-capture/

[155] Cwac-security, "About the flag secure child window issue." [Online]. Available: https://github.com/commonsguy/cwac-security/blob/master/docs/FLAGSECURE.md

[156] M. L. S. Advisor, "Screen capture via ui overlay in mediaprojection." [Online]. Available: https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-android-MediaProjection-tapjacking-advisory-2017-11-14.pdf

[157] A. Developers, "Android debug bridge." [Online]. Available: http://developer.android.com/intl/zh-CN/tools/help/adb.html

[158] Gdata, "Critical vulnerability: first android worm discovered." [Online]. Available: https://www.gdatasoftware.com/news/2018/06/30855-critical-vulnerability-first-android-worm-discovered

[159] T. Security, "Variant of satori/mirai detected attacking public available adb shells." [Online]. Available: https://telekomsecurity.github.io/2018/07/adb-botnet.html

[160] G. Cirlig, "Trinity - p2p malware over adb." [Online]. Available: https://www.ixiacom.com/company/blog/trinity-p2p-malware-over-adb

[161] S. Heron, "The rise and rise of the keyloggers," *Network Security*, vol. 2007, pp. 4–6, 06 2007.

[162] M. Vuagnoux and S. Pasini, "Compromising electromagnetic emanations of wired and wireless keyboards," *USENIX Security Symposium*, 01 2009.

[163] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," vol. 2004, 06 2004, pp. 3 – 11.

[164] M. Hussain, A. Al-Haiqi, A. Zaidan, B. Bahaa, M. L. Mat Kiah, N. Anuar, and M. Abdulnbi, "The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks," *Pervasive and Mobile Computing*, vol. 25, pp. 1–25, 01 2016.

[165] Shadow, "Shadow - kid's key logger app." [Online]. Available: https://play.google.com/store/apps/details?id=simpllekeyboard.main&hl=en

[166] R. Vedprakash, "Security issues with android accessibility." [Online]. Available: https://android.jlelse.eu/android-accessibility-75fdc5810025

[167] O. Wiese and V. Roth, "See you next time: A model for modern shoulder surfers," in *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 453–464.

[168] S. Son and V. Shmatikov, "The hitchhiker's guide to dns cache poisoning," vol. 50, 09 2010, pp. 466–483.

[169] X. corp., "Xarp - advanced arp spoofing detection." [Online]. Available: http://www.xarp.net/

[170] Android, "Android developers documentation, permissions overview." [Online]. Available: https://developer.android.com/guide/topics/permissions/overview

[171] S. Rod, “Malicious browser extension campaigns.” [Online]. Available: https://android.jlelse.eu/android-accessibility-75fdc5810025

[172] E. I. Security, “Cheater alert odlanor spyware destroys pokerstars and full tilt poker users’ odds of winning.” [Online]. Available: https://www.eset.com/int/about/newsroom/press-releases/research/cheater-alert-odlanor-spyware-destroys-pokerstars-and-full-tilt-poker-users-odds-of-winning/

[173] M. documentation, “Screen class.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.screen?view=netframework-4.7.2

[174] J. Grossman, “Top 10 web hacking techniques: "what’s possible, not probable",” *login Usenix Mag.*, vol. 34, no. 6, 2009. [Online]. Available: https://www.usenix.org/publications/login/december-2009-volume-34-number-6/top-10-web-hacking-techniques-whats-possible-not

[175] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schechter, and C. Jackson, “Clickjacking: Attacks and defenses,” in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security’12. USA: USENIX Association, 2012, p. 22.

[176] C. Rich, W. Jason, M. Neel, B. Ken, C. Wentao, and R. Megan, “An investigation of chrysaor malware on android.” [Online]. Available: https://android-developers.googleblog.com/2017/04/an-investigation-of-chrysaor-malware-on.html

[177] S. Jung and Y. Won, “Ransomware detection method based on context-aware entropy analysis,” vol. 22, no. 20, pp. 6731–6740, Oct. 2018.

