ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2017 ENST 0065

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique »

présentée et soutenue publiquement par

Onur CATAKOGLU

le 30 Novembre 2017

Using Web Honeypots to Study the Attackers Behavior

Directeur de thèse : **Davide BALZAROTTI**

Jury

William ROBERTSON, Northeastern University	Rapporteur
Magnus ALMGREN, Chalmers University of Technology	Rapporteur
Yves ROUDIER, University of Nice Sophia Antipolis	Examinateur
Albert LEVI, Sabanci University	Examinateur
Leyla BILGE, Symantec Research Labs	Examinateur

TELECOM ParisTech
École de l'Institut Télécom - membre de ParisTech

T H È S E

Using Web Honeypots to Study the Attackers Behavior

Thesis

Onur Catakoglu
Onur.Catakoglu@eurecom.fr

École Doctorale Informatique, Télécommunication et Électronique, Paris
ED 130

November 30, 2017

Advisor:

Prof. Dr. Davide Balzarotti
EURECOM, Sophia-Antipolis

Reviewers:

Prof. Dr. William ROBERTSON,
Northeastern University

Prof. Dr. Magnus ALMGREN,
Chalmers University of Technology

Examiners:

Prof. Dr. Yves ROUDIER,
University of Nice Sophia Antipolis

Prof. Dr. Albert LEVI,
Sabanci University

Dr. Leyla BILGE,
Symantec Research Labs

Acknowledgements

First and foremost I would like to express my gratitude to my supervisor, Davide Balzarotti. He was always welcoming whenever I need to ask questions, discuss any ideas and even when he was losing in our long table tennis matches. I am very thankful for his guidance throughout my PhD and I will always keep the mental image of him staring at me for various reasons as a motivation to move on when things will get tough in the future. I would also like to thank my reviewers for their constructive comments regarding this thesis.

Of course, I owe my friends and colleagues from the S3 group and Eurecom a big thank you for making it so much fun for me. All the coffee breaks, night-outs, tennis matches, barbecues, CTFs and games would not be enjoyable without them: Jonas, Marius, Melek, Leyla, Aurelien, Xavier, Emanuele, Fabio, Dario, Tom, Giovanni, Sebastian, Samuele, Iskander and Antonio.

I would also like to thank you all the Eurecom's administration staff and especially to Audrey Ratier for taking care of all the bureaucratic problems and making it easier for me. Your patience was always appreciated by me and I owe you my gratitudes.

When things were looking a bit down or when I needed some encouragement, I listened some of the greatest artists to find my strength. They helped me before the PhD, and they helped again during my time in Eurecom. Thus, I would like to thank Metallica, Megadeth, Iron Maiden, Children of Bodom, Avenged Sevenfold, Harun Kolcak and many more that gave me courage to continue.

Last but definitely not least, I cannot thank enough to my beautiful wife, Merve, and my parents and my sister for all the support they have been given to me. They were there when things were not looking the brightest but they found a way to cheer me up, ease my burden to make all this possible. I am very grateful for having such inspiring people as family in my life.

Acknowledgements

Abstract

Despite a continous effort from the security community, attacks against web applications are still one of the most common forms of compromise. The increasing rate of vulnerable websites continues to be a concern not only to the site owners, but also to the rest of the Internet. While the phenomenon of web attacks has been thoroughly studied from a client-side perspective, only a few work focused on the nature of this phenomenon. Hence, our understanding of attacks against web applications still relies on rudimentary tools and tedious manual analysis efforts.

In this thesis, we adopt a server-side approach to study the attackers behavior on the Web, by analyzing the information collected by using a high-interaction honeypot.

In the first part of the thesis, we propose for the first time an automated technique to extract and validate Indicators of Compromise, which are forensic artifacts used as signs that a system has been compromised, for web applications. Our experiments show that our solution is able to automatically generate web indicators of compromise that have been previously used by attackers for long periods of time without being detected by other conventional approaches.

In the second part, we explore the attack landscape in the secluded parts of the Internet, known as Dark Web, which are operated by decentralized and anonymous-preserving protocols like Tor. We deployed a high interaction honeypot in the Tor network for a period of seven months to conduct a measurement study of the type of attacks and of the attackers behavior that affect this still relatively unknown corner of the Web.

In the last part of the thesis, we shift our focus to server-side malicious code and we introduce the first fully-automated PHP code analysis sandbox. Our system consists of an instrumented PHP interpreter and a replay component that automatically extracts the required HTTP requests from the target web server log and use them to stimulate the server-side code mimicking exactly the action of the attacker. We validated our system using a large dataset of over 8.000 real attack sessions, and we discuss our findings and distill some key insights on the behavior of web attacks.

Résumé

Malgré les efforts de la communauté de la sécurité, les attaques contre les applications Web sont encore l'une des formes les plus courantes de compromission. Le taux de sites Web vulnérables continue d'être une préoccupation non seulement pour les propriétaires des sites, mais aussi pour les autres. Même si le phénomène des attaques web a été étudié à fond du côté client, seuls quelques travaux ont exploré la nature de ce phénomène. Par conséquence, notre compréhension des attaques contre les applications Web repose toujours sur des outils rudimentaires et des efforts d'analyse manuelle fastidieux. Dans cette

thèse, nous adoptons une approche côté serveur pour étudier le comportement des attaquants sur Internet en analysant les informations recueillies par un pot de miel à forte interaction. Dans la première partie de la thèse, nous proposons

pour la première fois une technique automatisée pour extraire et valider les indicateurs de compromis, qui sont des artefacts utilisés comme des signes qu'un système a été compromis, pour des applications Web. Nos expériences montrent que notre solution peut produire automatiquement les indicateurs de compromis, dans des cas où les attaquants n'avaient pas été détecté par d'autres approches conventionnelles. Dans la deuxième partie, nous explorons les menaces dans les

parties isolées d'Internet, connu comme toile profonde, basé sur des protocoles décentralisés et anonyme comme TOR. Nous avons déployé un pot de miel a haute interaction dans le réseau de TOR pour une période de sept mois, pour mener une étude sur les types d'attaque et sur le comportement des attaquants, dans le coin toujours relativement inconnu du Web. Dans la dernière partie de

la thèse, nous passons à l'analyse du mali ciel côté serveur et nous présentons le premier environnement de test entièrement automatisé pour PHP. Notre système consiste d'un interpréteur PHP instrumenté et d'un composant pour rejouer les attaques, qui extrait automatiquement les requêtes HTTP depuis le journal du serveur Web cible, et les utilise pour stimuler le code côté serveur, imitant exactement l'action de l'attaquant. Nous avons validé notre système avec un grand nombre de données de plus de 8000 attaques réelles. Nous discutons nos découvertes et nous donnons quelques aperçus sur le comportement des attaquants dans le monde du Web.

Contents

Chapter 1

Introduction

The Web nowadays resembles a big shopping mall with a worldwide audience for its goods and services. In this medium, businesses interact with their customers via web applications, which enable the presentation and/or sale of various products and services. However, adopting this communication channel brings new challenges, one of which is the security of such web applications. Although large companies tend to take this matter seriously, small to medium businesses often lack the skills, the time, and the resources to properly secure their systems. As a result of the fact that developers are frequently not aware of the different threats and their possible consequences, even the most basic security precautions are overlooked.

One common belief amongst many e-business owners is that the Internet is huge and thus, their website won't be noticed by miscreants. However, hackers commonly use automated scanners and search engine crawlers to find vulnerabilities. While targeted attacks (such as the data breach that affected LinkedIn in 2016) require a rigorous and time consuming process (including a meticulous reconnaissance and a well planned strategy and execution), automation provides mass exposure and higher odds of success by reaching many more possible targets far quicker. Furthermore, automated tools can also be used with success by inexperienced attackers – resulting in a constant background of not very sophisticated, but still very effective, attacks that can easily reach every vulnerable website on the Web.

Despite the considerable effort of the security community, the percentage of vulnerable websites did not show any sign of recession over the past years. In fact, a stunning 76% of websites scanned by security researchers in 2016 contained vulnerabilities [Sym17, Aka17]. Adversaries are routinely searching for vulnerable servers on the Web in order to exploit and take advantage of these exposed systems. Once compromised, web applications are abused for various purposes – including deploying botnets, serving exploit kits, installing phishing kits, or simply serving as a stepping stone for launching further attacks.

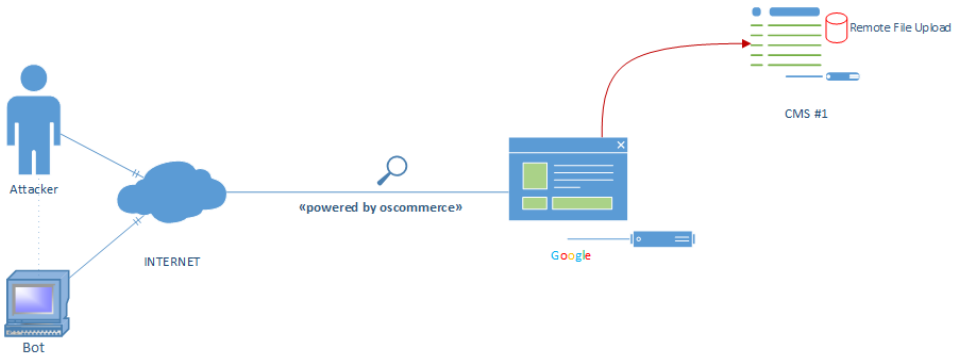

Figure 1.1: Usage of dorks for finding targets

Although cyber-criminals may also rely on dedicated web servers as part of their malicious infrastructure, compromising legitimate web servers are usually more advantageous as they are often perceived as trustworthy by other users and do not attract the attention of the security community. Moreover, a compromised Web server may allow further access to a private network that would normally be protected by a firewall. All these factors suggest that the security of web applications remains a crucial aspect of the entire security ecosystem as compromised sites still put customers, other businesses, and even public and government sites in danger.

Web Attack – An Overview

Before getting into the details of web attacks, it is essential to understand the process behind of this phenomenon. Because of the Internet's large and complex nature, the term *web attack* also has a very broad meaning. However, as this thesis focuses on the security of web applications, we narrow down our attack model to this particular domain.

The initial phase of an attack usually consists of the identification of the possible targets. For this purpose, attackers often start by taking advantage of popular search engines, by relying on automated bots to search for a set of keywords (typically called *Google dorks* [TACB16]) tailored to identify web sites that are either mis-configured or that are likely affected by a known vulnerability. For example, web sites that expose MySQL history files can be retrieved using the Google query `"?intitle:index.of?".mysql_history"`. Using the same technique it is also possible to find specific versions of vulnerable applications, for instance by looking for known banners like `"Powered by OsCommerce"`. The typical scenario of the identification phase is depicted in Figure 1.1.

Once the attacker finds her targets, she can proceed with the exploitation phase, again typically performed by automated scripts. In this thesis, we are particularly

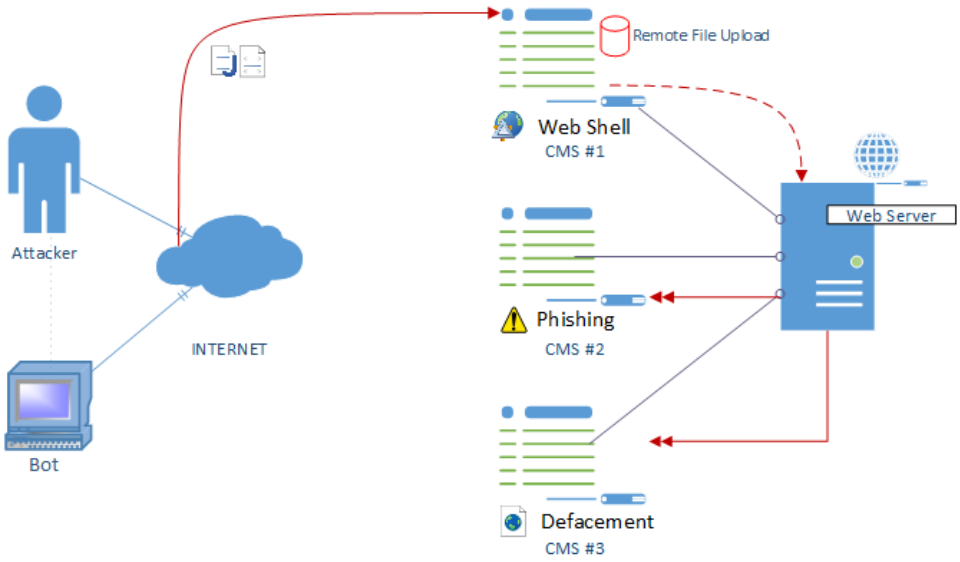

Figure 1.2: A general web attack model for web applications

interested in what happens after the attacker has successfully compromised the target application – in what Canali et al. [CB13a] called the post-exploitation phase. In this phase, presented in Figure 1.2, attackers try to achieve their final goal, which normally involves either uploading new files on the compromised machine or tampering with the content of existing code and HTML pages.

In particular, a very common tool which is often installed after an application has been exploited is called a **web shell**. The goal of these scripts is to allow the attackers to easily control the compromised machine and execute arbitrary commands by using an intuitive web interface. Web shells can be used for different purposes, such as leveraging other exploitation techniques to escalate privileges, gathering information about the system, scanning the internal network of the host machine, or exfiltrate documents and application data.

Another example of popular tool installed after a compromise is a **phishing kit**. Phishing is a type of scam to lure victims into providing sensitive data such as credit card details, passwords, or personal information used for identity theft. A phishing kit is a toolkit that mimic the appearance of existing websites (e.g., Bank of America or Paypal), thus allowing people with little technical experience to launch a phishing campaign. Individuals may use this toolkit to send spam mails or to turn the hosting machine into a fraudulent website for scamming people.

Attackers may also simply choose to **deface** the home page of the compromised web server by introducing a personalize message. Motivations behind website defacements may differ. For instance, attackers who are against a particular movement or a government (hacktivists) may choose to leave a political message,

others may simply taunt the site admins or advertise themselves for the sake of gaining fame.

Aside from the above mentioned examples, there are countless different types of malicious files and scripts that are uploaded on compromised web applications, each serving a different and specific purpose such as joining a botnet, sending large volumes of spam emails, or scanning the network for other possible victims. Throughout this dissertation, we will present many kinds of such attack instruments and we will further discuss their impact.

Understanding the Nature of Web Attacks

The analysis of nature, motivation, and technical details of how web attacks are conducted can provide an valuable information to the security community. Since miscreants usually leave traces after they compromise a system, studying their actions and the patterns they follow during and after an attack can be of great value. Understanding how miscreants infiltrate the system can help to secure it and fix possible vulnerabilities. Additionally, specific patterns, a unique file, or sometimes even just a keyword could help to identify similar cases of infections. A simple example would be an attacker who uses the same unique image to deface the homepage of his targeted web sites. Given the common usage of automated tools, such information may allow large-scale detection of the very same attack.

While no system is totally immune to web attacks, site owners can take additional precautions if they know how miscreants discover their targets. Such precautions are especially important in the existence of 0-day exploits, i.e., when the application suffers from previously unknown vulnerabilities for which a patch is not yet available. For example, a 0-day vulnerability affecting a certain version of a web Content Management System (CMS) would encourage miscreants to automatically identify such websites. By knowing the strategies attackers use to pinpoint their targets, it might be possible to evade attacks even though the website is still vulnerable per se [TACB16].

Moreover, revealing the motivation of an attacker can help to quickly recover after a compromise. If user data is leaked, clients should be notified to change their passwords; or if miscreants placed a backdoor, the security officers should patch it as soon as possible. Understanding how miscreants develop their attack vector and how they deliver malicious content are crucial for such investigation. For example, a certain behavioural pattern observed in a similar attack may help to identify the purpose of the attack, and consequently such information can be used to find impacted parts of the system.

To sum up, we believe that fighting against web attacks cannot be achieved only by building defense mechanisms. Equally important is the ability to understand the nature of the attacks and the motivation behind them.

Honeypots

In the previous section, we highlighted the necessity of having a system which allows the observation of web attacks to improve the security of web applications. In order to study the nature of the attacks at a large scale, one of the most common approaches is to set up vulnerable-looking bait systems – which are typically called *honeypots*. Honeypots can be built for almost any communication protocol, both at the network and application layer (e.g. HTTP, SSH, Telnet, and SMTP). However, since this dissertation concentrates on web attacks targeting web applications, we will only cover web-application honeypots for the rest of thesis.

We mainly separate web honeypots into two categories: *client-side honeypots* and *server-side honeypots*.

Client-side honeypots (also known as honey-clients) are software behaving like traditional web browsers which interact with remote web servers. The goal of these systems is to proactively search, find, and analyze malicious or compromised web applications. A common way to implement such system is to instrument an existing web browser [NWS+16]. These instrumented applications usually try to detect signs of anomalies while visiting a web site by, for example, tracking certain events during the execution of its JavaScript code [CKV10]. Another common use of honey-clients is to automatically collect malware samples from the web, by crawling malicious web sites [CGZ+11].

On the other hand, server-side honeypots aim to attract the attackers by exposing seemingly vulnerable services. Server-side honeypots are divided into two classes: `low interaction` honeypots and `high interaction` honeypots. Low-interaction honeypots are *simulated* systems that do not necessarily contain and run real services. While they are very convenient for information gathering and monitoring incoming attacks, their ability to capture the attack details is quite limited. For instance, they don't provide any backend or operating system functionality because they are not intended to be really exploited. Consequently, the data collected by low interaction honeypots may cover the recognition and exploitation phases but fails to accurately reflect the attackers' behavior and true intentions.

High-interaction honeypots are used instead to unravel the actual objectives of the adversaries. They are genuine systems in which the attackers can actually exploit a real web application and interact with the underlying operating system afterwards. The most common use case of deploying such system is to perform long-term monitoring of the attackers steps and inspect and analyze her behavior. Despite their advantages over simulated systems, high-interaction honeypots present also important challenges, as mentioned also in previous works [CB13a]. In particular, since they are real systems under control of an attacker, they can pose a risk for other services. Virtualisation by itself cannot contain all possible

scenarios, since a malicious web server does not only pose a threat to itself but also to other parties on the Internet.

Since this dissertation focuses on attackers monitoring and on the analysis of their behavior, high-interaction honeypots are the natural observation mechanisms adopted for this thesis. Later, we will further discuss how we overcome the challenges presented by running honeypots for long periods of time and how we tailor them according to our needs.

1.1 Problem statement

This thesis is focused on the problem of collecting and analyzing the attacker behavior by using web honeypots. More specifically, this dissertation addresses three main problems, summarized as follows:

- Existing detection methodologies are unable to keep up with the current rate at which websites get compromised. In fact, we will show how key artifacts used by the attackers can remain undetected for up to four years. Thus, new techniques are necessary for distinguishing potentially harmful websites from the benign ones in a simple and automated way.

- Thanks to the effort of previous researchers, it is now common knowledge how websites are exploited on the Web. However, the secluded part of the Web (also known as Dark Web) did not receive much attention from the security community in terms of analysis of its attack landscape. As cyber-criminals adopted the Dark Web as a platform to conduct their illegal activities, including hosting malware [O'N] and operating botnets [Bro10], it is still unclear how adversaries conduct attacks against the hidden services hosted on these private networks.

- To date, researchers have mainly focused their attention on techniques to study client-side malicious code, and only recently have looked at server-side code from a static analysis perspective. Meanwhile, incident response teams need to analyze web server logs and manually de-obfuscate scripts to try to make sense of which actions were performed as part of a successful attack. Such analysis process is very time-consuming and error-prone – and it could benefit from automated techniques similar to the ones we use today to analyze malicious binary samples.

1.2 Contributions

To address the problems presented in Section 1.1, the following contributions are presented in this dissertation.

- In Chapter 3, we propose for the first time an automated technique to extract and validate Indicators of Compromise (IOCs), which are forensic artifacts that are used as signs that a system has been compromised by an attack or that it has been infected with a particular malicious software, for web applications. We achieve that by analyzing the information collected by a high-interaction honeypot. Our experiments show that our system is able to automatically generate web indicators of compromise that have been used by attackers for several months (and sometimes years) in the wild without being detected. So far, these apparently harmless scripts were able to stay under the radar of the existing detection methodologies – despite being hosted for a long time on public web sites.

- We try to understand if the nature and the volume of web attacks have a parallel in the Dark Web compared to the traditional Web in Chapter 4. In particular, by deploying a high interaction honeypot in the Tor network for a period of seven months, we conducted a measurement study of the type of attacks and of the attackers behavior that affect this still relatively unknown corner of the Web. Our results show that web applications on Dark Web can receive automated attacks from the Surface Web with the help of Tor gateways that act as a proxy service. Moreover, we found that attacker behavior involves more manual activity instead of taking advantage of automated bots.

- Lastly, we introduce the first PHP code analysis sandbox in Chapter 5. Our system consists of an instrumented PHP interpreter and a replay component that automatically extracts the required HTTP requests from the target web server log and use them to stimulate the server-side code mimicking exactly the action of the attacker. This combination allows an unprecedented view over all steps performed during a web attack, which are captured by our sandbox and summarized in a clear report. We validated our system using a large dataset of over 8.000 real attack sessions, and we discuss our findings and distill some key insights on the behavior of web attacks.