[178] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148*, ser. DIMVA 2015. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 3–24. [Online]. Available: https://doi.org/10.1007/978-3-319-20550-2_1

[179] C. Cimpanu, "Necurs malware will now take a screenshot of your screen, report runtime errors." [Online]. Available: https://www.bleepingcomputer.com/news/security/necurs-malware-will-now-take-a-screenshot-of-your-screen-report-runtime-errors/

[180] S. S. Response, "Necurs attackers now want to see your desktop." [Online]. Available: https://www.symantec.com/connect/blogs/necurs-attackers-now-want-see-your-desktop

[181] S. Granger, "Social engineering fundamentals, part i: Hacker tactics," 2003. [Online]. Available: http://www.123seminarsonly.com/Seminar-Reports/021/6921587-Social-Engineering-Fundamentals-Part-1.pdf

[182] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Advanced social engineering attacks," *Journal of Information Security and Applications*, vol. 22, 10 2014.

[183] X. Li, J. Smith, T. Dinh, and M. Thai, "Privacy issues in light of reconnaissance attacks with incomplete information," 10 2016, pp. 311–318.

[184] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "The socialbot network: When bots socialize for fame and money," 12 2011, pp. 93–102.

[185] B. Parmar, "Protecting against spear-phishing," *Computer Fraud & Security*, vol. 2012, pp. 8–11, 01 2012.

[186] M. Documentation, "Enumwindows function." [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-enumwindows

[187] ——, "Findwindowa function." [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-findwindowa

[188] S. Holtmanns and I. Oliver, "Sms and one-time-password interception in lte networks," 05 2017, pp. 1–6.

[189] S. I. Harri Sylvander, "Xtremerat - when unicode breaks." [Online]. Available: https://www.sans.org/reading-room/whitepapers/malicious/paper/35897

[190] M. Jane, "What is shoulder surfing?" [Online]. Available: https://www.itpro.co.uk/security/31723/what-is-shoulder-surfing

[191] W. Zack, "Many popular iphone apps secretly record your screen without asking." [Online]. Available: https://techcrunch.com/2019/02/06/iphone-session-replay-screenshots/?guccounter=1

[192] Glassbox, "Glassbox." [Online]. Available: https://glassboxdigital.com/

[193] L. H. Philippe, W. Lauren, and R. Jonathan, "What is the document object model?" [Online]. Available: https://www.w3.org/TR/DOM-Level-2-Core/introduction.html

[194] M. documentation, "Capturing an image." [Online]. Available: https://docs.microsoft.com/fr-fr/windows/desktop/gdi/capturing-an-image

[195] Y. B Tao, J. Xingyu, Q. Bo, and H. Zhanglin, "New wine in old bottle: New azorult variant found in findmy-

name campaign using fallout exploit kit." [Online]. Available: https://unit42.paloaltonetworks.com/unit42-new-wine-old-bottle-new-azorult-variant-found-findmyname-campaign-using-fallout-exploit-kit/

[196] A. Blaich, A. Kumar, J. Richards, M. Flossman, C. Quintin, E. Galperin, L. (Firm), and E. F. Foundation, *Dark Caracal: Cyber-espionage at a Global Scale*. Lookout, 2018. [Online]. Available: https://books.google.fr/books?id=WP6XtgEACAAJ

[197] F. Matthieu and B. Jean-Ian, "Read the manual: A guide to the rtm banking trojan." [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2017/02/Read-The-Manual.pdf

[198] L. Denis, "Chafer used remexi malware to spy on iran-based foreign diplomatic entities." [Online]. Available: https://securelist.com/chafer-used-remexi-malware/89538/

[199] Mandiant, "Apt1: Exposing one of china's cyber espionage units." [Online]. Available: https://www.nationalcyberwatch.org/resource/apt1-exposing-one-of-chinas-cyber-espionage-units-2/

[200] PowerShellMafia, "Powersploit - a powershell post-exploitation framework." [Online]. Available: https://www.darknet.org.uk/2015/12/powersploit-powershell-post-exploitation-framework/

[201] "Teramind." [Online]. Available: https://www.teramind.co/

[202] C. Anton, "Operation groundbait: Analysis of a surveillance toolkit." [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2016/05/Operation-Groundbait.pdf

[203] Catchamas, "Catchamas." [Online]. Available: https://attack.blackbot.io/software/S0261/

[204] G. Josh and M.-O. Jen, “T9000: Advanced modular backdoor uses anti-analysis techniques.” [Online]. Available: https://www.netsciences.com/2016/02/10/t9000-advanced-modular-backdoor-uses-anti-analysis-techniques/

[205] A. Gostev, “The flame: Questions and answers.” [Online]. Available: https://securelist.com/the-flame-questions-and-answers/34344/

[206] F. Robert and L. Bryan, “Sofacy continues global attacks and wheels out new ‘cannon’ trojan.” [Online]. Available: https://unit42.paloaltonetworks.com/unit42-sofacy-continues-global-attacks-wheels-new-cannon-trojan/

[207] I. Ionut, “Russian hackers hide zebrocy malware in virtual disk images.” [Online]. Available: https://www.bleepingcomputer.com/news/security/russian-hackers-hide-zebrocy-malware-in-virtual-disk-images/

[208] H. Zuzana, “Invisimole: Surprisingly equipped spyware, undercover since 2013.” [Online]. Available: https://www.welivesecurity.com/2018/06/07/invisimole-equipped-spyware-undercover/

[209] E. Luca, “Analysis results of zeus.variant.panda.” [Online]. Available: https://cyberwtf.files.wordpress.com/2017/07/panda-whitepaper.pdf

[210] S. Manish, C. Vincent, F. Nalani, L. Yogesh, R. Nick, and O. Jacqueline, “New targeted attack in the middle east by apt34, a suspected iranian threat group, using cve-2017-11882 exploit.” [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/12/targeted-attack-in-middle-east-by-apt34.html

[211] L. Brandon, G. Josh, and B. Brittany, “Patchwork continues to deliver badnews to the indian subcontinent.” [Online]. Avail-

able: https://unit42.paloaltonetworks.com/unit42-patchwork-continues-deliver-badnews-indian-subcontinent/

[212] G. Nicholas, "Monsoon - analysis of an apt campaign." [Online]. Available: https://www.forcepoint.com/blog/x-labs/monsoon-analysis-apt-campaign

[213] T. Yair, "Micropsia malware." [Online]. Available: https://securityboulevard.com/2018/07/micropsia-malware/

[214] S. Heron, "Advanced encryption standard (aes)," *Network Security*, vol. 2009, no. 12, pp. 8–12, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1353485810700064

[215] L. Bryan and F. Robert, "Magic hound campaign attacks saudi targets." [Online]. Available: https://unit42.paloaltonetworks.com/unit42-magic-hound-campaign-attacks-saudi-targets/

[216] R. Vicky and H. Kaoru, "New malware 'rover' targets indian ambassador to afghanistan." [Online]. Available: https://unit42.paloaltonetworks.com/new-malware-rover-targets-indian-ambassador-to-afghanistan/

[217] K. Anthony and R. Dominik, "The gamaredon group toolset evolution." [Online]. Available: https://unit42.paloaltonetworks.com/unit-42-title-gamaredon-group-toolset-evolution/

[218] T. M. T. Project, "Malshare." [Online]. Available: https://malshare.com/

[219] V. Inc., "Virussign." [Online]. Available: https://www.virussign.com/

[220] Inetsim, "Inetsim: Internet services simulation suite." [Online]. Available: https://www.inetsim.org/

[221] Wireshark.org, "Wireshark." [Online]. Available: https://www.wireshark.org/

[222] Rohitab, "Api monitor." [Online]. Available: http://www.rohitab.com/apimonitor

[223] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu, "Peer to peer botnet detection based on flow intervals," vol. 376, 06 2012.