1.3 Organization of this Manuscript

In this thesis, we collect and analyze the behavior of attackers and its impact on web applications and web servers by using high-interaction server-side web honeypots. We also conduct experiments to evaluate the nature, volume, and outcome of web attacks. The rest of the dissertation is organized as follows:

Chapter 2 – Related work

Chapter 2 provides an outline of the state-of-the-art in this field. The text provides information on client-side approaches, which are mostly relevant for Chapters 3 and 5 since they cover the automatic detection of malicious web applications and the analysis of malicious code in the wild. On the other end, while previous work on high-interaction honeypots serves as background for all following chapters, it is especially relevant to Chapter 5, because of the presented dynamic server-side analysis technique to study web attacks. Finally, the related work ends with an outline of previous studies focused on measuring the characteristics of the Dark Web, which motivates our experiments presented in Chapter 4.

Chapter 3 – Web Indicators of Compromise

In this chapter, we introduce the use of Web Indicators of Compromise (WIOC) and an automated technique to automatically extract them from compromised machines. The chapter starts from the observation, derived from several years of operation of a web-application honeypot, that innocent looking components can be used as a leverage to pinpoint compromised pages. We then explains how these items can be used as WIOCs and introcue possible use cases. We then describe which features we identified to distinguish valid from invalid indicators and present the evaluation conducted on a number of live experiments. The work based on this chapter has been published in 25th International World Wide Web Conference (WWW) in 2016 [CBB16].

Chapter 4 – Attacks Landscape in Dark Web

This chapter describes the design and deployment of a high interaction honeypot in the Tor network for a period of seven months. Our goal was to understand if the threats traditional web applications are exposed to have a parallel in the Dark Web. In particular, the chapter discusses the main differences, in terms of advantages and disadvantages, between deploying and maintaining a honeypot in the traditional Web versus operating a similar infrastructure as a Tor hidden service. We detail the different advertisement strategies and their impact on data collection and finally discuss the results of our experiments. The work presented in this chapter has been published in The 32nd ACM SIGAPP Symposium On

Applied Computing (SAC) in 2017 and won the best paper award for the System Software and Security track [CBB17].

Chapter 5 – Automatic Analysis of Web Attacks using a PHP Sandbox

In this chapter, we introduce a novel dynamic approach to analyze malicious server-side web applications. We first explain the challenges of understanding attacks against web applications and how the current static and manual analysis techniques can be time-consuming and error-prone procedure. This serves as motivation for resorting to dynamic analysis and designing a dedicated sandbox for server-side web attacks analysis. The chapter ends by elaborating on the results obtained by replayed over 8,000 attacks which helped us to demonstrate how the attackers behavior can differ based on the target environment.

Chapter 6 – Conclusions

Finally, we conclude the thesis by summarizing previous chapters, reviewing their main contributions, sketching possible future work in the area.

Chapter 2

Related work

In this chapter, we cover the state of the art of using web honeypots to better understand web attacks. The chapter is divided in three main section. First, we outline previous studies focusing on the detection of compromised websites and malicious URLs. We then present papers covering three forms of malicious code analysis: client-side approaches, server-side approaches, and mechanisms to replay past attacks. Finally, we dwell into the darker side of the web and introduce previous studies that tried to understand the attack landscape in the Dark Web.

2.1 Detection of Compromised Websites and URLs

Previous work on detecting compromised websites includes a combination of anomaly detection, content monitoring, and custom feature extraction techniques. In Chapter 3, we discuss a technique to detect compromised pages by applying indicator of web compromise. However, since there were no previous studies on this topic, in this section we include a coverage of previous works focusing on detecting and analyzing malicious URLs.

To the best of our knowledge, the work most closely related to the solution presented in Chapter 3 is the study to automatically detect malicious web pages conducted by Invernizzi et al. [IBC+12]. The authors start with a set of URLs that are already known to be malicious and then make a guided search using a web crawler to find other pages that share certain similarities with the initial set of URLs. In this process, the authors use Wepawet, Google Safe Browsing, and their custom fake AV detector to check if the guided search results are successful in detecting malicious websites. In the work presented in Chapter 3, we gathered instead URLs from remote components uploaded to our honeypot during real attacks. Instead of analyzing the maliciousness of web pages, we analyze the features of the URLs that are frequently used by the attackers. These URLs

may or may not be malicious, but they still indicate a compromised or malicious page.

Most of previous work on the maliciousness of URLs includes a classification stage implemented by using machine learning algorithms. For example, Ma et al. [MSSV09a, MSSV11] used lexical and host-based features (IP address, domain name, etc.) to classify malicious and benign URLs. Their aim is to differ malicious URLs from the benign ones by training their models with the data gathered by querying blacklists (for the malicious URLs) and Yahoo's random URL selector (for the benign set).

A more recent study presented by Soska et al. [SC14] tries to predict if a benign web page will turn malicious in the future. The authors use traffic statistics, file system structure, and the web page content as features and they use the data gathered from a number of blacklist to build their ground truth. The authors trained their classifier on over 400,000 web sites and they were able to achieve 66% true positive rate to predict if a website would turn malicious within one year.

Zhao et al. [ZH13] presents two cost-sensitive learning algorithms in order to detect malicious URLs. They evaluate theoretical performances of their proposed algorithms on a large-scale real-word data set. Their study showed that it is possible to efficiently detect a malicious URLs by querying an extremely small fraction of the dataset used for classification.

Provos et al. [PMM+07] described several server-side and client-side exploitation techniques which are used for the distribution of malware. The authors instrumented Internet Explorer in a virtual machine to analyze anomalies when a malicious binary is downloaded while visiting a defaced or malicious web site. They then visited a large number of URLs and looked for suspicious elements such as an iFrame pointing to a host known to be malicious. If no such element was found, the authors further investigated the interpreted JavaScript contained in each page.

Webcop [SASC10] aims at finding the relations between malicious URLs and malware distribution sites. The authors used the data of a commercial Anti-Malware clients to decide if a URL is hosting malicious code. Then they used a web graph constructed by a commercial search engine crawler to find the malicious URLs directly linked to malware distribution sites via hyperlinks.

Nikiforakis et al. [NIK+12] draw attention to the fact that a website can be compromised if the remotely included libraries are changed by the owner of the remote server. The authors investigated the trust relationship of websites with their JavaScript library providers. In particular, they crawled popular websites to collect millions of URLs and measure the quality of the JavaScript providers based on various features – including hosts availability, cookies, anti-XSS and anti-clickjacking protocols, and their SSL/TLS implementation. The authors

then manually assigned a weight for each feature and evaluated their metrics, showing that even the highly popular websites can get compromised through their external library providers.

Bartoli et al. [BDM10] presents a compromised website detection service, called Goldrake, based on anomaly detection. The authors monitored websites and analyzed various elements including contents of the web page, frequency of items, typical defacement signatures (e.g. common phrases, black background) without requiring any kind of assistance from the monitored web page. Although they managed to keep the false positive rate low, their work is hard to extent to detect defaced web applications in the Internet, due to the fact that the proposed solution requires to continuously monitor the selected websites.

Evil Searching [MC09] takes a different approach to the detection of compromised websites by analyzing the search queries that attackers use to find the vulnerable pages to deface. In this work, the goal of the authors is to analyze the methods that attackers use to detect possible targets in the wild. Their aim is to find phrases, called "evil searches", that can be used in phishing attacks. In comparison, in the experiments presented in Chapter 3, we instead use search engines to find "evil scripts" that can be used in many types of attacks including phishing.

Akiyama et al. [AYY+17] tried to analyse the malicious behavior of URL redirections by conducting a long-term measurement study. For this purpose, they implemented a honeypot-based monitoring system that ran for four years and observed over 100K malevolent URL redirections which were collected from 776 different websites. The authors discovered that domain generation algorithms, which are usually adopted by botnets, gained popularity for the sake of increasing the entropy of URLs to evade blacklisting. By using the results of their experiments, the authors then proposed a set of countermeasures against such malicious redirections.

Corona et al. [CBC+17] presented an anomaly detection approach to detect phishing pages in compromised websites. The authors simply used the HTML source code and visual differences of potential phishing pages compared to legitimate web pages as key features of their presented work. They claim that their approach also works against evasion techniques based on manipulation of the HTML code of the phishing page to mislead pre-deployed security measures, such as Google's Phishing Pages Filter. They experimented with more than 1K phishing pages and almost 4,5K legitimate websites to evaluate their method. The results showed that their technique, DeltaPhish, was able to correctly detect 99% of the phishing pages with only 1% misclassification rate.

A static analysis approach that analyses the changing features of the two different versions of the same website was proposed by Borgolte et al. [BKV13]. The authors leveraged a machine learning algorithm to outline the changes of the website and determine whether they were harmful or not. Based on change-

related features, the researchers implemented a prototype called Delta-system which was able to detect web-based infection vectors. More specifically, the authors used a fuzzy tree difference algorithm to extract significant changes in the DOM tree and discard the small modifications. The paper includes an evaluation of the Delta-system over 26 million website pairs.

Finally, the same researchers, who this time focus on the detection of defaced pages, presented a tool called Meerkat [BKV15], in which they trained a classifier with the screenshots of the defaced web pages. After the learning phase, Meerkat was able to automatically detect newly-defaced websites. While not strictly related to our objective and methodology, our system can also be adapted to detect defaced websites by identifying their possible use of certain indicators of compromise.

2.2 Analysis of Malicious Code on the Web

Malicious code on the Web exists in many different forms and researchers have proposed a wide range of approaches to study and measure its characteristics. Thus, we divide previous work on malicious code analysis in three subsections, respectively dedicated to client-side approaches, server-side solutions, and attack replaying techniques.

2.2.1 Client Side Approaches

Most of previous research on the analysis of malicious code on the Web has focused on the client side. In particular, many researchers have looked at how to identify exploit kits [EV14, mWBJ+06, SLZ16], JavaScript malware [CKV10, RLZ09, KLZS12, KSC+13] and malicious Flash advertisements [FCKV09].

Cova et. al. [CKV10] proposes to use an instrumented web browser to track certain events during the execution of JavaScript or interpretation of HTML code. Some of these events that were found to be strong indicators of malicious behavior include a high number of redirections or the use of JavaScript functions related to dynamic code evaluation and deobfuscation. The findings were then assessed with an anomaly detection tool to identify malicious content.

Eshete et. al. [EV14] analyzed the inner workings of exploit kits and incorporated this information into machine learning features to detect if a given URL is hosting an exploit kit. They performed their analysis by setting up a virtual environment for exploit kits and visiting the corresponding URLs of the kits with honeyclients. Once the features of each exploit kit were extracted, the authors trained their learning algorithm and later evaluated its accuracy by conducting a number of live experiments.

Honeymonkey [WBJ+06] is a client-side honeypot, which consists of several vulnerable web browsers running in virtual machines of different patch levels. The honeypot can be used to scan web pages to detect the malicious ones. The authors used these virtual machines with vulnerable browsers to visit various URLs and observe the exploit-related modifications occurring to the system (such as newly created executables, child processes, registry entries). Moreover, for malicious URLs, a redirection analysis was performed to get all the steps involved in the exploit. Finally, the same malicious URLs were analyzed on fully patched virtual machines to test if the attacks were still successful. This process also helped to find zero-day exploits. In their experiments, the authors discovered 752 URLs from 288 web sites that can successfully exploit different browser vulnerabilities.

Stock et al. [SLZ16] proposed an approach that aims to take advantage of reused code segments in exploit kits for signature-based detection. The authors first clustered exploit kits according their unpacking behavior, using the k-means clustering algorithm. Then, they located a common substring of maximum 200 tokens for each cluster. Such sub-sequences were then analyzed to create a regular expression that can be used as a signature. To evaluate the generated signatures, the authors run a one month-long experiment, where they found that the accuracy of their automatically generated signatures are comparable to those manually created by experts.

In Nozzle [RLZ09], the authors tried instead to detect heap spraying attacks where the attackers use embedded JavaScript code to exploit browser-related vulnerabilities. The proposed method monitors heap objects at runtime by instrumenting memory allocation and deallocation routines. This approach was used to detect several published exploits, as well as synthetically generated ones. In Rozzle [KLZS12], the authors developed a $multi\text{-}execution$ virtual machine to analyze multiple execution paths of a JavaScript code in a single run. This approach can identify environment-dependent JavaScript malware much more efficiently, compared to other traditional static and dynamic analysis techniques.

Kapravelos et. al. [KSC+13] proposes to dynamically execute JavaScript code in a browser emulator, to detect evasive behavior and identify scripts similarity (polymorphism). Using the data collected from Wepawet [CKV10], the authors detected more than 4K evasive scripts and more than 101K scripts that contain code in a packed form.

Taylor et al. [TSOM16] introduced a honeyclient to detect malicious behavior of the visited websites by temporarily caching the HTTP traffic to reduce the cost of their analysis on selected suspicious unseen exploitable files. By imitating the client, they dynamically run exploit kits in a controlled virtual environment and compared the performance of the honeyclients against content-based signature approaches, showing that their method outperforms other solutions at the time of the attack. They authors ran their framework on a campus network for five

days to conduct a live traffic analysis.

J-Force [KKK+17], a forced execution engine for JavaScript, analyses the function parameters to discover suspicious DOM injections and further reveal malicious behavior of the web-based malware. Its authors evaluated J-Force's effectiveness on 50 exploits taken from popular exploit kits and over 12K Google Chrome extensions.

2.2.2 Server-Side Approaches

Researchers have also investigated solutions to analyze server-side malicious code, but so far mainly from a static analysis perspective.

In this category, the majority of previous studies focused on the static analysis of exploit kits [KM, AKM13, DMKS+14, EAM+15]. Among them, Kotov et al. [KM] reported on the leaked source codes of 30 different exploit kits. They showed that such tools actually don't have a complex and sophisticated architecture to pinpoint the client vulnerabilities but rather adopt a brute-force approach. In [AKM13], authors again followed a similar path by analyzing ten leaked exploit kits from the black market. They deployed each exploit kit in an isolated environment called `MalwareLab` and tested the reliability of such tools against changing software configurations. The authors also reported on different strategies taken by the adversaries when implementing such tools and explained the trade-off between the longevity of the exploit kit and its infection rate.

Maio et al. [DMKS+14] extended `Pixy` [JKK06b], a data flow analysis tool for PHP, to propose a system called `PExy` for the automated static analysis of the source code of exploit kits. Using this tool, the authors performed taint and behavioral analysis to produce signatures for exploit kits. In addition, Eshete et al. [EAM+15] identified several vulnerabilities in exploit kits and demonstrated the feasibility of automated exploit generation for those kits.

Server-side honeypots have also been used to study web-based attacks. Canali et al. [CB13a] have analyzed the attack post-exploitation phase by redirecting the traffic from 500 domains to a high-interaction honeypot containing different vulnerable applications. Although the authors made some analysis on the attacks collected using their honeypots, our approach presented in Chapter 5 is much more comprehensive, as it allows the analyst to monitor each step of the attacks thanks to our instrumented PHP interpreter – instead of simply analyzing the uploaded files as done by Canali et al.

Starov et al. [SDA+16a] focused instead on malicious web shells and they reported on how attackers can take advantage of visible/invisible features and on the backdoors often planted in those shells. While the authors also adopt a dynamic approach by setting up honeypots, they only focused on the analysis of

the web shells' home phoning feature, which is a mechanism that notifies the author once a web shell is installed on a compromised web site. Our dynamic approach allows us to also precisely observe how attackers are interacting with the uploaded shells.

Han et al. [HKB16] make use of server-side honeypots to collect different kinds of phishing kits and sandbox them in order to monitor not just attack itself but also the victims interaction with the phishing kits. The authors measured the lifetime of phishing kit and analyzed the distinguishing behavioral factors of attackers, victims, and third part researchers. The valuable findings presented in their work emphasize once again the importance of dynamic analysis of web malware.

2.2.3 Replaying the Web Attacks

Another line of research close to our work focused on recording and replaying web attacks for forensic, offline analysis, and repeatability reasons.

In this category, `Clickminer` [NPLN14] reconstructs the user-browser interactions by actively replaying the HTTP network traces via an instrumented web browser. Researchers were able to reconstruct almost 90% of the user-browser interactions with around 1% false positive rate through a user study that involved 21 participants.

Nelms et al. [NPAA15], on the other hand, focuses on the web paths which lead real users to download a malware. The authors tried to automatically analyze the sequence of web pages prior to the attack by deploying their proposed system, called `WebWitness`. Then, by using the information they gathered during their analysis, the authors identified the malware download paths and discovered the most prominent attack trends. The same authors also proposed a defense module to decrease the infection rate for a specific type of drive-by-download attack.

Chen et al. [CGZ+11] implemented a system to automatically collect web-based malware by using honey clients and replaying malware infection scenarios. The presented system mimics a real human interaction by simulating the web browsing behavior of the real users. Using this approach, the authors identified Ĩ,8K exploit-URLs hosted in 741 web sites.

Finally, Mohamed et al. [MAS16] proposed a methodology for forensic examination of Web-browser artifacts and implemented a Firefox browser-extension to investigate malicious URLs that host malicious executables. By comparison, our work presented in Chapter 5 aims instead at understanding the attackers' behavior on the server side, by replaying their interaction with the vulnerable web application or with the additional components uploaded during the attack itself.

2.3 Previous Studies on the Dark Web

Attacks against web applications have already been studied by using either low-interaction or high-interaction web honeypots as we previously covered in Section 2.2. However, all these works targeted the Surface Web and we believe we are the firsts to document attacks in the Dark Web by mean of the practical deployment of a high-interaction honeypot.

A related set of studies focused on measuring the characteristics of the Dark Web, including its size, the connection between websites, and the different services (i.e. protocols) provided over the Tor network. OnionScan, for example [Lew], leverages hyperlinks contained in web pages and other features (like correlation on `ssh` fingerprints and `ftp` banners), to build relationships among hidden services. This dataset consists of about 5,600 active sites that were scanned in June 2016.

In a similar work, Ciancaglini et al. [CBMR] actively crawled the Dark Web for a period of two years and reported on the cyber-criminal services and illicit goods available in the Dark Web– including marketplaces, laundering services for crypto-currencies, and hosting platforms for malware.

When it comes to attacks in the Dark Web, or against the darknets used to operate the Dark Web, a consistent amount of literature has been produced. A first category of papers propose attacks aimed at de-anonymizing hidden services, e.g. by recovering the public IP address on which the hidden service operates. CARONE [MKC15] makes use of heuristics to match information in the content of the hidden service and certificates chain with candidate Internet endpoints. Kwon et al. [KAL+15] propose instead an attack in which a combination of website fingerprinting and circuit fingerprinting techniques are used to de-anonymize hidden services. While website fingerprinting is already widely used (e.g., in the Surface Web), authors revealed that during the circuit construction phase between clients and hidden services, darknets as Tor exhibit fingerprintable traffic patterns that allow an adversary to efficiently and accurately identify and correlate the circuits involved in the communication. Panchenko et al. [PLZ+16] propose a more general approach that identifies the content of encrypted and anonymized connections (e.g., Tor) by observing patterns of data flows such as packet size and direction. Researchers from Carnegie Mellon recently received media attention when they revealed their ability to de-anonymize users and hidden services in Tor [car, 201].

A different class of attacks has been analyzed in [WKM+14] and [SN]. Winter at al. [WKM+14] document malicious exit relays in the Tor that are injecting/-modifiyiing HTML and conducting man-in-the-middle (MitM) attacks over TLS and SSH. The authors developed `Exit Relays Scanners` for credential harvesting and MitM attacks, and used them to identify malicious exit relays nodes. By operating two exit relay scanners for several months, they found 65 relays that were misconfigured and/or malicious. They also demonstrated a number of

countermeasures in which they implemented scanners that probe the exit relays for different kinds of MitM attacks.

More recently, Sanatinia et al. [SN] exposed another category of misbehaving Tor relays that are integral to the functioning of the hidden services. In their short paper, the authors found that some of the Tor relays, more specifically Hidden Service Directories (HSDirs) which are somehow the equivalent to DNS servers on Surface Web, indeed scans the Tor network for well-known vulnerabilities. However, their report on this phenomenon is very limited and covers only minimal part of the real attack landscape of the Dark Web.

On top the Dark Web-specific attacks described so far, denial of service (DoS) attacks against hidden services have also been reported in the wild [FB]. In fact, with the increase on the number of business-related websites been deployed in hidden services, well-understood attacks (like DoS) have been observed more and more frequently in the Dark Web. What it is not clear, so far, is how much a hidden service is exposed to threats like web-based attacks (e.g. SQLi, path traversal, etc..), bruteforce attacks, and how these attacks are conducted in the Dark Web– e.g., if manually or automatically. This thesis is trying to answer this questions in Chapter 4.

In a recent study by Sanchez-Rola et al. [SRBS17], the authors explain the link between Surface Web and Tor hidden services and analyze the web tracking behavior of those services. Though this work focuses on the privacy analysis of the web pages hosted in the Tor network, it is essential for revealing the connected nature of Surface Web and Dark Web. In fact, the researcher found out that more than 20% of the resources are fetched directly from Surface Web. This also proves that the attack landscape of the Dark Web is worth exploring as it may suffer from the same weaknesses present in the Surface Web and further emphasize the importance of our work presented in Chapter 4.