[224] GReAT, "Silence - a new trojan attacking financial organizations." [Online]. Available: https://securelist.com/the-silence/83009/

[225] B. Parys, "The key boys are back in town." [Online]. Available: https://vx-underground.org/archive/APTs/2017/2017.11.02(2)/Keyboys.pdf

[226] G. Michael, "Agent tesla: A day in a life of ir." [Online]. Available: https://securityboulevard.com/2020/11/agent-tesla-a-day-in-a-life-of-ir/

[227] S.-R. John, A. R. Bahr, A. Hulcoop, B. Matt, and K. Katie, "Group5: Syria and the iranian connection." [Online]. Available: https://citizenlab.ca/2016/08/group5-syria/

[228] C. R. Team, "Operation wilted tulip - exposing a cyber espionage apparatus." [Online]. Available: https://www.clearskysec.com/tulip/

[229] L. Daniel, H. Jaromir, and P. Cedric, "Untangling the patchwork cyberespionage group." [Online]. Available: https://www.trendmicro.com/en_us/research/17/l/untangling-the-patchwork-cyberespionage-group.html

[230] Sbai, "Screeminals." [Online]. Available: https://github.com/sbhugo/Screeminals

[231] "Show my pc." [Online]. Available: https://showmypc.com/

[232] "Mingleview." [Online]. Available: https://www.mingleview.com/

[233] XpressTex, "How remote computer repairs can help you!" [Online]. Available: https://www.xpresstex.com.au/why-remote-computer-repairs/

[234] "Anydesk." [Online]. Available: https://anydesk.com/fr

[235] "Apple remote desktop." [Online]. Available: https://support.apple.com/remote-desktop

[236] "Chrome remote desktop." [Online]. Available: https://remotedesktop.google.com

[237] "Tinytake." [Online]. Available: https://tinytake.com/

[238] "Telestream screenflow." [Online]. Available: http://www.telestream.net/screenflow/overview.htm

[239] M. Polly and M. Anders, "Employers deploy spy software to monitor at-home workers." [Online]. Available: https://www.insurancejournal.com/news/national/2020/03/27/562594.htm

[240] "Time doctor." [Online]. Available: https://www.timedoctor.com/

[241] "Activtrak." [Online]. Available: https://activtrak.com/

[242] "Interguard." [Online]. Available: https://www.interguardsoftware.com/

[243] "Hubstaff." [Online]. Available: https://hubstaff.com/

[244] "Staffcop enterprise." [Online]. Available: https://www.staffcop.com/news/ent_official_release.php

[245] B. Albert, "Weka 3: Machine learning software in java." [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[246] O. Argus, "Argus." [Online]. Available: https://openargus.org

[247] E. Beigi, H. Jazi, N. Stakhanova, and A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches,"

2014 IEEE Conference on Communications and Network Security, CNS 2014, pp. 247–255, 12 2014.

[248] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[249] B. Jason, "Recursive feature elimination (rfe) for feature selection in python." [Online]. Available: https://machinelearningmastery.com/rfe-feature-selection-in-python/

[250] T. Reed, "New signed malware called janicab." [Online]. Available: https://www.thesafemac.com/new-signed-malware-called-janicab/

[251] Microsoft, "Detours." [Online]. Available: https://www.microsoft.com/en-us/research/project/detours/

[252] T. E. of Encyclopaedia Britannica, "Afterimage psychology." [Online]. Available: https://www.britannica.com/science/afterimage

[253] F. R, P. S, W. A, and W. E, "Gaussian smoothing." [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[254] Sbai, "Persistest (http://persistest.herokuapp.com/)." [Online]. Available: http://persistest.herokuapp.com/

[255] S. Hugo, "Persistest informed consent." [Online]. Available: https://app.evalandgo.com/s/index.php?a=JTk2ciU5MmklOTglQTk=&id=JTk4aSU5MWglOUQlQUM=