Part I

Analysis of Web Attacks Based on Live Honeypot Data

In this first part we investigate how to deploy and use high-interaction web-application honeypots to collect data about web attacks, process this information to better understand the attackers behavior, and use the results to protect users or detect compromised applications.

This part covers two separate contribution. In Chapter 3, the collected data from a traditional web honeypot is used to build an automated detection mechanism for compromised websites. Here, we address the common misconceptions about honeypots and discuss which steps should be taken to benefit from the artifacts uploaded by the miscreants.

In the next chapter, we shift our attention to the Dark Web. Our goal is to explain the differences and possible similarities between Surface Web and Dark Web in terms of their attack landscape. We will show how, despite the isolated nature of the Dark Web, the web sites hosted as Tor hidden services may still be exposed to large volumes of attacks coming from the Surface Web.

Chapter 3

Web Indicators of Compromise

**This chapter is based on a publication which has been presented at the World Wide Web conference (WWW) in 2016 [CBB16].*

In 2013, Canali et al. [CB13b] performed a study to measure the typical behavior of an attacker after a website has been compromised – showing that many attacks result in the installation of new pages (e.g., phishing kits) or the modification of existing ones (e.g., to serve malicious code or redirect the victims to another location). This happens so frequently that it is very hard for the security community to react in time, detect the malicious or newly infected web pages, and update existing blacklists (such as Google SafeBrowsing [Goo15a] or Phishtank [Phi15]) to protect users.

In fact, existing approaches based on honeyclients [WBJ+06, IHF08], web crawlers [CCVK11, SKV13, IBC+12], or user reports [Phi15], are not able to keep up with the current rate of infections. Therefore, we need new techniques to automatically distinguish, in a simple but effective way, "bad" pages from benign ones. To detect the presence of malicious programs in traditional systems, the forensics community relies on *Indicators of Compromise* (IOCs), i.e., simple network or operating system artifacts whose presence is a reliable indicator of a computer intrusion or malware infection. For example, the presence of a certain entry in the Windows Registry or of a file with a given MD5 in a temporary directory may be associated to a certain banker trojan. These indicators are often used as part of malware detection and investigations [HYL13] and are often shared between experts as part of other threat intelligence informations [Man15]. Unfortunately, to the best of our knowledge, the use of indicators of compromise has never been studied in the context of web applications.

Our work starts from a simple observation, which we made after operating a web honeypot for several years: Attackers often use external components in their malicious or compromised pages. For example, these pages often rely on JavaScript code to perform a wide range of actions. In our experience we noticed that these accessory scripts are rarely installed by the attacker on the compromised hosts, but they are instead included from public URLs hosted on remote machines. A possible reason for this behavior is that this choice provides more flexibility for the attacker to update these components without the need to modify all the pages they had previously compromised. However, this may also seem like a potential weakness, as this part of their infrastructure could be easily detected and taken down – jeopardizing a large number of infected pages.

Quite surprisingly, while investigating some of these remote components, we discovered that in the vast majority of the cases they were not malicious per se. For instance, we identified three main classes of external components: popular JavaScript libraries (e.g., `jquery`), scripts to control the look and feel of the page (e.g., by adding dynamic effects to its text), or scripts that implement reusable functionalities (e.g., to fingerprint the user browser, to disable the right click of the mouse, to overlap the page with transparent frames, or to insert an advertisement banner in the page). Since none of these categories is harmful to the final user, these components can be safely hosted by the attackers on public pages or shared hosting services, with no risk of being detected and blocked by security scanners.

The main idea behind the work we present in this Chapter is that, while these components are indeed innocuous, their presence can be used to precisely pinpoint compromised or malicious pages. In other words, the link to a particular benign JavaScript can be considered as some sort of signature of the attack – therefore acting as an *indicator of compromise for web applications*. We call this new types of indicators, Web Indicators of Compromise (WIOCs).

We believe that the extraction and use of indicators of compromise has several important advantages. In particular, while most of the existing approaches focus on the detection of *malicious* pages, our solution allows to detect *compromised* pages. This category is much broader and much harder to identify in a black-box manner. In fact, compromised pages are not necessary harmful for the user browser, but also include defacements, phishing pages, or banners to redirect users into other web sites.

Our experiments show that our system was able to extract, in average, one new indicator per day. These indicators were then used by Trend Micro, a popular antivirus vendor, to cross-check their customers' requests in their web telemetry dataset, finding thousands of users each day visiting previously unknown compromised websites.

To summarize, this Chapter makes the following contributions:

- To the best of our knowledge, we are the first to propose the use of indicators of compromise for web applications.
- We propose a novel technique to automatically extract and validate these indicators – starting from the data collected by a web honeypot.
- We discuss several features that can be used to distinguish good indicators of compromise from components that are also used as part of benign websites.
- We tested our system over a period of four months. In this period, almost 100 new WIOCs were extracted and validated from our dataset. Finally, we use these indicators in collaboration with Trend Micro to estimate the number of users that are affected by compromised webpages that include these components.

3.1 How Web Applications gets Compromised

In Chapter 1, we briefly explained how attackers approach and compromise a vulnerable web application, summarizing their actions during and after the attack is performed. There, we described the methods followed by the attackers including how they take advantage of search engines and how they rely on automated bots. This process, also described also in previous studies [CB13b, JYX+11], serves as motivation for this study.

Here, we are particularly interested in what happens after the attacker has successfully compromised the target application – in what Canali et al. [CB13b] also called the *post-exploitation phase*. In this phase, attackers try to achieve their final goal which could be to install a webshell, to deface the home page with a political message, to send spam, or to install a phishing page as we already discussed previously. These goals are generally achieved by either uploading new files on the compromised machine or by modifying the sources of the existing HTML pages. Either way, the attacker often needs to use a number of JavaScript libraries which can be uploaded as well on the compromised machine or just included from a remote source. Since our primary goal is to identify indicators of compromise for web applications based on these remote components, in the rest of the chapter we will not focus on means of exploitation and on the techniques commonly used to compromise the web applications.

One common misconception about the post-exploitation phase is to consider all the components uploaded by the attacker after a successful exploitation as malicious. Although among all these uploaded components a portion of them is indeed responsible to perform some sort of malicious activity (such as malware

```
1  ...
2  <head>
3  <meta http-equiv="Content-Language" content="
       en-us">
4  <meta http-equiv="Content-Type" content="text/
       html; charset=windows-1252">
5  <title>4Ri3 60ndr0n9 was here</title>
6  <SCRIPT SRC=http://r57.gen.tr/yazciz/ciz.js>
       </SCRIPT>
7  ...
```

Figure 3.1: HTML example of a compromised web page

distribution, exploit kits, or phishing pages), we discovered that the majority of them are often not related to any type of malicious behavior. On the contrary, the post-exploitation phase usually involves the usage of a number of harmless JavaScript components to work properly.

For example, Figure 3.1 shows a snippet of code extracted from a compromised web application. In this example, the attacker injects a remote JavaScript code (i.e., `ciz.js`) to the defaced victim web page. By retrieving this remote component, we discovered that it only contained two lines of code, which are reported in Figure 3.2. The script first creates a new image object and then its source URL is set according to the value of `location.href`.

The goal of this component seems to be to log compromised web pages by sending a signal back to the attackers. Interestingly, the same script is included in a number of popular web shells which are distributed (and backdoored) by the same hacking group, as a mechanism to promptly detect and gain access to third party installations. Even though this code may look suspicious when manually examined because of the use of the `r00t` leetspeak in the URL, automated scanners only looks at the maliciousness of the file itself and, inevitably, this simple piece of code is not detected as malicious by any available system or antivirus product. As a result, this JavaScript component could be hosted on any public page, without the risk of raising any suspicion from security tools. Moreover, this gives the attacker the advantage of rapidly changing the URL in all compromised pages, without the need to re-deploy the JavaScript file on all the target machines.

As we further investigate the websites which use this JavaScript as an external resource, we foud that other websites which include the same script were also compromised by the same hacking group. Also in the other compromised sites the script was included at the same place in the code as it is shown in Figure 3.1, and all the defaced pages looked identical when visited.

```
1 a = new /**/ Image();
2 a.src = 'http://www.r57.gen.tr/r00t/yaz.php?a=' +
    escape(location.href);
```

Figure 3.2: Source code of `ciz.js`

This very simple example perfectly summarizes the idea behind our technique: a little and harmless script included by attackers in compromised pages could be used to precisely fingerprint the action of these attackers and therefore can serve as an *indicator of compromise* for web applications. In our experiments, as described in more details in Section 3.3, we identified many of these examples ranging from few to thousands lines of code, and from custom scripts to popular libraries. We believe that this type of indicators of compromise can complement existing detection techniques that are purely based on manual reports or on automated scanners that – unfortunately – can only identify malicious components.

3.2 Approach

As explained in the example presented in the previous section, our idea is to analyze compromised and malicious web pages – looking for seemingly innocuous external resources that can be used to identify a certain group of attackers or a certain attack campaign.

In the rest of this section we describe each step of our automated technique.

3.2.1 Data Collection

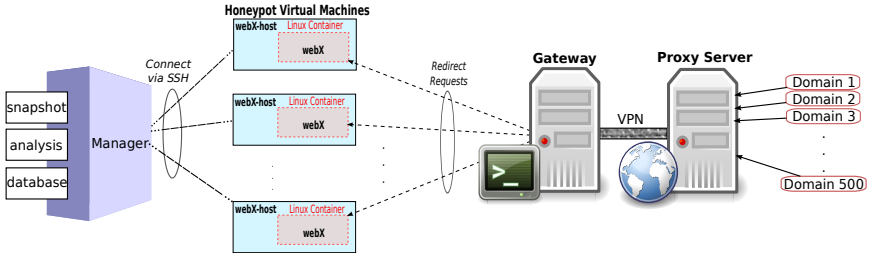

Figure 3.3: Overview of the Honeypot Infrastructure

The first component of our system is a high-interaction honeypot that we use to observe the behavior of the attackers and collect the pages they modify or they

upload into the vulnerable system. The role of this component is only to collect a large number of pages compromised by attackers. Other techniques could be used to obtain a similar dataset, for instance by crawling the web or by using intelligence feeds from security companies.

Our honeypot infrastructure, summarized in Figure 3.3, is implemented as previously described by Canali et al. [CB13b]. The deployment consists of proxy services (associated to 500 different domain names), which redirect the traffic through a VPN gateway to seven virtual machines running in our premises. Each VM is responsible to run a different vulnerable web applications isolated in a Linux container. As attackers exploits these applications, they gain full control of the corresponding container – where they are free to modify existing web pages and install new ones.

In order to discover what has been modified after an attack is performed, each VM automatically collects and compares the original state of the container with the exploited state. When a difference is detected between two states, all the files that are modified or uploaded by the attacker are extracted by our system. Moreover, the vulnerable applications are reverted back to their original "clean" state at the end of each day. All the collected data is stored in a database hosted by the `manager` machine, which is also responsible to run the subsequent analysis.

We configured each virtual machine to prevent attackers from using the honeypot as stepping stone to run attacks and propagate over the network. For this end, we run all services as non privileged user and keep each of our honeypots up to date with software and security patches. Additionally, we drop all outgoing connection in order to prevent attackers to use our system to perform attacks or send spam messages. We also mitigate the problem of hosting malicious content by reverting virtual machine back to its clean state on a regular basis.

3.2.2 Extraction of Candidate Indicators

The second component of our system is in charge of pre-processing the collected data to automatically extracts the URLs of the components remotely included in the attackers' files, and to store them along with some additional information in our database. This requires our system to analyze each HTML file and collect the addresses of all the external resources.

In addition to the URL, we store the $base\ date$, which is the date when the URL was first seen in our honeypot, and the $last\ date$ in which we observed it. Moreover, our system periodically probes on a daily basis each URL to verify if it returns a valid response. If it does not encounter any error, it updates the $last\ good\ response\ date$ in the database. Finally, we also store the information of how many times a URL is included in the uploaded components from the base date to the last date.

While our technique can be applied to any resource type (e.g., JPEG images), in this work we focus in particular on JavaScript files. In particular, since we extract the JavaScript URLs from the uploaded files after an actual attack is performed, one would probably expect that the vast majority of these scripts would contain malicious code. However, a manual inspection reveals that it is quite common for an attacker to include simple scripts that implement simple visual effects or common JavaScript libraries in their code. Unfortunately, this makes the identification of indicators of compromise much more complex. In fact, considering the fact that many of the scripts are not malicious in nature but they might still be used for malicious intents, it is impossible to tell whether a certain URL is in fact a good indicator of compromise by just looking at the content of the JavaScript code.

For example, simple scripts designed to prevent the right click of the mouse, which are not malicious per se, are widely used by attackers to prevent users from inspecting the source code of an infected page. However, to be certain that one of these scripts can be used to indicate that an attack has been successfully performed against the website which includes it, we need to extend our analysis to inspect not just the script content per se, but also the *context* in which it is used and the other pages on the Web that import it.

3.2.3 Searching the Web for Indicators

As we shift our focus more on the web pages that include each candidate indicator, we need a way to search the World Wide Web for a particular snippet of source code. Unfortunately, common search engines do not provide such functionality. For example, Google provides a search operator, `intext:`, that lets users search for a term contained in the text of the document. However, this only includes the content of a page that is displayed to the user, and not its HTML tags. As a result, it is not possible to use popular search engines to search for JavaScript entries or for other file included in a HTML document. Therefore, we needed a more sophisticated tool that indexes the source code of each visited page. For this reason, our prototype applications uses $Meanpath$ [Mea15], a search engine that also captures and index HTML and JavaScript source codes. While Meanpath does not have the same coverage as other classical search engines such as Google or Bing, its archive of over 200 Million web sites can help us to identify the web pages that actually *include* external scripts. In our context, these scripts are the ones pointed by our candidate indicator URLs.

3.2.4 Features Extraction

For each candidate indicators, we extract five different groups of features:

- **Page Similarity**
  The vast majority of the attacks are largely automated, and therefore attackers tend to re-use the same template for each website they compromise. We capture this characteristic by looking at the similarities of the web pages that include a candidate indicator URL as an external resource. For this purpose, our system automatically queries Meanpath to find websites that include the candidate URLs and it then downloads the HTML source code of the first 100 results. We use a fuzzy hashing algorithm ($ssdeep$ [Kor06]) to compute the similarity of the content of each website and then group the similarity of each unique pairwise comparison in one of five categories: low (similarity below 0.25), medium-low (0.25 to 0.5 similarity), medium (0.5 to 0.75 similarity), high (0.75 to 0.97 similarity) and perfect match (higher than 0.97 similarity). For each class we count the number of web pages that falls in the corresponding range. So if the $high\ similarity\ count$ of a candidate indicator is high, it means that our tool came across almost the same content over and over again in the top 100 websites that include that indicator. Likewise, if the $lowest\ similarity\ count$ is high, it means that all the websites that include the candidate URL have almost nothing in common.

- **Maliciousness**
  Although the majority of the candidate indicators are not malicious, they are often included as an external resource inside malicious pages. Hence, we also compute the maliciousness of the top 100 web pages that include a certain candidate URL, as part of the features we use to distinguish a normal script from a indicator of compromise. To this end, we automatically scan each website using the VirusTotal API [Goo15b] and Google SafeBrowsing [Goo15a]. We then categorize websites into three categories according to their maliciousness level: *maybe malicious* if less than five AV detected as so, *likely malicious* if five to ten AV return a positive match, and *malicious* if it is identified as so by SafeBrowsing or when the positive matches are more than 10 out of the 60 available AVs. Finally, we use the total number of websites in each category as features for clustering candidate indicators.

- **Anomalous Origin**
  We also observed that attackers sometimes use popular JavaScript libraries in their pages. However, instead of including them from their original domain, they host their own copy on other servers under their control.

For instance, an attacker may include the very popular JQuery library (e.g., `jquery-1.11.3.min.js`) not from `jquery.com` but from a personal server located in Russia. This could be a suspicious behavior, and in fact we encountered many examples in which web pages that include popular JavaScript libraries from external domains were compromised. In particular, we observed two different phenomena. First, some attackers use popular library names to hide code that has nothing to do with the library itself. For instance, we found a `jquery.js` file that was used to disguise a modified version of the `ciz.js` script shown in Figure 3.2. In a different scenario, attackers use instead a copy of the original script, often obfuscating its content (possibly to hide small modifications or customizations of the code). While this feature alone is not sufficient to generate WIOCs, our experiment demonstrates a high correlation between these cases and compromised websites.

- **Component Popularity**
  As the popularity of the external component increases, it is less likely that it is associated only to malicious activities, and therefore that it is a good indicator of compromise. For instance, some scripts associated to the Facebook Software Development Kit (e.g., `connect.facebook.net/en_US/all.js`) can also be found in the remote components uploaded by the attackers on our honeypot. However, since the same script is used by millions of other websites, it is unlikely that it is used only for malicious intents. Even if this was the case, it would have probably already attracted the attention of the security community and therefore other protection mechanisms and blacklists would be sufficient to protect the end users. Therefore, in our system we use the total number of search results from Meanpath as a feature to filter out very popular URLs.

- **Security Forums**
  In addition of using Meanpath to retrieve the pages that include a certain resource, we also query Google to collect how many times the candidate indicator is *mentioned* on the Web. From these results we extract two separate features: the total number of search results, and how many of the top 10 results mention the candidate indicator together with certain security related keywords such as "hacked", "malware", "compromised", and "antivirus". This is used to capture online forum discussions or threat web pages maintained by antivirus companies – in which people discuss the role of certain JavaScript files or ask for more information after a piece of JavaScript has been detected in their websites.

3.2.5 Clustering

After we automatically extracted all the features for each candidate external URL component, we applied an unsupervised learning algorithm to separate different classes of components. The reason for not using a supervised classifier is that it would require a considerable effort to build a ground truth. In fact, verifying if a certain URL is a good WIOC can take a large amount of time also for a skilled manual analyst. On the contrary, we believe that the features of good and bad indicators would differ enough to be clearly separated by a clustering algorithm.

In particular, we are interested in differentiating three main cluster categories:

- **Good Indicators of Compromise**
  This category includes the components that are, to the best of our knowledge, used **only** by attackers when they compromise a web page or install a malicious one. Although in our experiments the page similarity was the most distinctive feature to detect good indicators, all features contributed to the identification of this category.

- **Invalid Indicators of Compomise**
  This category covers the opposite case, in which a certain component is used as part of attacks but also as part of benign pages. As expected, the most distinctive feature in this category is the popularity of the candidate URLs.

- **Undecided**
  This cluster category describes those components for which the available data was not sufficient to take a final decision. Therefore, the URLs which fall into this category cannot be labeled as either good or bad indicators, even after a manual inspection. In fact, some components are so rare that both Google and Meanpath return no results (even though the remote JavaScript is online and can be retrieved by our system). In other cases, only few of matches are found in the search engines. Even if they were all examples of compromised pages, it would still be too risky to classify the indicator with such a limited amount of information.

We conducted a number of experiments with different thresholds and finally obtained the best results by using the K-means algorithm with k equal to eight. Other values of k may provide equivalent results, as our goal in this phase is only to show that it is possible to clearly separate the different behaviors in distinct groups. With this setup, the clustering algorithm was able to clearly separate each behavior and group candidate indicators in clusters that only contained a certain type (valid, invalid, or undecided).

To verify the accuracy of our approach, we manually verified a number of random items picked from each cluster. Out of the eight clusters identified by our

algorithm, one contained only bad indicators, five only good indicators (three mainly defacements and two mainly malicious pages), and two were associated to the undecided group. In the Experiment Section we report on the accuracy of our clustering approach when applied to categorize potential indicators extracted by our live honeypot.

3.2.6 Impact on End Users

To measure the impact of our technique, we collaborated with Trend Micro, a popular antivirus vendor, to estimate how many real users have interacted with our WIOCs. Using a cloud-based infrastructure, the vendor collects over 16 terabytes of data per day from 120 million client installations worldwide. We based our analysis on a subset of this data, based on a telemetry feed that collects information on the URLs that are accessed by users over HTTP(S) – using their browser or any other client.

Whenever one of the AV client visits a page that includes our web indicators of compromise, her browser sends an HTTP request to fetch the missing component and we can detect and log this action.

In an operational environment, we envision that our approach could be deployed in three ways. First, to generate a blacklist that a company can use to prevent users from visiting compromised web pages. Second, by combining the `Referer` HTTP header with the telemetry information, a security company can use our indicators of compromise to automatically discover, in real time, new URLs of infected pages. While we were not able to test this configuration in our experiments, we believe that this scenario would provide even greater advantages compared with other existing mechanisms to detect malicious web pages. Finally, our indicators could be used as seeds to quickly search for malicious or compromised pages on the web. It would be enough to query for the pages that include these components to build a list of candidate targets, which can then be visited with more sophisticated scanners or honeyclients solutions.

3.3 Experiments

In this section, we explain the tests we conducted to evaluate our approach and the results of our experiments. We also discuss the impact of our solution by correlating our data with the telemetry information of Trend Micro.

3.3.1 Dataset

Over a period of four years, our honeypots collected over 133K unique URLs of remote components – either uploaded by the attackers as part of their pages or

as modification of the honeypot pages themselves. Note that in this study we were not interested in distinguishing between attack types, nor in measuring the frequency of attacks, or time period between successive threats. The reader may refer to previous studies [CB13b] for a detailed analysis of these characteristics.

Out of all the remote components, our analysis focused on 2765 unique JavaScript files. In average, each of them was re-used several times (an average of seven and a maximum of 202) as part of different, likely automated, attacks. However, more than half of the JavaScript URLs were observed only once – as confirmation that our honeypot also captured unique events probably performed manually by the attackers.

To test our system, we trained our feature extraction and validation routines on the data collected between January and April 2015. While older data was available in our database (and it was used to analyze long-lasting campaigns), some of the features used by our technique need to be computed in real-time. Therefore, we were forced to operate only on the attacks performed after we started our study of indicators of compromise. We then used the result of the clustering to classify the new URLs observed by our honeypot over a period of four months starting in mid-April. The results are presented in the following sections.

3.3.2 Model Training

In order to evaluate our work, we first used our clustering approach to divide the URLs in the training set in different categories. The dataset included 373 candidate indicators. The clustering was performed using `Weka` [HFH+09] – a common open source tool for machine learning and data mining tasks. After the clustering operation was completed, we manually inspected the content of each cluster to assign it with the correct label (i.e., good indicator, invalid indicator or undecided), as described in Section 3.2.5.

This phase allowed us to tag five clusters as valid web indicators of compromise, for a total of 12% of the total number of candidate indicators. However, the goal of the clustering was not to detect indicators, but instead to separate the features space and provide reference values for the next phase.

3.3.3 Results

Our live experiment was conducted over a period of four months. During this time, the honeypot continued to collect new URLs of external components and to pass them to our analysis framework. The analysis system collected the external information and then computed the individual features. Finally, it computed the distance between the features of each URL and the different clusters built

during the training phase and it assigned the URL to the category of the closest cluster. So, if a new candidate indicator was close to a cluster marked as "Invalid Indicators", the URL would be considered invalid as well and discarded. If, instead, the closest cluster was flagged as "Good Indicators", then the candidate URL was considered valid. Table 3.1 shows the results of our classification.

As we already mentioned, the page similarity was the most distinctive feature, followed by the presence in security forums and by the number of hits in VirusTotal. Interestingly, most of the websites that include an indicator URL were not detected as malicious. However, even a single page flagged by VT in the set of hundred results can be a very distinctive factor once combined with the other features. On the other end of the spectrum, the component popularity feature was the one with the highest negative correlation.

With a considerable manual effort, we investigated each single case to understand if our system was correct with its classification and to look for possible false positives. As we better discuss in the next section along with a number of examples and case studies, we only found two false positive out of 303 analyzed URLs.

The first false positive is a very popular library provided by Google and used as external resource by many websites (including some defaced and some malicious ones). Unfortunately, some of these websites were duplicated in different domains (therefore with exactly the same content) and this caused an increase in the similarity rate which, inevitably, results in a false positive. The other false positive is a JavaScript file used for video and animated online ads (AdInterax). Although there were no results on Meanpath for this URL, it was often discussed on security forums by users who were afraid it was a malicious component.

Except for these two isolated cases, we were able to confirm that the web indicators of compromise extracted by our tool were indeed constantly associated with malicious activities and did not appear in benign pages.

Almost 20% of those indicators were URLs of JavaScript component that we never observed before in our honeypot. Interestingly, the remaining 80% were instead components that were observed several times by our honeypot during the previous years. In average, these indicators of compromise were first observed 20 months before our experiment, with a maximum of 44 and a minimum of 5 months. Figure 3.4 shows the difference between the first seen date and the last seen date for each valid indicators of compromise identified by our tool. The graph shows that the average lifetime of these JavaScript components is very high. This is likely a consequence of the fact that the majority of these scripts are not malicious per se, and therefore the identification of these components is very difficult by using existing techniques. As a result, some stay unnoticed for months or even years. Figure 3.5 shows instead the total number of times each indicators was used in attacks against our honeypot.

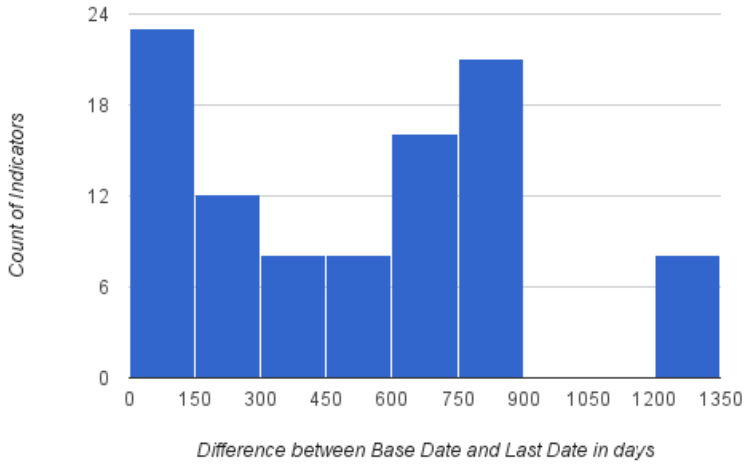

Figure 3.4: Differences in days between the first seen date and the last seen date of the Valid Indicators of Compromise

Category	Number of Items
Invalid Indicator	22
Valid Indicator of Compromise	96
Not-enough-data	185

Table 3.1: Clustering results for the detection set

3.3.4 Antivirus Telemetry

To assess the impact of our indicators of web compromise in a real deployment, we asked Trend Micro to cross-check our results. For this purpose, we sent them the set of our indicators and asked them to match the URLs against their web telemetry dataset collected in the same time period.

Overall, over 90% of our web indicators were previously unknown to the vendor and were considered benign by their internal intelligence database. Moreover, the vast majority of pages that included those components were not detected as infected by any AV tool used by VirusTotal. In total, only 5.3% of the webpages including an indicator were detected by at least for one antivirus products. Once more, this confirms our initial hypothesis that existing automated scanners only flag a page when there is an clear evidence of malicious activity and fail to detect more subtle signs of a possible compromise.

Interestingly, some of our indicators were hosted on domains for which Trend Micro observed only hits toward the URL of that particular Javascript and nothing

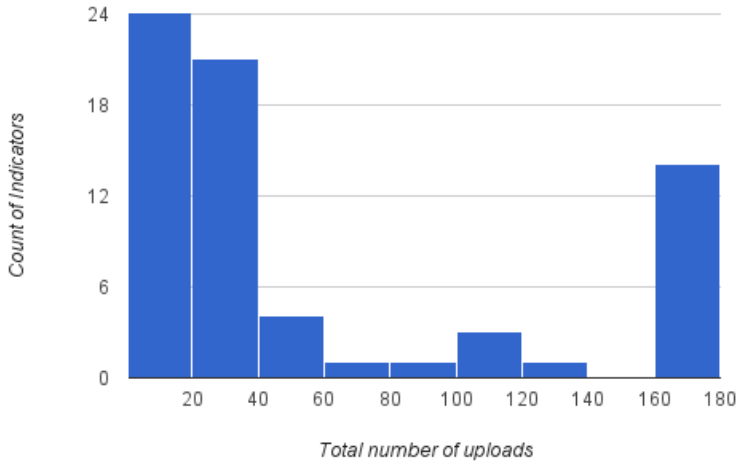

Figure 3.5: Total number of uploads of WIOCs

else in their telemetry dataset, as if that component was the only item hosted on the same domain.

In average, each indicators was requested by 47 different users per day. However, the distribution was quite unbalanced, with one case that was never observed in the wild and one case that had more than 800 visits per day.

However, we believe that an automated system that could prevent each day thousands of users from visiting malicious or compromised websites – not caught by any other detection systems and by the blacklists already in use at the antivirus vendor – is a promising result that shows the value of our idea and the need for more research in the area of web indicators of compromise.

3.4 Case Studies

The 96 indicators extracted over four months by our system belong to different categories. For instance, 26% of the JavaScript components were used to implement simple visual effects (such as moving or teletype text, or a snow effect over the page) commonly used in defacement campaigns. Other were used as part of phishing pages, or to redirect the visitors towards other websites.

In this section we discuss in more details some of these cases, to try to show different scenarios and different types of malicious activity detected by our system.

Affiliate Programs

Attackers often try to monetize the traffic towards web sites they compromise, for example by joining affiliate programs or including Adwares banners. In our evaluation, we identified several cases of JavaScript dedicated to this purpose.

For example, one of the indicators found by our system[1] is part of a large affiliate network called VisAdd [Vis]. The script acts as a traffic redirection system – i.e., a sort of proxy that brings the users of the exploited web page to the affiliate web site. In this way, the miscreant get rewarded for each visitor of the site she was able to compromise. Interestingly, VisAdd also makes use of a malicious software called `A.Visadd.com`[2] to bring additional visitots into its network. By correlating this indicators with Trend Micro's web telemetry dataset, we confirmed that an average of 620 users per day were affected by sites including this JavaScript. In another example, a malicious browser plugin – in form of an Internet Explorer Browser Helper Objects (BHOs) was loaded by a JavaScript file at run-time in order to hijacking the browser's user's session. We observed the same Javascript embedded in a multitude of defaced web sites.

In both cases, it is interesting to observe that cyber criminals used a combination of client-side approaches – like malware and BHOs – and server-side compromised websites to redirect legitimate traffic to affiliate programs. We recorded an average of 594 visits per day to this indicator.

Since these JavaScript files were quite popular in the antivirus dataset, it would be possible to use them to track the activity and evolution of this large campaign, whose list of compromised websites is still increasing at the time this work was submitted.

Web Shells

A second class of indicators that we automatically identified as malicious are related to web shells, which are often deployed by the attackers and hidden in compromised web sites. Their goal is to allow the attackers to easily control the compromised machine and execute arbitrary commands by using an intuitive web interface. We found several cases in which, despite the attacker protected the access to the web shell via password, our system was able to automatically flag these cases because they embedded a malicious indicator.

As already described in the example of Section 3.2, we also discovered a small JavaScript responsible to send a signal back to the attackers every time someone visited one of the installed web shells. Some of these (e.g. `http://r57shell.net/404/ittir.js`) automatically leak the information of the visit to the attacker's dropzone, e.g. by embedding the request in the form of a image retrieval

[1] `http://4x3zy4ql-l8bu4n1j.netdna-ssl.com/res/helper.min.js`
[2] `http://malwaretips.com/blogs/a-visadd-com-virus-removal/`

– in a technique similar to a CSRF. The inclusion of this URL on compromised websites is a clear indicator of an organized network in which the attackers monitor the websites they infected as well as the ones infected by other groups that reuse the same web shells.

Code Repositories

In our dataset, we found a considerable amount of indicators of compromise hosted in public code repositories, such as Google Code. Even though it is not unexpected for attackers to use such repositories, it was surprising to observe how long these indicators can survive before they get noticed and taken down by the maintainer or the security community.

For example, we found two indicators hosted on Google Drive and eight on Google Code. Interestingly, one of them was online for at least 729 consecutive days before it was finally suspended by Google and just in a single month MeanPath reported dozens of defaced websites and drive-by pages using this script.

During the manual verification of this case, we realized that most of the web pages that include WIOCs look almost identical. Furthermore, even a quick search on Google returned many forums in which people complained about how their website got hacked as well as scan results from popular sandboxes. This case confirms that our features provides an accurate characterization of indicators.

Mailers

Another use of compromised websites is to turn them into a spam mailing server to send large amounts of fraudulent emails. Instead of switching between different providers or relying on botnet-infected machines, attackers often search for non-blacklisted vulnerable websites to use as part of their infrastructure.

In our experiments, our system reported two indicators of compromise corresponding to two copies of the *JQuery* library, hosted respectively on Google and Tumblr. The majority of websites using these libraries (e.g., `http://www.senzadistanza.it/` and `http://www.hprgroup.biz/`) contained pages injected with a popular mailer called Pro Mailer v2, which is often shared among hackers in underground forums. Because of the popular domains used by these indicators, and because of the fact that they were unmodified copy of popular libraries, these files very likely misclassified as benign by both automated scanners and manual inspection. Therefore, we believe this particular example is very important, since it emphasize the fact that even the most harmless and legitimate URLs can be valid indicators of compromise.

Phishing

Phishing pages are commonly found in our dataset, as attackers try to install copy of popular websites in our honeypot after they compromise one of our web applications. As a last example, we want to discuss two borderline cases we found in our results.

In these cases, the attackers installed phishing pages for the Webmail portal of two popular websites, AOL and Yahoo. Instead of simply uploading the entire content of the site on our honeypot, they re-used the original AOL and Yahoo JavaScript files hosted on their respective provider's domain. Since the components were clearly also used by benign websites, these URLs were misconceived as benign and classified as false positive during manual verification. However, a quick search for both examples returned many websites including these scripts that were clearly not related to AOL or Yahoo (e.g., http://www.ucylojistik.com/ for AOL and http://fernandanunes.com/ for Yahoo), and that turned out to be all compromised to host phishing pages.

We decided to discuss this case as it demonstrates how a benign URL can be used to leverage phishing pages. Even though both URLs also serve for benign purposes, they are also excellent indicators of compromise when they are observed on web sites registered outside of their original domain or autonomous system. In other words, any time users requested these JavaScript files while visiting a page that was not on the AOL/Yahoo domain, then they were victims of phishing. However, since these components are also used by their legitimate service, we did not count their hits in the AV dataset in our report.

3.5 Limitations

Since our method relies on the fact that attackers remotely include resources in their pages, it is possible to evade our technique by using a different deployment strategy. For example, attackers could include their code inline rather than importing the indicator's URL from an external source, or they could generate a different URL for each target. Even though these techniques would effectively undermine our ability to extract valid indicators of compromise, these changes would also result in a loss of flexibility (e.g., the attacker would not be able to change at once the code used in hundreds of compromised pages) or in an increased complexity in the deployment of the code.

In our current implementation, our system relies on a clustering phase to separate the good from the bad indicators. While we did not need to repeat this training during our experiments, it may be helpful to update the clustering at least once a year – to account for possible changes in the features distribution. For example, it is possible that security forums become more popular in the future, or that the results returned by Meanpath increase (or decrease) over time.

Finally, while this work is the first to introduce the concept of web indicators of compromise, we expect more researchers to focus on this interesting problem and to propose more sophisticated and more robust solutions to extract WIOCs in the future.

3.6 Conclusions

In this chapter we presented a novel technique to use the information collected by a high interaction honeypot of vulnerable web applications. Our approach starts from the observation that attackers often include remote JavaScript components in the pages they modify or they upload after a successful attack. These components are rarely malicious per se, but their URLs can still be used to precisely pinpoint the activity of a certain group and therefore the fact that a web page has been compromised. For this reason, in this chapter we proposed a technique to collect these components, validate them using a number of features, and finally use them as *Web Indicators of Compromise* (WIOCs).

We implemented our system and run it on our premises for several months. After an unsupervised training phase, we tested for four months its ability to automatically extract valid WIOCs. The results showed that these indicators cover several types of malicious activities, from phishing sites to defacements, from web shells to affiliate programs. Moreover, most of these components have been used for a long time by the attackers, who hosted them on public websites – since their apparently harmless content was not detected as suspicious by any of the existing tools and techniques.

We believe that more research is needed in this area, to help the security community to reliably extract and share this new type of indicators of compromise.

Chapter 4

Attack Landscape in Dark Web

**This chapter is based on a paper presented at the 32nd Annual ACM Symposium on Applied Computing (SAC) in 2017 [CBB17], where it won the best paper award for the security track.*

Based on the accessibility of its pages, the Web can be divided in three parts: the *Surface* Web – which covers everything that can be located through a search engine; the *Deep* Web – which contains the pages that are not reached by search engine crawlers (for example because they require a registration); and the more recent *Dark* Web – which is dedicated to websites that are operated over a different infrastructure to guarantee their anonymity, and that often require specific software to be accessed.

The most famous "neighborhood" of the Dark Web is operated over the Tor network, whose protocols guarantee anonymity and privacy of both peers in a communication, making users and operators of (hidden) services in the Dark Web more resilient to identification and monitoring.

As such, over the last years, miscreants and dealers in general have started to adopt the Dark Web as a valid platform to conduct their activities, including trading of illegal goods in marketplaces, money laundering, and assassination [CBMR]. Moreover, Tor has been reported to be leveraged in hosting malware [O'N], and operating resilient botnets [Bro10]. While, to a certain extent, these studies have shown how the Dark Web is used to conduct such activities, it is still unclear if and how miscreants are explicitly conducting attacks against hidden services, like a web application running within the Tor network.

In the previous Chapter we described how we can use a web honeypots deployed on the *Surface* Web to automatically discover other compromised websites. The study was based on a number of observations on the behavior of web attackers,

Area	Impact	Better in
Attack Identification	Results	Surface Web
Service Advertisement	Operation	Surface Web
Stealthiness	Deployment	Dark Web
Operational Costs	Deployment	Dark Web
Collected Data	Operation	Surface Web

Table 4.1: Advantages and Disadvantages of Operating a High-Interaction honeypot in the Dark Web

which however may be specific to the common Web. In fact, while web attacks, or attacks against exposed services on the Internet, are common knowledge and have been largely studied by the research community [CB13a, CBB16, SDA+16b], no previous work has been conducted to investigate the volume and nature of attacks in the Dark Web.

To this extend, in this Chapter we discuss the deployment of a high-interaction honeypot within the Dark Web to collect evidence of attacks against its services. In particular, we focus our study on web applications to try to identify how attackers exploit them (without a search engine for localization) and what their purpose is after a service has been compromised. Our preliminary measurement casts some light on the attackers' behavior and shows some interesting phenomena, including the fact that the vast majority of incoming attacks are unintentional (in the sense that they were not targeted against Dark Web services) *scattered attacks* performed by automated scripts that reach the application from the Surface Web through Tor2web proxies.

4.1 Honeypot in the Dark web

In this section, we discuss the main differences, in terms of advantages and disadvantages, between deploying and maintaining a honeypot in the Surface Web versus operating a similar infrastructure in the Tor network.

In fact, the anonymity provided by Tor introduces a number of important differences. Some are positives, and make the infrastructure easier to maintain for researchers. Some are instead negative, and introduce new challenges in the honeypot setup and in the analysis of the collected data.

Table 4.1 summarizes the five main differences between the two environments, mentioning their impact (on the deployment, operation, or on the results collected by the honeypot) and which environment (Dark or Surface Web) provides better advantages in each category.

Attack identification

The most significant difference between a deployment on the surface Web and on the Tor network is the anonymity of the incoming requests. In a traditional honeypot, individual requests are typically grouped together in *attack sessions* [CB13a] to provide an enriched view on the number and nature of each attack. A single session can span several minutes and include hundreds of different requests (e.g., to probe the application, exploit a vulnerability, and install post-exploitation scripts).

Since many malicious tools do not honor server-side cookies, this clustering phase is often performed by combining two pieces of information: the timestamp of each request, and its source IP address. Thus, requests coming in the same empirically-defined time window and from the same host are normally grouped in a single session.

Unfortunately, the source of each connection is hidden in the Tor network, and therefore the identification of individual attacks becomes much harder in the Dark Web. Moreover, if the attacker uses the Tor browser, also the HTTP headers would be identical between different attackers.

Stealthiness

If on the one hand the anonymity provided by the Tor network complicates the analysis of the attacks, on the other it also simplifies the setup of the honeypot infrastructure. In fact, a core aspect of any honeypot is its ability to remain *hidden* as the quality of the collected data decreases if attackers can easily identify that the target machine is likely a trap.

For instance, the nature of the Surface Web reveals information like the `Whois` and `SOA` records associated with a domain name, or the geo-location of the IP address the honeypot resolves to. To mitigate this risk, Canali et al. [CB13a] employed a distributed architecture including hundreds of transparent proxy-servers that redirected the incoming traffic via VPN to the honeypots hosted on the researchers' lab. This solution successfully solve the problem of hiding the real location of the web applications, but it is difficult to maintain and requires the proxies to be located on many different networks (often on online providers).

Luckily, this problem does not exist on the Dark Web. The honeypot can run anywhere, without additional expedients as the Tor network is sufficient to guarantee the anonymity of the endpoints. Moreover, if a particular domain is blacklisted by the attackers, it is sufficient to generate a new private key/hostname pair to host content under a new domain name.

Service advertisement

As the most important value of a honeypot is the collected data, it is essential to attract a large number of attackers. On the Surface Web, it is typically the role of search engines to make the honeypot pages visible to the attackers interested in a certain type of target. For instance, honeypots often employ vulnerable versions of popular CMSs, as attackers routinely look for them by using Google Dorks [ZNG14].

It is also possible for a website on the Surface Web to attract attackers by simply placing some keywords or specific files as John et al. described in their work [JYX+11]. For example, including a known web shell or disclosing the vulnerable version of an installed application along with its name is a widely used strategy to lure attackers.

These popular "advertisement" approaches are not straightforward to apply to services hosted on the Tor network. As we later discuss in Section 4.3, it is still possible for `.onion` web sites to be indexed by Google. However, in order to gain popularity and attract attackers, researchers should carefully employ alternative techniques – such as advertising the website in forums, channels, or link directories specific to the Dark Web.

Operational costs

Since, as explained above, operating a honeypot in the Dark Web does not require any special domain registration or dedicated hosting provider, the total cost of the operation is typically very low. Canali et al [CB13a] had to register hundreds of domain names (and routinely change them to avoid blacklisting) as well as several dedicated hosting providers – which are often difficult to handle because they often block the accounts if they receive complains about possibly malicious traffic.

In comparison, an equivalent infrastructure on the Dark Web only requires the physical machines where the honeypot is installed, as creating new domains is free and can be performed arbitrarily by the honeypot administrator.

Nature of the collected data

Some criminals use the Tor network to host illicit content like child pornography, since it protects both the visitors and the host by concealing their identities. Therefore, as we explain in Section 4.2, we had to take some special precautions to prevent attackers from using our honeypot to store and distribute this material. Unless researchers work in collaboration with law enforcement, these measures are required to safely operate a honeypot in the Dark Web.

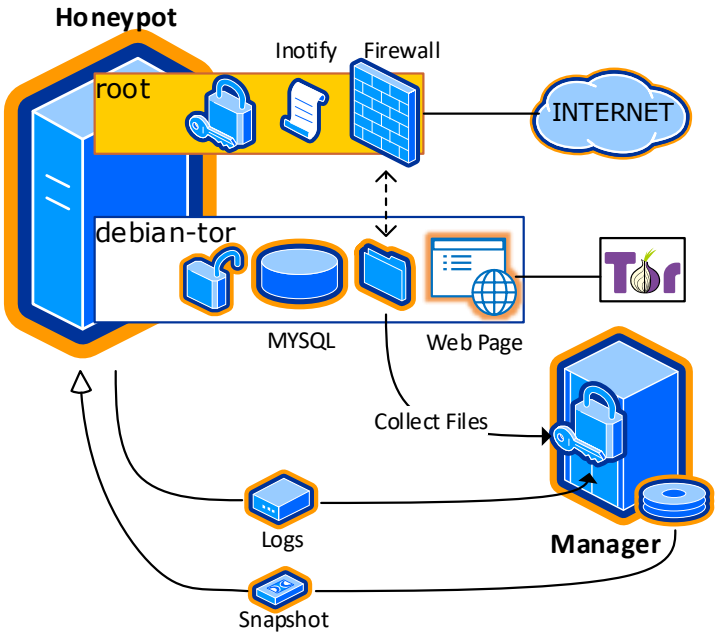

Figure 4.1: Simplified Honeypot Infrastructure

4.2 Honeypot Setup and Deployment

In this section we describe the setup of our honeypot. Our deployment is composed of three types of web-based honeypots and a system-based honeypot. Each of them is installed in a separate virtual machine (VM) hosted on our premises. The use of virtual machines allow us to revert the honeypots to a clean state after they are compromised. All honeypots are connected to the Tor network and made available as hidden services.

Note that each VM was fully patched to prevent privilege escalation, i.e. an attacker who compromised any of our machines would not be able to modify any system file and could only interact with the content of few selected directories[1].

Moreover, we used a set of firewall rules to restrict the attackers' network capabilities. In particular, we blocked all incoming and outgoing connections from all ports, except the ones used by Tor to operate, and ports associated to services that we explicitly offered. The firewall was also configured to enforce strict rate-limits to prevent denial-of-service attacks.

[1]Excluding attacks leveraging 0-days and undisclosed vulnerabilities

Web Applications

To mimic the setup used by a casual user, we decided to install all the applications in their default configuration, e.g. with all files located under `/var/www` and owned by the user `debian-tor`.

In each honeypot we installed `ModSecurity` [mod], a popular monitoring and logging tool for the Apache web server. We configured ModSecurity to log the content of all HTTP POST requests along with their headers.

We also used a real-time file system event monitoring framework called `inotify` to detect all newly created/modified files, and copy them in a private directory for later inspection. Most importantly, using `inotify` we promptly detected, deleted, and shred any multimedia file uploaded by an attacker – to prevent our servers from hosting illegal material.

After we completed the configuration of our honeypots, we took a VM snapshot of their clean state. Later, every night, our system was configured to automatically retrieve all the files collected by `inotify` and a copy of all log files, and then to revert each VM to its original snapshot. A simplified representation of our honeypot infrastructure is given in Figure 4.1.

In order to bait the attackers, we decided to deploy three different honeypot templates:

1. **A website disguised as an exclusive drug marketplace that only trades between a close circle of invited members** – The website was realized using an old version of the popular OSCommerce e-marketing application. The version used in our experiments contains several known vulnerabilities, which allow an attacker to take over the admin panel and arbitrarily manipulate accounts and files.

2. **A blog site that advertises customized Internet solutions for hosting in the Tor network** – The website was realized using an old version of WordPress, which contained several known vulnerabilities.

   The honeypot also contained a number of sub-directories with different web shells, in order to mimic the fact that the site was already compromised by other attackers. The website was misconfigured to allow directory listing, so that an attacker (or an automated script) could easily navigate through the structure of the website and locate the shells.

3. **A custom private forum that only allowed registered members to login** – The website described the procedure to become a member, which required a valid reference from another existing member. In this case, we manually included a custom remote file-inclusion vulnerability that allowed an attacker to upload arbitrary files by manipulating PHP filters. The vulnerability was designed to be quite "standard", mimicking many existing

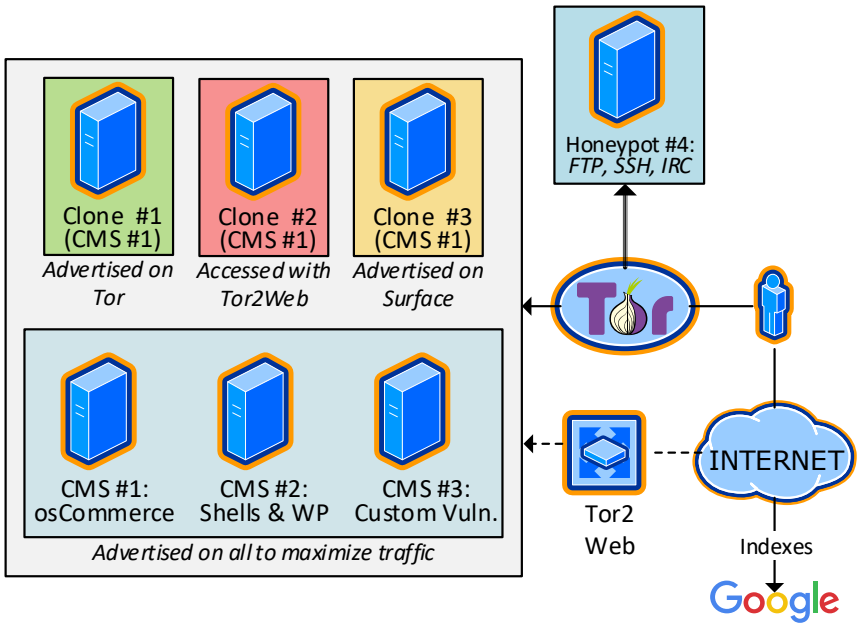

Figure 4.2: Initial deployment of honeypots with different advertisement strategies

vulnerabilities of the same type reported in other applications. However, it was also designed not to be straightforward to identify for automated scanners, as the goal of this third honeypot service was to collect information about manual attacks.

Although slightly tailored for our scenario in terms of advertised content, the first two templates were also used by a previous study of web honeypot [CB13a]. These were intentionally chosen to be able to compare the types of attacks received on the Dark Web with those normally observed in the Surface Web.

The third template was instead specifically designed to avoid automated scanners and study more sophisticated attackers who may be interested in manually exploiting services hosted in the Dark Web.

We started by advertising our honeypot applications in three different ways: (i) by posting their URLs in several Tor network's forums, channels, search engines and yellow pages, (ii) by visiting (twice a day) the applications via the Tor2Web proxy – which shares the visited URLs with Ahmia [ahm], a search engine for Tor, and (iii) by posting their URLs to several pages on the Surface Web.

In particular, to measure the success of our approaches, we deployed the first template three times (i.e., one for each advertisement technique). On top of that, we also deployed a copy of all templates by using a more aggressive strategy that includes all the three mechanisms described above.

Other Services

We also decided to include in our system a machine dedicated to collect system-level attacks directed towards other type of services appropriately configured to facilitate reconnaissances from attackers (e.g., by leaking the list of users via finger) or to expose weaknesses or misconfiguration.

This machine, reachable only over the Tor network, ran the following services:

1. We used `finger` to broadcast the list of active users and we provided a file containing the hashed version of a user's password on an open FTP server. We also used message-of-the-day informative to advertise our honeypot as a file-server.

2. We offered an open (anonymous) FTP server. We served a valid upload directory (`incoming`) for hypothetical illicit uses like drop-zone and exploitation, and we provide some documents for download, one of which contained the password for one of the system users.

3. We enabled SSH login on 2 users. The shell was chroot jail protected. Both accounts were easily guessable, i.e. the first having a straightforward name and password combination (`guest:guest`); the second having the base64-encoded version of the password leaked in the FTP document.

4. IRC. Chats are known to be used as rendez-vous points to discuss illicit offers (e.g. stolen accounts) or host C&C servers of botnets. With the intent of understanding whether attackers would try to abuse chats in the Tor network, we installed an open IRC service (`UnrealIRCd`) and registered anonymous logins.

This machine was also advertised using all the previously described channels (for the Tor2web case, we used the proxy to access a static webpage hosted on the honeypot, describing the machine as a Dark Web file hosting server). Figure 4.2 shows a summary of the initial deployment strategy used in our experiments.

4.3 Data Collection and Analysis

We run our experiments over a period of seven months between February and September 2016. Due to maintenance and re-configuration of the honeypots, the individual honeypot services were online for a total of 205 days.

The experiments were divided in three phases. During the first phase (which lasted for 37 days until the end of March) we applied the three advertisement strategies described in Section 4.2 on a single honeypot template (CMS #1), to

	Clone #1 (Tor Only)	Clone #2 (Tor2Web)	Clone #3 (Surface Only)
GET	3.29M	1.26M	1.02M
POST	20	147	1

Table 4.2: Number of GET & POST requests for different advertisement strategies

measure their impact on the incoming traffic and on the number of attacks. In the second phase (from the 1st of April to end of May) we advertised the three templates using all available strategies, to maximize the amount of collected data. Finally, for the last four months of experiments, we restricted the access to our honeypot by blocking Tor2web proxies, in order to exclusively focus on attacks within the Tor network.

In the rest of the section we describe the impact of these three factors on the collected data: the advertisement strategy, the source of the attack (from the Surface or the Dark Web), and the type of honeypot template.

4.3.1 Impact of Advertisement Strategies

As we mentioned in Section 4.2, we created three clones of our first honeypot template (CMS #1), which we then advertised using different channels.

In Table 4.2, we present the total number of requests received by the three clones. Quite interestingly, all clones received a comparable amount of overall traffic (between 1 and 3.3M hits). However, looking at the POST requests the picture is quite different. For instance, the honeypot advertised on Tor only received over 3M GET requests but only 20 POSTs. The first number is inflated because the same visitor may have requested multiple resources – and we already discussed how difficult it is to track visitors in the Dark Web, when using the same browser and no endpoint information are available. In addition, since attackers required a POST request eventually to upload their files, we decided that looking at POST requests was a better way to estimate the "interesting" traffic, and filter out most of the harmless visitors, automated Internet scanners, and other forms of background noise.

Finally, it is interesting to note how the second clone – advertised through Tor2web, was the only one to receive attacks (over 20) in this first phase of our experiments.

4.3.2 Role of Tor Proxies

The Tor2web[2] projects provide a simple way for users to access resources on the Dark Web by simply appending special extensions to onion domains. These

[2] https://www.tor2web.org/

Proxy	Online	Transparent
**.onion.to*	✓	✗
**.onion.link*	✓	✗
**.onion.city*	✓	✗
**.onion.nu*	✓	✓
**.onion.cab*	✓	✓
**.onion.direct*	✗[1]	*Unknown*
**.onion.lt*	✗[2]	*Unknown*
**.onion.sh*	✗	*Unknown*
**.onion.ink*	✗	*Unknown*
**.tor2web.org*	✗	✗
**.tor2web.fi*	✗[3]	✗
**.onion.rip*	✓	✗

[1] discontinued
[2] website is offline
[3] redirects to tor2web.org

Table 4.3: List of inspected Tor proxies

special domains (such as `.onion.to`, `.onion.link`, and `.onion.city`) resolve to one of the Tor2web operators which in turn act as proxies from the Surface Web to the Dark Web. These services facilitate the access to the Tor network with the disadvantage of sacrificing the anonymity of their users.

Since Tor proxies make hidden services in Tor reachable with a normal HTTP request over the Internet and with *no* additional configuration, they can be used by traditional browsers but also by automated scripts and crawlers. The presence of these proxies turned out to be extremely important for our experiments. In fact, once a proxy domain is indexed by a search engine, the target website can be located using traditional Google Dorks [TACB16] and therefore becomes implicitly a target of automated exploitation scripts [CB13a].

Once we noticed this phenomenon and the fact that the vast majority of the attacks indeed came through these proxy services, we decided to block the request coming from Tor2web.

Table 4.3 shows a list of different operators, mentioning those that were online during our experiments and those that we could identify by looking at the HTTP headers they append. For example, some of the Tor2web operators like `.onion.to`, `.onion.link`, and `.onion.city` includes extra headers in the request field (such as `HTTP_X_TOR2WEB`, `HTTP_X_FORWARDED_PROTO` and `HTTP_X_FORWARDED_HOST`). This allowed us to identify and block the requests coming from these services, by serving them a static page explaining that our services were only available from the Tor network.

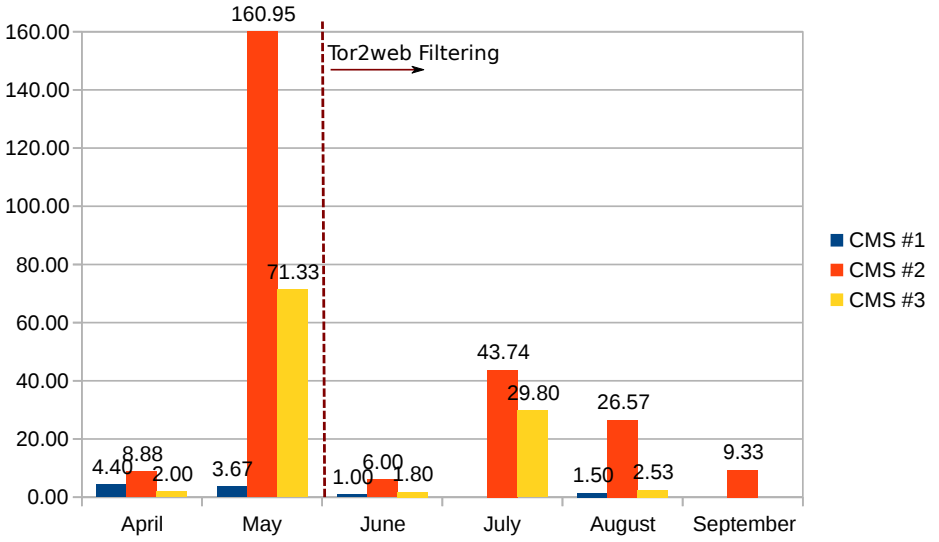

Figure 4.3: Average number of daily post requests

	CMS #1 (OsCommerce)	**CMS #2** (Shells & WordPress)	**CMS #3** (Custom Vuln.)
Tor2web	115 (8 days)	1,930 (23 days)	0
TOR	0	2,146 (79 days)	689 (5 days)

Table 4.4: Number of attack-related POST requests

This change – implemented from June in our experiments, lead to a sharp drop in the number of incoming requests and in the number of incoming attacks (see Figure 4.3).

Interestingly, blocking Tor2web proxies also had a clear effect on the *type* of files uploaded by the attackers. For example, we stopped receiving phishing kits or mailers (see Section 4.4 for more details) after we implemented our blocking strategy. Therefore, it is safe to say that even though it was possible that some requests still came through transparent proxies, our countermeasure was able to effectively prevent most of the automated attacks specific to the Surface Web– which, indeed, was our initial goal.

4.3.3 Honeypot Templates

As we explained in Section 4.2, we used three honeypot templates based on different web applications and, more importantly, with different types of vulnerabilities.

Table 4.4 shows the number of attack-related POST requests along with the number of days in which we received at least a single attack. We consider a request as attack-related if it contains an attempt to exploit a vulnerability or if it is involved in the post-exploitation phase – for example, to upload further files on the exploited application or to inspect the host through a webshell.

Predictably, the web shells installed on CMS #2 served as bait to attract the highest number of attack-related requests. CMS #1, instead, received less attacks and all originating through proxy services likely as the result of using Google Dorks.

Interestingly, some attackers used dorks on Tor-specific search engines. For instance, an attacker found one of the honeypots through a Tor search engine[3] by simply querying the set of keywords *"Index of /files/images/"*.

Finally, CMS #3 (i.e., the one containing our custom vulnerability) received the lowest number of attacks and none of them actually succeeded. Even more interestingly, while CMS #1 was only attacked from the Surface, CMS #3 was only attacked from the Tor network. This is in line with our expectations: well-known vulnerable CMSs are targeted by automated scripts using dorks, while custom websites are only targeted by humans or dedicated scanners (which in our case were run in the Tor network).

Afterward, we manually analyzed all POST requests to identify the successful attacks, i.e. those in which the attacker successfully compromised the honeypot application. CMS #1 was successfully compromised by attacks originating from the Surface Web. Out of 115 attack-related requests, 105 (91%) of them were successful attacks. After we started blocking Tor proxies, we did not observe further attacks.

For CMS #2, we counted an attack as successful if it exploited the WordPress vulnerability or if it used one of the existing webshell to install or modify a file in the system (i.e., a simple inspection of the system was not considered a successful attack). Overall, 1,255 (65.0%) of the attack-related requests originating through Tor2web succeeded. On the contrary, only 154 (7.2%) of the ones coming through the TOR network succeeded. This is a very interesting result, and it shows that the majority of the attackers who interacted with our shells from the Surface Web ended up performing some change on the system. The attackers from the Tor network instead mostly inspected the system and moved away without touching any file (more about this phenomenon will be discussed in Section 4.4).

CMS #3 never received a single attack from Tor2web, if we exclude some manual attempts to guess a valid username and password (which we did not count as an attack in our statistics). As we mentioned in the beginning of this section, this one never received a successful attack.

[3] http://hss3uro2hsxfogfq.onion

4.4 Attack Examples

Over the entire experiment, attackers uploaded on our honeypot 287 files (an average of 1.4 per day). In comparison, Canali et al. [CB13a] collected over 850 files per day – but they used 500 clones (against the 3 we used in our experiments). In fact, the goal of our study was not to collect a large number of attacks, but rather to study their nature and how the effect of different factors like the advertisement channel, the type of service, and the source of the attack affected.

In the following sections, we classify the attacks into three different categories: automated scattered attacks from the Surface Web, automated attacks from the Tor network, and manual attacks. For each category, we present some examples and we discuss in more details the attacks most commonly observed in our honeypots.

4.4.1 Scattered attacks

As we described previously, regular search engines unexpectedly index web pages hosted on the Dark Web through Tor2web proxies. As a result, websites located in the Tor network receive part of the background noise of automated attacks that plague the Web, scattered through the proxies that act as gateways between the two "sides" of the Web.

For this reason, the vast majority of the attacks observed in our honeypot were simple, and very similar to what was observed by previous studies [CB13a, JYX+11]. Basically, the modus operandi of these attackers consisted of locating our websites using Google Dorks, and employing automated scripts to visit the pages, exploiting the known vulnerabilities, and possibly uploading files for the next phase of the attack. In the majority of the cases, these attacks involved the use of web shells, which allowed the attackers to later run system commands on our honeypot. Using these web shells, attackers could upload other files including web mailers, defacement pages, and phishing kits.

The completely automated nature of these attacks and the types of files uploaded in the honeypot make us believe that in the majority of the cases the attackers were not even aware of the fact that they compromised applications hosted in the Dark Web.

Web shells – We collected 157 unique variations of web shells uploaded by the attackers. This was an expected behavior since most of the time the attackers made use of automated scripts for the first phase of the attack. We also observed that once a web shell was deployed, other shells were often uploaded using this first web shell, over a short period of time. Usually, while the initial shell was unobfuscated, the subsequent ones were protected with a password. Some of

the collected web shells were base64-encoded and they were configured to de-obfuscate at run time by means of the PHP's *eval* function.

Phishing kit & Mailers – Surprisingly, attackers uploaded six phishing kits for popular targets (in particular Paypal). Having a phishing kit for such applications does not make much sense in our setting, since there is no Paypal on Tor to begin with. But the fact that all the phishing-related attacks were coming through TOR proxies, strengthen our hypothesis that the attackers (or their automated scripts) were probably not aware of the location of the exploited application. Similarly, 22 mailers were uploaded through Tor proxies, but none of them was ever used by the attackers.

Defacements – Our web applications got defaced 33 times. Usually, a web shell was uploaded before the defacement and subsequently the index page was modified or a brand new one was uploaded by using the prior web shell. From an analysis, this process looked automated since the same pattern was observed multiple times with the same defacement page. Half of the attacks originated from the Surface Web, and the rest came directly from the Tor network.

In one defacement specific to Dark Web, the defacer modified the index page of CMS#2 to promote one of his sites called Infamous Security[4], where the authors apparently advertise their hacking services.

4.4.2 Automated Attacks through Tor

Automated Scans – Our honeypots received over 1,500 path traversal attempts (e.g. to fetch `../../etc/passwd`, or `../../etc/vmware/hostd/vmInventory.xml`). As we could infer from the User Agent, attackers seemed to be using the NMap[5] scripting engine for scanning their targets.

Access to the Service Private Keys – One of the most common scan attempt we received within the Tor network was the download of the *private key* that we voluntarily hosted on the web applications' root directory.

Every time the Tor service starts, it creates a private key (if not existing) and assigns the corresponding hidden-service descriptor (i.e. the hostname) to this private key. While the private key must *not* be accessible with default Tor and Apache configurations, in our case we intentionally misconfigured the service to let it accessible from the Web for CMS #3 starting from mid Agust. Exposing a private key simply means that the owner risks losing the hostname to the adversary (and therefore potentially all incoming traffic). Thus, the first and the easiest automated attack is to fetch this key, if its location and permission are not configured correctly.

[4] http://5eaumbq2k6yc4sjx.onion/
[5] http://nmap.org

During the operation of our honeypot, we observed and confirmed over 400 attemps to fetch the private key. Attackers could use those keys to impersonate our honeypot and conduct attacks like phishing or hosting of malware.

Other Services – We reported a number of successful connections to our FTP, SSH, and IRC services that, most likely, represent instances of banner grabbing or information gatherings. In total, we confirmed 74 SSH connections (client-side terminated or timeout), 61 successful FTP (anonymous) logins and 91 IRC logins.

4.4.3 Manual attacks

Post-Exploitation Actions – We noticed that attackers connecting via Tor network (instead of using Tor proxies) were generally more careful and spent more time to investigate the environment. For instance, their first action when using a web shell was to gather additional knowledge by listing directories, checking the content of the local database, fetching `phpinfo` and system files such as `crontab`, `passwd`, `fstab` and `pam.conf`.

Such attackers never went beyond exploring the system, compared to the ones we mentioned in Section 4.4.1 – who almost always installed additional components. In fact, manual attackers from the Tor network often deleted their files and left the honeypot after their initial inspection. In few cases, the attackers also left messages (such as "Welcome to the honeypot!") or redirected our index page to a pornographic video. In one example, the attacker downloaded 1GB of random data from a popular website to test the network download speed and renamed the file as 'childporn.zip' – supporting the fact that many attacks from Tor were manually operated and resulted in a successful identification of the honeypot.

While these cases support the fact that there are people manually exploiting websites on the Dark Web, all these attacks used previously installed web shells or extremely popular CMS vulnerabilities. None of them was able to exploit the (still relatively simple) custom vulnerability on CMS #3.

FTP and SSH – Overall, we identified 71 FTP file downloads. Interestingly, all occurred in a sub-directory and *none* on the root directory of the server – showing the manual nature of the action and the interest in accessing specific data. In one case, the miscreant used our bait login credentials included in our honey-document to log in to the SSH server. This was an interesting scenario, in which the attacker was able to manually extract information collected from one service to connect to another service.

Even more interestingly, the attacker first connected to the SSH server sending his real username for the login, likely due to the fact that this is performed automatically by ssh clients. The attacker then immediately killed the session and reconnected with the correct username previously gathered from the honey-document.

Attacks Against the Custom Application – The application with the custom vulnerability received little attention through the entire experiment. Except for some automated background noise of SQL injections and directory traversal attempts, we noticed 87 requests (GET and POST) that attempted to tamper with the parameter vulnerable to remote file inclusion, but without any success. One attacker analyzed the entire website using the Acunetix [acu] web vulnerability scanner but the tool was unable to exploit the vulnerability. Another attacker focused on the login form and run the sqlmap[6] tool to try to detect a possible SQL injection, again without success.

4.5 Conclusions

This study discusses the deployment of a high-interaction honeypot in the Tor network, to explore the modus operandi of attackers in the Dark Web. We conducted our experiments in three different phases over a period of seven months and we assessed the effectiveness of advertisement strategies on the number and nature of the attacks. Our preliminary results show that also hidden services can receive automated attacks from the Surface Web with the help of Tor proxies. Moreover, we found that miscreants in the Dark Web tend to involve more manual activity, rather than relying only on automated bots as we initially expected. We hope that our work will raise awareness in the community of operators of hidden services.

[6] http://sqlmap.org

Part II

Dynamic Analysis of Server-Side Malicious Code

To carry out the experiments presented in the previous Chapters, we performed a considerable amount of manual attack analysis. This was required for a variety of reasons, including building a ground truth, detailing different attack vector, measuring the impact of a compromise, and better understanding the attackers behavior. While it was necessary, it was a very time-consuming task and any future attempts to study attacks against web application will inevitably face the same difficulties. This also poses obvious limits to the scalability of server-side analysis – as static approaches such as investigating the log files or identifying the attack-related components may become impractical in presence of large datasets.

Therefore, in this second part we present a solution to automate the analysis of previously-collected data (e.g., from web application honeypots) and discuss the effectiveness of adopting a dynamic approach inspired by the sandbox used today to process traditional malware samples.

Chapter 5

Automatic Analysis of Web Attacks using a PHP Sandbox

**This chapter is based on a paper which is currently under submission.*

In the first part of this dissertation we have studied the prevalence of web attacks and how the security community can benefit from their analysis. In particular, we emphasized the importance of such analysis in Chapter 3 by automatically detecting compromised websites, while in Chapter 4 we proved that also the hidden parts of the Web are not immune to the background noise of automated attacks. However, these studies also revealed that the analysis of the attackers behavior is a challenging and time-consuming task – which so far required a considerable amount of manual analysis.

In fact, while a large number of studies have focused on client-side (i.e., JavaScript-based) malicious code [GL09, Naz09, EVW11, MSSV09b], malicious server-side code have instead attracted much less attention and its analysis has so far being performed either manually or by using static analysis tools. For example, the previous study on the attackers behavior conducted by Canali et al. [CB13a] relied on static clustering to categorize the files uploaded by the attackers on compromised web applications. Although such approach is able to capture the essence of the attacks, they often overlook the details of how the attack progresses. For instance, attackers may end up uploading a certain tool (e.g., to send spam emails) in a compromised server only after having carefully investigated the machine and having tried other paths without success. Moreover, as we will demonstrate in the following sections, attackers often obfuscate the contents of malicious files beyond what can now be handled by static deobfuscators. This complicated a manual inspection and make it more difficult to determine the purpose of the attack by static analysis.

In this study, we also argue that the analysis of malicious server-side scripts is fundamentally different from the analysis of other forms of malicious code (such as traditional malware binaries). In fact, attackers interact with most of these server-side components through a, often complex, web interface. Therefore, the same tool can be used to perform a great variety of different tasks, depending on where the attacker clicked and which sequence of commands (i.e., HTTP requests) she sent to the aforementioned interface. The nature of these attacks forces security analysts and forensic examiners to reconstruct the individual steps of a compromise by a tedious process of correlating the incoming requests' logs with the de-obfuscated code of the server-side components. The result is a time-consuming and error-prone procedure, that often results in a poor reconstruction of the attack behavior.

To solve this problem, in this work we present the first dynamic approach to analyze malicious server-side web applications. In particular, we introduce a fully-automated dynamic analysis solution to study malicious PHP files, inspired by existing malware analysis sandboxes. The main difference between our system and a malware analysis sandbox is that our solution uses real attack traces extracted from web server log files as input, to drive the malicious application and observe its behavior. This allows an unprecedented view over the actions performed during a web attack, including the extraction of fine-grained information about any external interaction between the attacker's tool and the target environment.

We achieve this by instrumenting the PHP interpreter and further installing it in a virtual environment. We chose PHP as server-side programming language in our prototype since it is used by the large majority of websites [W3T17] and almost the totality of malicious code and web shells. Our sandbox supports HTTP logs that are often collected by server-side monitoring systems, and then it identifies and replays all HTTP requests involved in an attack against our instrumented sandbox. Finally, our system automatically generates a detailed report containing all the details of the monitored attack session. In few seconds, and without any manual interaction, our approach can provide a complete overview on the actual behavior of the adversaries.

In summary, in this chapter we make the following contributions:

- We propose a method to dynamically analyse web attacks' server-side code. To the best of our knowledge, we are the first to perform this type of analysis, which was so far conducted either manually or with the support of static analysis techniques.

- We evaluate our system by replaying more than 8000 attacks collected over two years by two web application honeypots. This is the largest dataset of real attack sessions that have ever been analyzed to date.

- By looking at the behavioral report generated by our system, we present a detailed overview of numerous aspects of web attacks.

5.1 The Role of Dynamic Analysis

Today, the analysis of malicious code largely rely on dynamic analysis techniques. While static binary and code analysis could provide enormous advantages to better understand what a piece of code *can* do, the vast majority of modern examples of malicious code rely on obfuscation and packing techniques that defeat any attempt to statically analyze the sample. Therefore, for the analysis of traditional malware, researchers designed special sandboxes [CWS17] where samples can be executed in an instrumented environment, and the runtime behavior of the sample can be captured and analyzed. This solution is today a standard technique used by security companies to process hundred of thousands binaries each day.

Despite this success, researchers have failed to export the same model to other environments. In particular, the analysis of malicious code on the Web is still largely delegated to manual investigation or to static analysis routines designed to de-obfuscate the code and extract a number of indicators about its possible (benign or malicious) behavior.

The Impact of Obfuscation

Several de-obfuscators tools and services exist for Web-related languages – such as the PHP evalhook extension [Ste10] and JavaScript JSUnpack [JSU]. These solutions typically execute the obfuscated code intercepting common language and API functions (such as `eval`, `gzip`, and `base64`) to retrieve and dump the original deobfuscated code. It is also interesting to note that, similar to packing techniques used for program binaries, also packed Web scripts often rely on multiple nested layers of obfuscation – even over 100 in few cases reported by Canali et al. [CB13a]. To further complicate the task, the deobfuscation process can also be parametrized, for instance by receiving a decryption key by the attacker or by extracting elements from the DOM tree of the webpage. Canali et al. [CB13a] also encountered PHP files obfuscated with commercial tools – such as `ionCube PHP Encoder` – which the authors were not able to de-obfuscate.

As explained more in details in Section 5.3, in our experiments almost one third of the attackers' scripts were packed. Even worse, some of those used complex techniques that are not supported by state-of-the-art static PHP de-obfuscators.

The Difference between Attack and Code Analysis

This is an important aspect which is often overlooked in the case of traditional malware. In fact, for Windows binaries, researchers are typically interested in what a sample *can* do and not just what it *actually did* on a singe execution. This is because the focus is on the sample rather than on the particular attack. For example, it is important to know that a malware can encrypt files on the disk or steal social network passwords, even if this behavior may only be triggered in particular circumstances.

However, on the Web this may be different. According to previous studies, almost 40% of the attacks against web application result in the upload of a web shell [CB13a], which is later used by the attacker to interact with the target system and perform its malicious objectives. In this case, an analysis of what a web shell *can* do is of limited interest. First because most of them provide a rich and somehow equivalent set of functionalities, and second because what it really matters in this case is what the attacker *did* with the tool and not what the tool itself can do.

File-based categorizations have been used so far for the classification of web attacks, but results can be misleading if the actual behavior of the attackers cannot be extracted. For example, Canali et al. mitigated this problem by developing custom parsers for the commands received by a number of standard web shells [CB13a]. Unfortunately, with thousands of existing variants, this method is hard to scale. Our approach aims instead at providing a more general solution for web attack analysis based on PHP files.

5.1.1 Use Cases

This study proposes a dynamic analysis sandbox for server-side web attacks. We envision that such tool can be extremely useful in three main scenarios.

Incident Response

Our solution allows to automatically reply and analyze full attack sessions, starting from an Apache mod-security log and a pre-defined entry point. This makes our approach useful for breach investigations and incident response cases that involve a web application compromise. In this case, the analyst needs to first locate the attack vector, e.g., the initial malicious request or the vulnerability used by the attacker to exploit the system. This information can come from an intrusion detection system, a web anomaly detector or web application firewall, or can be extracted from vulnerability reports or security patches. Then, the analyst just need to feed this entry point to our system together with the web

server logs and a copy of the target application. Our system can keep track of any other file uploaded by the attacker and generate a complete behavioral report containing all the attack steps that have been performed on the compromised machine.

Live behavioral analysis integrated in high interaction honeypots

Another possible use of our system is to integrate it in an existing web-based high interaction honeypot. In this case, there is no need of using the log files, as the instrumented PHP engine can simply replace the one provided by the operating system and produce aggregated behavioral reports for each incoming attack.

Replay and Analysis of previously collected attacks

Finally, our system can be used for experiment reproducibility. It allows researchers to replay previously collected attacks, both from real-world systems or from honeypot deployments. The experiments performed in this work belong to this category, as it allows to quickly reproduce and analyze thousands of different attacks.

5.2 Approach

The goal of our study is to design a sandbox to dynamically analyze the execution of malicious PHP files, while providing the exact same input used during a real attack. For this reason, our sandbox employs an instrumented PHP interpreter installed in a pre-configured virtual machine (VM). This setup, inspired by other common malware sandboxes, allows the analyst to easily revert back the VM to a clean state after an analysis is completed.

An important difference when compared with traditional malware analysis sandboxes is that to make sense of a web attack, looking at the malicious PHP files is often not enough. In fact, these files provide a visual web interface to some tools, and need to be 'stimulated' to actually perform different tasks. Therefore, the final behavior is a consequence of the interaction between the attackers and the web interface of the uploaded components, which is captured by the parameters of the incoming HTTP requests.

Our sandbox captures this interaction by replaying the input used by the attacks, as it is logged by Apache `ModSecurity` – a very popular tool for web application monitoring, logging and access control, which also allows logging POST requests' payload. Other log formats could be supported by simply extending our request

Table 5.1: List of Patched Components

Component Name	# of Patched Functions	Examples
	Standard	
Zend	1	*zend_make_compiled_...*
Basic Functions	1	*move_uploaded_file*
Dir	3	*scandir*
Exec	3	*php_exec, shell_exec*
File	9	*file_get_contents, fwrite*
Fsock	1	*php_fsockopen_stream*
Head	1	*setcookie*
Link	2	*symlink, readlink*
Mail	1	*mail*
Streamfuncs	2	*stream_socket_client*
	Extensions	
Curl	1	*curl_init*
Ftp	4	*ftp_connect, ftp_get*
Mysql	6	*mysql_db_query*
Mysqli	3	*mysqli_common_connect*
Posix	25	*posix_getpid, posix_getuid*
Sockets	3	*socket_recv, socket_send*

parsing module. Our tool replay the requests extracted from the server logs towards our sandbox, where the behavior of the malicious application is dissected and analyzed in detail.

In the following sections, we will explain how we instrumented the PHP interpreter, our tool to automatically replay and extract the details of the attacks, and how we put all components together to create our sandbox.

5.2.1 PHP Instrumentation

As we mentioned in Section 5, we implemented our dynamic analysis system for the PHP language because it is very popular among web applications and among attackers (e.g., most of Web Shells and malicious server-side scripts are implemented in PHP).

To determine the methods to instrument on the PHP interpreter, we started by examining a set of malicious files that we previously collected through our web honeypot to identify popular PHP methods and API used in malicious code. Additionally, we further checked the PHP documentation to find equivalent methods

```
static int php_mysql_select_db(php_mysql_conn *mysql, char *db
    TSRMLS_DC)
{
  PHPMY_UNBUFFERED_QUERY_CHECK();
  // Following 4 lines added
  FILE * fp;
  fp=fopen("/var/log/phpsys.log", "a");
  fprintf(fp,"Mysql database selected: %s\n" , db);
  fclose(fp);
  // --fin
  if (mysql_select_db(mysql->conn, db) != 0) {
    return 0;
  } else {
    return 1;
  }
}
```

Snippet 5.1: Patching example for *mysql_select_db* function

```
Posix: geteuid
Posix: getpwuid
Posix: getegid
Posix: getgrgid
Open File: {/tmp/bc.pl}, mode:w, dir:{/var/www/sh3llZ/
    uploadshell}
Write File: {/tmp/bc.pl}, dir:{/var/www/sh3llZ/uploadshell}
Close File: {/tmp/bc.pl}, dir:{/var/www/sh3llZ/uploadshell}
Exec Command: perl /tmp/bc.pl 192.168.1.1 31337 1>/dev/null
    2>&1 &
```

Listing 5.2: Example of lines added to the log file after a POST request made to a web shell

or other less common ways to interact with the external environment. The purpose of our instrumentation is simply to collect information about the behavior of the running application. For example, we patched PHP's `shell_exec` to log the external commands that are executed. However, commands are not blocked and can be executed correctly also in our modified interpreter, to preserve the results and side-effects which can then be used as input for following actions.

Patching a method is usually a straightforward process. An example is depicted in Snippet 5.1, where we edit the *mysql_select_db* function of the MySQL component by simply adding few extra lines to log which database has been selected. This change will not affect the outcome of the program, but will help us to understand attackers' purpose. There are also some specific cases where we need to take some extra actions. For instance, if the attacker tries to create a file in a directory that does not exist, our instrumentation creates the required

directory tree (in fact, it is possible that a path exists on the target system but not in our sandbox). We list the patched PHP components in Table 5.1.

Standard Components – *Zend* provides the engine that interprets PHP. We patched one function `zend_make_compiled_string_description` that allows us to monitor when an *eval* function is called and where it is called in the source code. We also patched `move_uploaded_file` to log file uploads, as well as other traditional *File* related functionalities such as read, write, and delete (`file_get_contents`, `fwrite`, `unlink`). The *Dir* category covers methods for navigating directories such as open, read, or scan. *Exec* is one of the key components and patching it allows us to log all executed system commands. *Fsock* and *Streamfuncs* contains methods for opening and handling socket connections. *Head* is responsible for HTTP headers, where we instrumented the `setcookie` method. In *Link* we patched two functions dedicated to the creation and access of symlinks. Last but not least, we instrumented the PHP mail component, since it is widely used by mailers, to retrieve and log the contents of outgoing messages.

Extension Components – We already mentioned stream sockets as part of the standard components (Streamfuncs & Fsock), but it is also possible to create raw socket connection by using the *Sockets* API. Additionally, we instrumented *MySQL* and *MySQLi* which enables us to log any database query done by using their methods. It is not an uncommon behavior for an attacker to check if he has necessary permissions in a system that he/she infiltrated. Thus, we instrumented the *Posix* APIs, which provides various functionalities to query key information such as IDs (user, group, effective etc.), checking accessibility of a file and so on. Finally, we patched *FTP* API to log FTP connections and the *Curl* extension to keep track of attempted HTTP requests.

Listing 5.2 shows a snippet of the log generated by our instrumented PHP interpreter during a real interaction between an attacker and his web shell. In this example, we see the lines added after a POST request is made to the file `xwso.php`. By looking at the logged information, we understand that the code first retrieves the identity information. Then, a Perl script, `bc.pl`, is created and after written under the $/tmp$ folder. Finally, the attacker tries to execute this newly created script.

5.2.2 Attack Replay

As mentioned earlier, our purpose is to create a virtual machine running an instrumented version of the PHP interpreter, and then to replay attacks by automatically sending the requests extracted from a `ModSecurity` log. Since ModSecurity stores not only the request payload, but also all the HTTP headers, our system can faithfully replay the exact same requests as they were received

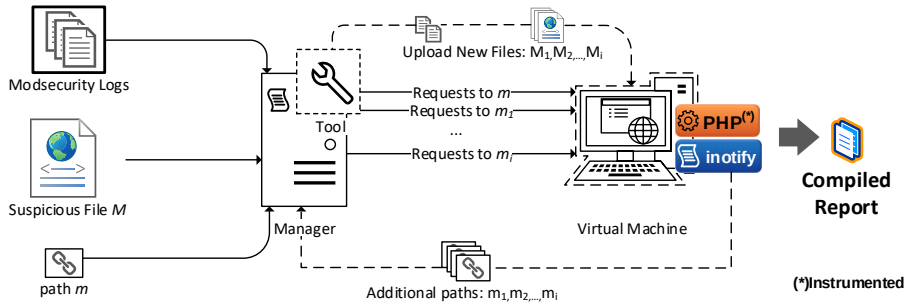

Figure 5.1: Overview of the Sandbox Infrastructure

on the target machine. Note that it is also possible to use our sandbox on a live system (e.g., installed directly in a honeypot), and in this case the log analysis and request replay components are not necessary. However, we chose to concentrate on an offline approach as this is a more common use case and makes our experiment reproducible.

As we consider that the system whose logs we use is previously unknown to us, at least a 'suspicious file' and a path/stem of URL corresponding to the file have to be provided as inputs to the tool. As we example in Section 5.3, this may require the analyst to first discover attack signatures or anomalies in the logs and identify the uploaded/modified attack-related files as well as the corresponding URL paths. However, techniques for web anomaly detection [SS02, HYH+04, JKK06a] already exist and they are outside the scope of this study.

Hence, the execution of our system involves five different steps:

1. Parse ModSecurity logs of the web application.
2. Copy the suspicious *file* to analyze to our sandbox VM.
3. Replay all the attack-related HTTP requests to the uploaded file and any other file subsequently uploaded by the attacker.
4. Compile a report that summarizes the collected information and the actions performed during the attack.
5. Revert the VM back to its clean state.

As mentioned in step three, it is important to be able to identify requests that are made to different paths, but belong to the same attack. For instance, it is very common for an attacker to upload a file by exploiting a vulnerability in the target web application. Then, this file is used to upload other components, which are then activated and controlled by subsequent HTTP requests. To properly handle this common scenario, our tool updates the list of target URL paths for every

newly uploaded files. This is achieved by installing into the sandbox a real-time file system event monitoring framework, `inotify`. Once `inotify` detects a new file, it immediately notifies our replay component that adds its path to the list of those that need to be extracted from the log file and replayed to the sandbox.

The overview of the sandbox is depicted in Figure 5.1. When the tool is executed to analyze a file M (with corresponding URL path m), our system immediately starts searching for HTTP requests made to m inside the `ModSecurity` logs. Then, it will replay them one by one to the VM. As new files $M_1, M_2, ..., M_i$ are uploaded by the attacker, their corresponding URL paths $m_1, m_2, ..., m_i$, retrieved through `inotify`, will also be added to the list of URL paths. The execution will continue until no more requests can be found inside the ModSecurity logs for the target URLs.

5.3 Experiments

In this section we present the experiments we conducted to evaluate our tool using the Modsecurity audit logs of two web application honeypots that we operated over a period of two years (between 2014 and 2016): One of the honeypots runs an intentionally misconfigured web server that allows directory listing and contains a number of sub-directories with different web shells to mimic a previously compromised machine. The other honeypot runs instead an old and vulnerable version of a popular e-marketing web application called $OSCommerce$. The vulnerability allowed attackers to upload arbitrary files. In the course of running these honeypots, we collected the ModSecurity log data daily, and reverted the honeypots to their original clean state every midnight. Moreover, we blocked all outgoing connections from the honeypots with a firewall to prevent the attackers from abusing our honeypots as a medium to launch further attacks.

Our experiments consist of two main phases: attack identification and attack analysis. In the first phase, we extract from the HTTP requests sent to the two honeypots the entry points required for the sandbox analysis. In particular, since we want to study with our system the behavior of malicious files uploaded by the attacker, and not the dynamic behavior of our Web Shells or of the OSCommerce application, we first need to identify those uploaded files. In the second phase, we run the extracted files in our sandbox, while replaying all requests targeting their URLs.

5.3.1 First Phase: Extracting the Malicious Files

As we chose to evaluate our tool offline (Section 5.2.2), we first set up two virtual machines that perfectly replicated the content of the honeypots. The overview

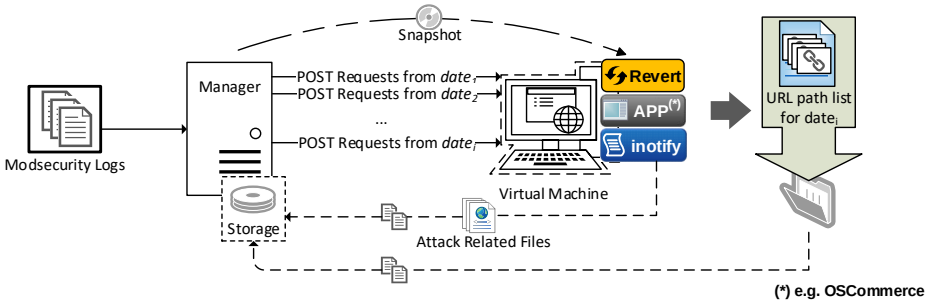

Figure 5.2: Overview of the Replaying Traffic

of the first phase of our experiments is shown in Figure 5.2. The **manager** machine at the center of the picture stores the ModSecurity logs, as well as the output data of both the first and the second phases. Since we are interested in uploaded files, in this phase we only replayed POST requests. The virtual machines are equipped with a `inotify` script to automatically acquire files and URL paths that are affiliated with an attack. The newly uploaded/modified files are then automatically retrieved by the manager.

Note that sometimes attackers upload files with the same name, which may overwrite a previously uploaded file. Therefore, our system keeps track of the timestamp when each file was uploaded, and later use this information in the second phase to replay only the correct part of the log to each file.

The output of the first experiment phase consists of a set of malicious files that include various data for each day. In particular, an intermediary file contains multiple rows with the following: the location of the attack related files in the manager machine (will be referred as the 'local file path'), attack related URLs (target URLs), and timestamp.

5.3.2 Second Phase: Attack Analysis

For the second phase we again used two virtual machines that mimic the content of the honeypots, this time with our installed sandbox and the instrumented PHP interpreter. Then, for each malicious file identified in the first phase, we uploaded the file to the virtual machine (preserving its original path where it was installed by the attacker) and we then replay all HTTP requests from the ModSecurity logs in the appropriate time-frame.

As mentioned in Section 5.2.2, our system replay all requests sent to the file under analysis, as well as any other uploaded files in a recursive way.

Once all requests to a particular file and all its "children" have been successfully replayed, the manager retrieves the PHP logs, computes an aggregated behavioral report, and store it to be later examined by an analyst.

Table 5.2: Number of attacks according to the information gathered

	Look Around	Investigate	Network
OSCommerce	1286	16	12
Web Shells	1646	88 (187)	5

5.3.3 Information Gathering

Gathering information about the target system is usually one of the first steps of a manual attack. We now look at three types of information collected by the attackers.

System Exploration – One of the most common behaviors of an attacker is to collect basic information about the system, or simply to 'look around'.

An easy way to do that is by fetching the contents of the current directory, retrieving the user id, or by printing the system information. Thus, acquiring such information via system commands such as `id`, `whoami`, `uname`, `hostname`, `ps aux`, `pwd`, and `ls` falls under this category. However, identifying and labeling other system exploration actions are not straightforward. For instance, listing the contents of the working directory is an ordinary and common process, which is also automatically performed by certain web shells. However, the same operation can also be done manually to locate important application or configuration files. Therefore, to be conservative, we only counted generic filesystem tasks as system exploration if they were executed as shell commands and not if they were performed by PHP routines. As can be seen from the Table 5.2, this behaviour is observed in a considerable number of attacks for both honeypots.

Files Investigation – In this second category we put more fine-grained information gathering actions, in which the attacker specifically targeted application configurations or the content of other system files. For instance, a common example in this category is accessing the `/etc/passwd` file to list the registered users. Another example is locating the `.htpasswd` files that are used to store username-password pairs for basic authentication of HTTP users and also to configure Apache to run scripts via HTTP. We consider the system commands `find`, `cat`, `ls`, `awk`, `ln` and `locate` to belong to this category. Here, the `ls` and `cat` command are only counted if provided with particular parameters to look into configuration directories or the `/proc` filesystem. For instance, in one attack, the attacker manually navigated to the `/sys/` folder and ended up listing the contents of the `/sys/block` where information regarding block devices is being held.

We found this file investigation behavior in only 16 (0.5%) and 275 (5.7%) attacks for the OSCommerce and Web Shell honeypots respectively.

Table 5.3: Number of attacks by using disguised or obfuscated files

	Obfuscated	Disguised
OSCommerce	611	1377
Web Shells	1649	40

Local Network – Last but not least, we observed that attackers tried to retrieve some information about the local network setup in a small number of attacks. In order to do so, they used various system commands, including `netstat` and `ifconfig`.

5.3.4 Disguised & Obfuscated Files

We find that 29% of the attacks included at least one obfuscated file (see Table 5.3). The majority of them relied on a simple procedure that first decodes a long encoded string (usually `base64`), then decompresses it by calling `gzinflate`, and finally passes the result to the *eval* function to be interpreted as PHP code. Some attacks adopted more complex obfuscation, using for instance "variable variables", naming variables in Unicode, or even implementing a custom decryption/deobfuscation algorithm. For instance, the popular online deobfuscation tool *UnPHP*[1], which was also used by a previous work [SDA+16a] to statically analyze malicious files, was unable to deobfuscate 16% of the packed files in our dataset. We believe this further emphasizes the importance of using the dynamic analysis approach we adopted in this work.

Attackers also uploaded scripts (mainly PHP and Perl) with false header information to disguise them as image files. This is a common behavior, adopted to evade simple control mechanisms and file upload checks. For instance, attackers abused some OSCommerce features (such as those to set a banner image or add a new thumbnail for a product) to upload disguised malicious files. Thanks to the our instrumented PHP, we were able to detect the execution of files which are given a different extension rather than their original type. To be more specific, we found that such files downloaded and executed via system commands in 31 attacks. Because the web applications may use blacklists for the files with the certain dangerous extensions (e.g. Perl), files disguised in this way can penetrate through the blacklist.

[1] https://www.unphp.net

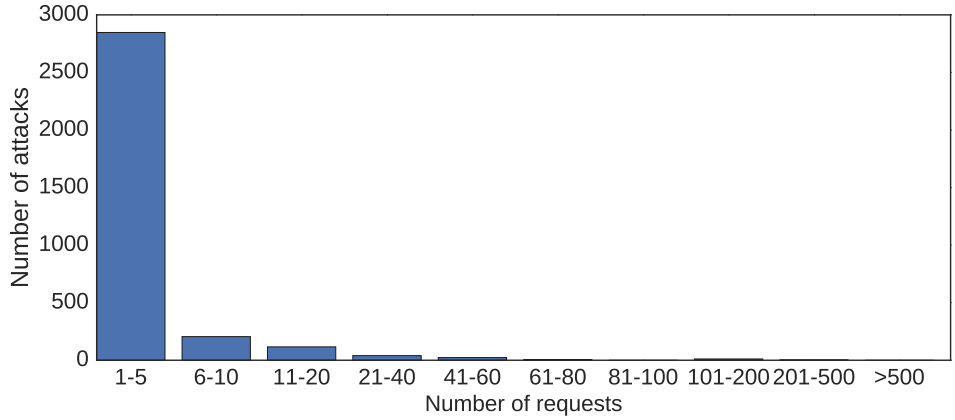

Figure 5.3: Number of requests by number of attacks for OSCommerce

5.4 Results

We used our tool to analyze all successful, non-empty attacks received by the two honeypot systems. We considered an attack *successful* if the attacker uploaded at least one additional file (either by exploiting the vulnerability in the OSCommerce application or by taking advantage of one of the pre-installed web shells) and *non-empty* if such file received at least one HTTP request from the attacker. These simple heuristics rule out simple defacement attacks, which are still relevant but do not require a dynamic analysis system to understand their behavior, as well as automated attacks that upload files that are never used afterwards. It is important to note that since our honeypot machines were rolled back every 24h, this second category boils down to successful exploitation in which the attacker did not interact with the system before midnight.

After this filtering stage, we were left with over 8K attack sessions: 3248 recorded against the OSCommerce application and 4818 against the server hosting Web Shells. As it can be seen from Figure 5.3 and Figure 5.4, a large proportion of the attacks consisted of just few requests, performed right after the malicious file was uploaded. These were most likely fully automated attacks in which a script exploited the vulnerability, uploaded a malicious file, and then visited the corresponding URL to trigger its behavior. This is also confirmed by the fact that the triggering request was often received multiple times in a row in a rapid burst, probably to be certain that the uploaded file was executed – and by the fact that most of these attacks were recorded on the OSCommerce system. In this case, the report generated by our system corresponded to the behavior observed from a single execution of the malicious PHP file.

The remaining 34% of the attacks contained instead multiple different requests, with a pattern that suggested the presence of a human attacker interacting with the system. In 28% of these cases the attackers uploaded multiple files. In this

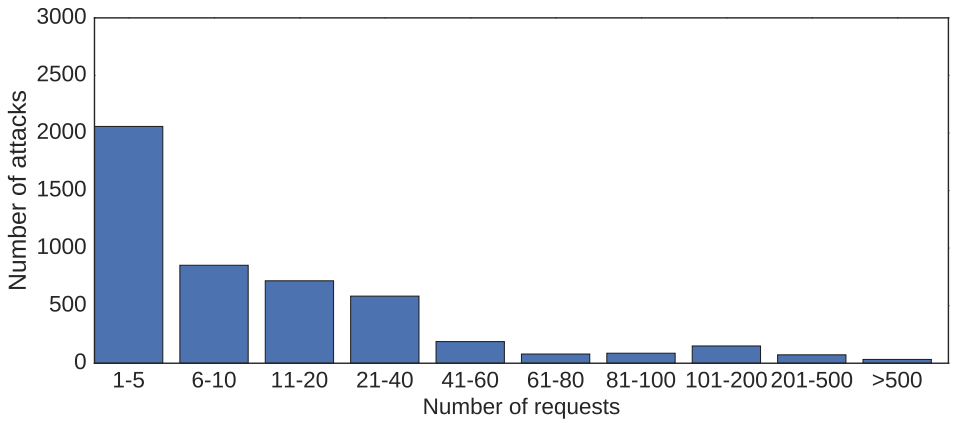

Figure 5.4: Number of requests by number of attacks for Web Shells

case, the report generated by our system covered the behavior of multiple PHP files, each possibly executed multiple times with different parameters.

It is also interesting to see how different types of vulnerable machines may result in a completely different nature and distribution of attacks they collect. The OSCommerce vulnerability was most likely identified using Google Dorks [Joh17] and targeted by automated scripts. On the other hand, the second honeypot mimicked an already compromised machine where previous attackers left a web shell installed on the system. In this case, automated crawlers or Google Dorks might have been used to discover the presence of the shell, but to take advantage of it required a manual interaction from the attacker. Our data also shows that miscreants tend to be more careful when they manually infiltrate an already compromised system.

This also resulted in different attack behaviors, as summarized by Figure 5.5. Again, we see that different classes of actions appear with largely different percentages in the two honeypots. For example, it was not unusual for the attackers on the web shell honeypot to delete their files after they completed their actions. However, this behavior was almost never observed in the attacks against the vulnerable web application. Even more extreme is the case of emails. Over 60% of the attacks against OSCommerce resulted in at least one email being sent – either as a way to notify back the attackers of a successful exploitation or as part of a spam campaign sent by an uploaded mailer. Another possible explanation could be that a CMS like OSCommerce usually have a mail server set up and since the automated scripts are configured to target a specific CMS, attackers expected to use this mechanism freely in such servers. Attacks originating on the web shell honeypot instead *never* attempted to send emails, confirming the mostly manual nature of these attacks.

It is no surprise instead that a large number of attacks on both platforms involve

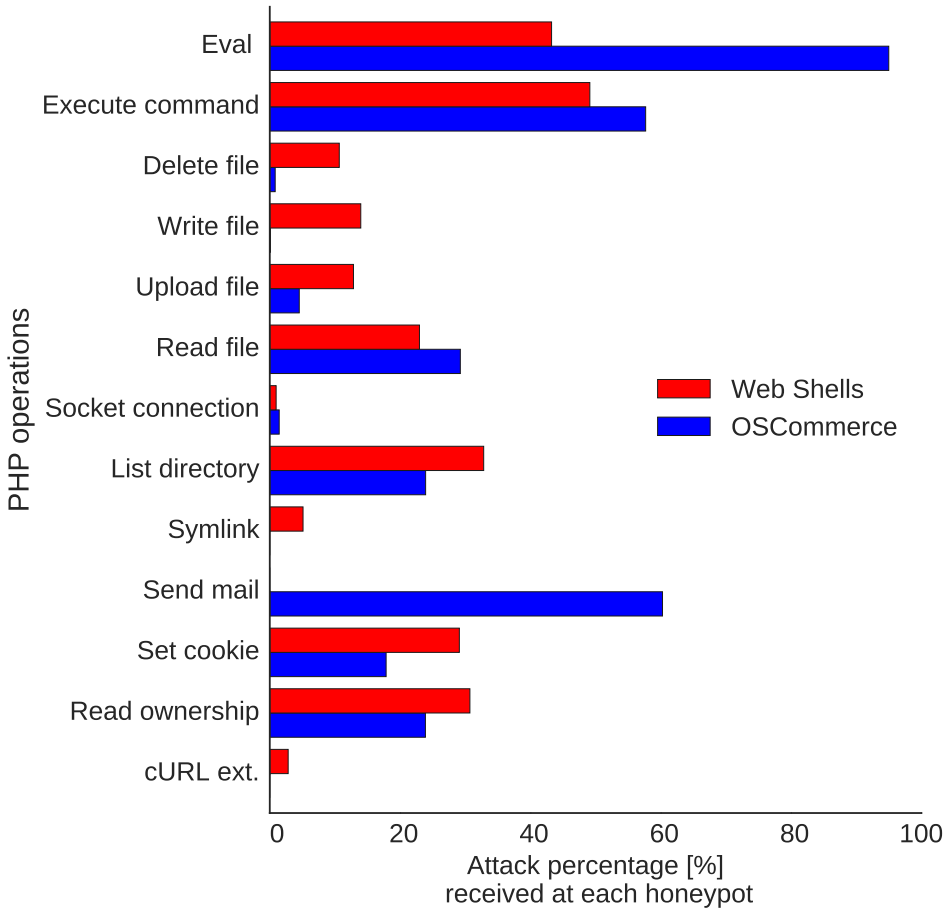

Figure 5.5: Number of attacks containing specific operation

the execution of system commands. The most frequently invoked commands include tools to gather information such as `whoami`, `uname`, `ls`, `which`, `id`, and `pwd`. Some of them were invoked by the attacker, while others were executed automatically at every access from certain uploaded scripts or web shells (as it is the case, for instance, of '`id`').

There is not much difference also when it comes to reading the contents of files, browsing directories, or fetching user privilege information such as user and group ids.

The use of obfuscation, and in particular of the `eval` statement, is again largely different among the two machines – but this time being more prevalent on the attacks against OSCommerce. The use of the '`write`' API also shows a clear unbalance. In this case, a closer look shows that it was used almost exclusively by mailers, which are tools dedicated to send large volumes of spam or phishing e-mails. Among other uses, miscreants created/modified '`.htaccess`' files in

226 attacks to restrict access to the directory they store their files. Also in 75 attacks, an `index` file was created to prevent visitors to list the content of the directory.

The cURL extension was used for different purposes, including to launch a, intentional or not, DOS attack against localhost's telnet, ftp, and cPanel (2082) ports. For example, in one attack the malicious file attempted to connect to (`http://localhost:2082`) over 314K times. Attackers also attempted to launch DOS attacks by using PHP sockets. For example, one attack attempted to send respectively 150K, 251K, and 9.4M UDP packets against three separate IP addresses.

Additionally, attackers tried to open connections to some IRC domains as part of 12 attacks against OSCommerce. A quick search for some of them (e.g. `irc.byroe.org`) revealed that they are indeed related to botnets. Finally, contrary to expectations, database APIs received very little attention and did not yield any interesting results for both honeypots. This may suggest that attacks against OSCommerce were not interested in stealing users' information but were mostly looking for a compromised machine to run their tools.

5.5 Case Studies

In this section, we present two case studies to better demonstrate how our methodology helps in understanding the behavior of a malicious file and the actions performed during a web attack.

5.5.1 Case I

In this first example, the attacker exploited the OSCommerce vulnerability and used it to upload a simple web shell. We extract the URL of the shell from the Apache logs and run it in our sandbox. The behavioral report shows that the attacker first used the web shell to create a `.htaccess` file that configured the Apache server to run Perl CGI scripts from the current directory. He then uploaded a Perl script named `cwo.pl`. According to its content, the purpose of this tool appears to be gathering sensitive information from the server, including the hosted domains and the list of users. Still interacting with his web shell, the attacker gives the execute permission (using the system command *'chmod 755 cwo.pl'* to the script, and run it by sending a simple GET request to its URL. At this point the attacker might have noticed something suspicious, as the information collected from our honeypot may not have looked as realistic as in an active CMS. Possibly for this reason, the attacker started exploring the users of the honeypot by performing the following steps:

- Check if any user is logged on using the `who` command

- Upload a second web shell
- Scan the content of the `..OSCommerce/catalog/`
- Attempt to retrieve the content of the `/etc/group` and `/etc/passwd` files
- List the content of `/var/www/`
- Check what logged in users are doing using the `w` command
- Finally, delete everything in the current working directory recursively (`rm -r *`), and leave.

This attack shows three important things. First, that the attack left no signs in the system after the attacker left. Second, that simply looking at the uploaded files, as it was done by Canali et al. [CB13a], would only allow to identify two web shells. Even a careful static analysis of these shells would reveal nothing interesting about the attack. Likewise, executing the shells would be pointless as their behavior is only dictated by the input provided by the user. Finally, reconstructing the attack steps by simply looking at the web server log files would be difficult, as to interpret some of the actions it would be necessary to first understand the code of the web shell and the format of its commands.

Our sandbox instead can automatically replay the entire attack session while monitoring the attacker's steps. The report generated by our system shows how carefully the attacker investigates the server before deciding to leave the machine.

5.5.2 Case II

The second example is taken from the Web Shell honeypot. Also in this case, the attacker used one of the pre-installed web shells to install another shell of her choice – which is the entry point for our analysis. Using her shell, the attacker retrieved the users' information from `/etc/passwd` and the block device attributes cache from `/etc/blkid.tab`.

She then created a `.htaccess` file similar to the one encountered in our previous example, and eventually uploaded a Perl script called `slowloris.pl`. A quick research reveals that this file is a DDoS attack software and works by opening multiple connections to the target machine. Immediately after installing her tool, the attacker executed it by passing it as parameter an IP address located in North Africa. Since our honeypot prevented any outgoing connections, what happened after that is particularly interesting. As the attacker probably noticed that the attack had failed, she returned few minutes after and used her web shell to upload another file, this time a PHP code. This file was also triggered by an HTTP request and its behavior, captured by our sandbox, was again to

start another DOS attack by opening multiple socket connections to the same IP address. This second file was obfuscated with a complex encryption schema, and current state-of-the-art PHP deobfuscators failed to retrieve its content.

5.6 Discussions & Limitations

Both our honeypot systems and our analysis sandbox were configured to block any outgoing connection attempt. This makes the result of our analysis consistent with what was collected and performed in the honeypot machine. However, if the web server logs were acquired from a machine with Internet access, and then replayed on a firewalled sandbox, it is possible that the final behavior would diverge if the attacker downloaded (instead of uploading) his components. In fact, in this case the download operations would not be logged by ModSecurity and would fail in the sandbox execution. A similar result could be obtained by replaying an attack in a sandbox long after the attack was performed (when externally downloaded components may not be available anymore). This is a common problem in malware analysis, where C&C servers are often offline at the time a sample is executed in the analysis environment. However, this phenomenon did not verify in our experiments, since even the machine where the logs were collected did not allow outgoing connections, thus forcing the attackers to upload, instead of downloading, their files.

In our experiments, we consider an attack anything that happened following an uploaded file on one of the two honeypots. However, it is possible that an attacker exploited the same vulnerability two times in a row to upload two separate components – which therefore belonged to the same attack. In this case, the two files would serve as separate entry points for our analysis, thus being counted as two separate attacks. However, this is not a limitation of our sandbox, but just a way we process the logs acquired with the web application honeypots. If this was a real attack, an analyst would simply pass both URLs as part of the same analysis, therefore allowing our system to generate a single behavioral report for the entire attack session.

5.7 Conclusions

The goal of this Chapter is to facilitate the analysis of server-side web attacks, by introducing the first dynamic analysis framework specifically designed for this purpose. Previous static analysis approaches focused more on the malicious code itself, which may fail to capture subtle attack details and the overall attacker behavior. Therefore, inspired by modern malware sandboxes, our aim is to perform a fine-grained analysis of web attacks by following a dynamic approach.

Our work starts with the instrumentation of the PHP interpreter, which we then install in a virtual environment and feed with real attack traces to to monitor the behavior and evolution of web attacks. We evaluate our approach using the two previously deployed honeypots described in Chapters 3 and 4. Overall, we successfully replayed more than 8000 attacks to our system, and generated detailed attack reports for each of them. The analysis of these reports show that different types of web applications receive attacks of different nature. Moreover, even though the adversaries may rely on the same set of tools, their goal may be quite different – ranging from system exploration and file investigation to performing DDoS attacks and page defacements.

Chapter 6

Conclusion and Future Perspectives

In this thesis, we argued that the analysis of web attacks and especially of the attackers behavior is essential to better understand current threats against web applications. In fact, the analysis of these attacks enables us not only to determine the real impact of an attack but also to automatically identify similar cases and potential victims. Motivated by this observation, we relied on high-interaction virtualized bait systems, normally called honeypots, to conduct our experiments.

To support our argument, we demonstrated in the first part of the thesis how the automated nature of web attacks can be used as a leverage to discover other compromised web applications. We showed that the external resources of an exploit kit, which may be completely benign per se, can be successfully used as `Indicators of Compromise` for web applications. We conducted live experiments on several honeypots to automatically extract these external resources from real attacks and proposed a set of features to distinguish the ones that qualify as valid IOCs.

Second, we looked into the attack landscape in a private part of the Internet called Tor network, by deploying high-interaction honeypots as Tor hidden services. We showed that attacks in Dark Web do indeed have a lot in common to those we observe on the normal Web, because of the fact that proxy services can channel automated exploitation scripts from the surface to the Dark Web. Additionally, our work revealed certain attack vectors specific to the Tor network and different post-explanation behaviors once the traffic from the Surface Web is filtered out.

In the final part of the thesis, we explain why the analysis regarding the attacker behavior against web applications is challenging and we emphasize the importance of a dynamic approach in that area. We present a sandbox with an

instrumented PHP interpreter in which we replay the traffic of more than 8,000 attacks acquired from two honeypots. The results of our experiments showed that different kinds of vulnerabilities in web applications attract different type of attacks. Our tool provides a much more complete overview on the actual behavior of the attackers compared to other state-of-the-art methodologies.

6.1 Future Work

In this dissertation, we studied how to collect, analyze, and use information related to the behavior of attackers on the Web. However, many aspects of Web attacks still remain unexplored and many of their details are left to discover for future experiments.

6.1.1 Public external resources

As mentioned in Chapter 3, attackers often maintain part of their toolkits on public code repositories. We have discovered this phenomenon during the collection of URLs extracted from the components uploaded by attackers. We believe that many of such public external resources may be used to build Web indicators of compromise. In this direction, an interesting future direction can focus on the analysis how this data can be an asset for detecting compromised web applications.

Some of the popular libraries for changing the visual appearance of web applications are tailored and used by large companies like Yahoo or Paypal. We discovered that miscreants may use the very same resources - even sometimes the same URL - to imitate the visual aspects of their web sites. As this allows a perfect setup for phishing attacks, analyzing such external components can provide a valuable source of information for the automatic detection of phishing pages.

6.1.2 Unexplored aspects of the Dark Web

The work we presented in Chapter 4 was the initial attempt to understand the attacker behavior in the Dark Web. For our experiments we only considered the Tor network because it is the most popular one. However, there are also other instances of Dark Web, such as the Invisible Internet Project (I2P) where the attack landscape is still unknown.

Our study also reveals attack vectors specific to the Tor network. Though we explained our findings, the vulnerabilities peculiar to Tor were never truly exploited. Thus, questions like how prevalent these attacks and what attackers do

once they exploit these vulnerabilities were left unanswered. Additionally, our work adopts a server-side approach for the analysis of the attacks in the Tor network. Recently, researchers (see Section 2.3) started to look into the suspicious behavior of hidden services, but still more experiments adopting a client-side approach may help to identify potential risks related to the Tor network.

6.1.3 Improvements on Server-Side Analysis of Web Attacks

The honeypot infrastructure have certain limitations just because they are systems designed to attract adversaries on the Web. One of them is that the popularity (i.e., the ranking on a search engine) of honeypots is often much lower than the of other existing web pages. While this could be an advantage for normal users since the rate of access to a compromised web application will decrease, low search index will also attract much fewer real and manual attackers. It would be possible to perform a more refined study if such experiments were conducted together with some of the vendors on the Internet.

Our work proposed in Chapter 5 is also designed to address the above mentioned concerns. However, our instrumented PHP interpreter and/or sandbox for web attacks cannot solve such issues by itself. While PHP is the most common server-side language for developing web applications, there are also other various scripting languages commonly used for web development. A possible improvement in that sense would be to include more instrumented scripts for web applications to further use in a sandbox.

In addition, our work initially relies on a component, such as an Intrusion Detection System or manual inspection, to detect the initial step of the attack. However, it is possible to create a much more sophisticated and fully automatized web attack sandbox by also including a static approach for detecting/capturing the initial step of the attack.

6.2 Concluding thoughts

Even though the Internet users are today more aware of the possible threats, security of web applications is likely to remain an important issue for the future. In order to adopt a proactive approach, understanding the nature of the attacks and the mindset of the adversaries is crucial. We believe our work will help to create more awareness and encourage more academic research on this interesting topic.

Appendix A

Résumé en français

Le Web ressemble aujourd'hui à un grand centre commercial avec un public mondial pour ses produits et services. Dans ce média, les entreprises interagissent avec leurs clients via des applications Web, qui permettent la présentation et / ou la vente de divers produits et services. Cependant, l'adoption de ce canal de communication apporte de nouveaux défis, dont la sécurité de telles applications Web. Bien que les grandes entreprises aient tendance à prendre cette question au sérieux, les petites et moyennes entreprises manquent souvent de compétences, de temps et de ressources pour bien sécuriser leurs systèmes. En raison du fait que les développeurs ne sont souvent pas conscients des différentes menaces et de leurs conséquences possibles, même les mesures de sécurité les plus élémentaires sont négligées. Une croyance commune parmi de nombreux propriétaires

de commerce électronique est que l'Internet est énorme et, par conséquent, leur site Web ne sera pas remarqué par les mécréants. Cependant, les pirates utilisent généralement des scanners automatisés et des robots d'indexation des moteurs de recherche pour trouver des vulnérabilités. Alors que les attaques ciblées (telles que la violation de données affectant LinkedIn en 2016) nécessitent un processus rigoureux et fastidieux (notamment une reconnaissance méticuleuse et une stratégie et une exécution bien planifiées), l'automatisation offre une exposition massive et des chances de succès plus grandes. cibles beaucoup plus vite. En outre, les outils automatisés peuvent également être utilisés avec succès par des attaquants inexpérimentés - résultant en un arrière-plan constant d'attaques pas très sophistiquées, mais toujours très efficaces, qui peuvent facilement atteindre chaque site Web vulnérable sur le Web. Malgré les efforts considérables de la

communauté de la sécurité, le pourcentage de sites Web vulnérables n'a montré aucun signe de récession au cours des dernières années. En effet, 76% des sites Web analysés par des chercheurs en sécurité en 2016 contenaient des vulnérabilités [Sym17, Aka17]. Les adversaires sont régulièrement à la recherche de serveurs vulnérables sur le Web afin d'exploiter et de tirer parti de ces systèmes

exposés. Une fois compromis, les applications Web sont utilisées abusivement à diverses fins, notamment pour déployer des botnets, servir des kits d'exploitation, installer des kits d'hameçonnage ou simplement servir de tremplin pour lancer d'autres attaques Bien que les cybercriminels puissent également compter sur

des serveurs Web dédiés dans leur infrastructure malveillante, il est généralement plus avantageux de compromettre les serveurs Web légitimes, car ils sont souvent perçus comme fiables par d'autres utilisateurs et n'attirent pas l'attention de la communauté de sécurité. De plus, un serveur Web compromis peut autoriser un accès supplémentaire à un réseau privé qui serait normalement protégé par un pare-feu. Tous ces facteurs suggèrent que la sécurité des applications Web reste un aspect crucial de l'écosystème de sécurité dans son ensemble, car les sites compromis mettent toujours en danger les clients, les autres entreprises et même les sites publics et gouvernementaux.

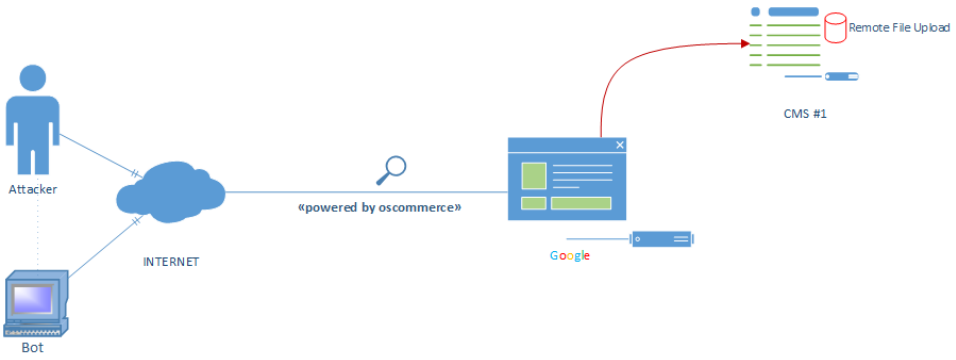

Figure A.1: Utilisation de dorks pour trouver des cibles

Web Attack - Un aperçu

Avant d'entrer dans les détails des attaques web, il est essentiel de comprendre le processus derrière ce phénomène. En raison de la nature vaste et complexe d'Internet, le terme attaque Web a également une signification très large. Cependant, comme cette thèse se concentre sur la sécurité des applications web, nous affinons notre modèle d'attaque à ce domaine particulier. La phase initiale d'une

attaque consiste généralement en l'identification des cibles possibles. À cette fin, les pirates commencent souvent par tirer parti des moteurs de recherche populaires, en s'appuyant sur des robots automatisés pour rechercher un ensemble de mots-clés (généralement appelés Google dorks [TACB16]) conçus pour identifier les sites Web mal configurés ou non. probablement affecté par une vulnérabilité connue. Par exemple, les sites Web qui exposent les fichiers d'historique MySQL peuvent être récupérés à l'aide de la requête Google "? Intitle: index.of?" .mysql_history "En utilisant la même technique, il est également possible de trouver des versions spécifiques d'applications vulnérables, par exemple Pour les bannières connues comme "Powered by OsCommerce", le scénario typique de la phase d'identification est illustré dans la Figure A.1.

Une fois que l'attaquant a trouvé ses cibles, elle peut procéder à la phase d'exploitation, encore une fois généralement effectuée par des scripts automatisés. Dans cette thèse, nous sommes particulièrement intéressés par ce qui se passe après que l'attaquant ait réussi à compromettre l'application cible - dans ce que Canali et al. [CB13a] a appelé la phase post-exploitation. Dans cette phase, présentée dans la Figure A.2, les attaquants essaient d'atteindre leur objectif final, qui consiste normalement soit à télécharger de nouveaux fichiers sur la machine compromise, soit à altérer le contenu du code existant et des pages HTML. En particulier, un outil très commun qui est souvent installé après qu'une applica-

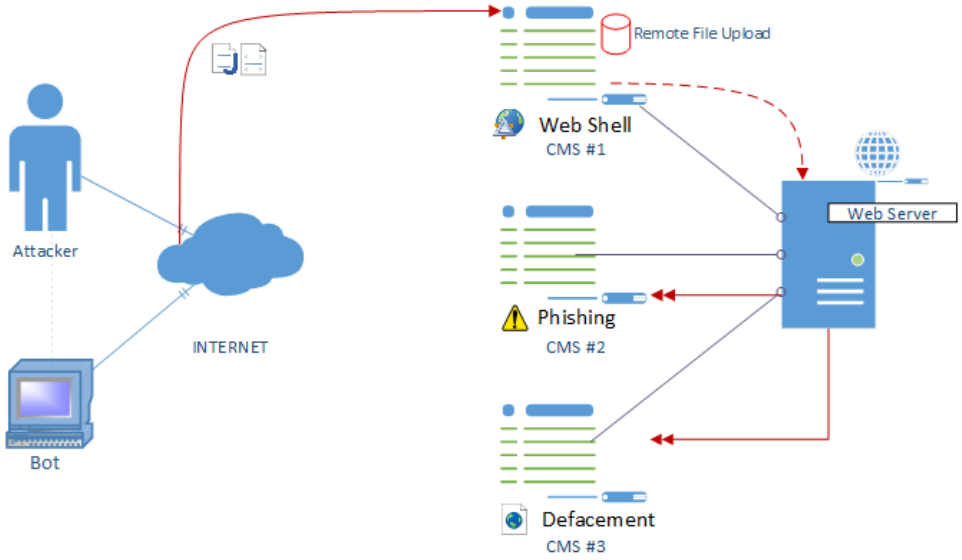

Figure A.2: Un modèle d'attaque Web général pour les applications Web

tion a été exploitée s'appelle un "web shell". L'objectif de ces scripts est de permettre aux pirates de contrôler facilement la machine compromise et d'exécuter des commandes arbitraires en utilisant une interface web intuitive. Les web shell peuvent être utilisés à différentes fins, par exemple en exploitant d'autres techniques d'exploitation pour augmenter les privilèges, collecter des informations sur le système, analyser le réseau interne de la machine hôte ou extraire des documents et des données d'application. Un autre exemple d'outil populaire installé

après un compromis est un "kit d'hameçonnage". L'hameçonnage est un type d'escroquerie pour attirer les victimes à fournir des données sensibles telles que les détails de carte de crédit, les mots de passe ou les informations personnelles utilisées pour le vol d'identité. Un kit d'hameçonnage est une boîte à outils qui imite l'apparence des sites Web existants (par exemple, Bank of America ou Paypal), permettant ainsi aux personnes ayant peu d'expérience technique de lancer une campagne d'hameçonnage. Les particuliers peuvent utiliser cette boîte à outils pour envoyer des spams ou pour transformer la machine d'hébergement en un site frauduleux pour escroquer des gens. Les attaquants peuvent également

"défacement" la page d'accueil du serveur web compromis en introduisant un message de personnalisation. Les motivations derrière les suppressions de sites Web défacement différer. Par exemple, les attaquants qui sont contre un mouvement particulier ou un gouvernement (hacktivistes) peuvent choisir de laisser un message politique, d'autres peuvent simplement railler les administrateurs du site ou se faire de la publicité pour gagner de la célébrité. Mis à part les exemples

mentionnés ci-dessus, il existe d'innombrables types de fichiers malveillants et

de scripts téléchargés sur des applications Web compromises, chacun servant un objectif différent et spécifique, comme joindre un botnet, envoyer de gros volumes de spams ou scanner le réseau. autres victimes possibles. Tout au long de cette dissertation, nous présenterons de nombreux types d'instruments d'attaque de ce type et nous discuterons plus en détail de leur impact.

Comprendre la nature des attaques Web

L'analyse de la nature, de la motivation et des détails techniques de la conduite des attaques Web peut fournir une information précieuse à la communauté de la sécurité. Puisque les mécréants laissent généralement des traces après avoir compromis un système, l'étude de leurs actions et des modèles qu'ils suivent pendant et après une attaque peut être d'une grande valeur. Comprendre comment les méchants infiltrent le système peut aider à le sécuriser et à corriger d'éventuelles vulnérabilités. De plus, des schémas spécifiques, un fichier unique ou parfois même simplement un mot-clé pourraient aider à identifier des cas similaires d'infections. Un exemple simple serait un attaquant qui utilise la même image unique pour défigurer la page d'accueil de ses sites Web ciblés. Compte tenu de l'utilisation courante des outils automatisés, de telles informations peuvent permettre une détection à grande échelle de la même attaque. Bien qu'aucun système ne soit totalement à l'abri des attaques Web, les propriétaires de sites peuvent prendre des précautions supplémentaires s'ils savent comment les criminels découvrent leurs cibles. De telles précautions sont particulièrement importantes dans l'existence d'exploits de 0 jour, c'est-à-dire lorsque l'application souffre de vulnérabilités précédemment inconnues pour lesquelles un correctif n'est pas encore disponible. Par exemple, une vulnérabilité de 0 jour affectant une certaine version d'un système de gestion de contenu Web (CMS) encouragerait les mécréants à identifier automatiquement ces sites Web. En connaissant les stratégies utilisées par les attaquants pour localiser leurs cibles, il peut être possible d'échapper aux attaques même si le site est toujours vulnérable [TACB16]. De plus, révéler la motivation d'un attaquant peut aider à récupérer rapidement après un compromis. Si des données d'utilisateur sont divulguées, les clients doivent être avertis de changer leurs mots de passe; ou si les mécréants placent une porte dérobée, les agents de sécurité doivent la réparer dès que possible. Comprendre comment les mécréants développent leur vecteur d'attaque et comment ils délivrent un contenu malveillant est crucial pour une telle enquête. Par exemple, un certain type de comportement observé dans une attaque similaire peut aider à identifier le but de l'attaque, et par conséquent de telles informations peuvent être utilisées pour trouver les parties impactées du système. Pour résumer, nous croyons que la lutte contre les attaques web

ne peut être réalisée uniquement en construisant des mécanismes de défense. Tout aussi important est la capacité de comprendre la nature des attaques et la motivation derrière elles.

Honeypots

Dans la section précédente, nous avons souligné la nécessité d'avoir un système permettant l'observation des attaques web pour améliorer la sécurité des applications web. Afin d'étudier la nature des attaques à grande échelle, l'une des approches les plus communes consiste à mettre en place des systèmes d'appâts à l'aspect vulnérable - qui sont généralement appelés pots de miel. Les pots de miel peuvent être construits pour presque n'importe quel protocole de communication, à la fois au niveau du réseau et de la couche applicative (par exemple HTTP, SSH, Telnet et SMTP). Cependant, puisque cette dissertation se concentre sur les attaques web ciblant les applications web, nous ne couvrirons que les pots de miel d'applications web pour le reste de la thèse. Nous séparons principalement les honeypots Web en deux catégories: les pots de miel côté client et les pots de miel côté serveur. Les honeypots côté client (également connus sous le nom de honey-clients) sont des logiciels se comportant comme des navigateurs Web traditionnels qui interagissent avec des serveurs Web distants. L'objectif de ces systèmes est de rechercher, rechercher et analyser de manière proactive des applications Web malveillantes ou compromises. Une façon courante de mettre en œuvre un tel système consiste à instrumenter un navigateur Web existant [NWS+16]. Ces applications instrumentées tentent généralement de détecter des signes d'anomalies lors de la visite d'un site Web, par exemple en effectuant un suivi de certains événements lors de l'exécution de son code JavaScript [CKV10]. Une autre utilisation courante de miel-clients consiste à collecter automatiquement des échantillons de logiciels malveillants sur le Web, en explorant des sites Web malveillants [CGZ+11]. D'autre part, les honeypots côté serveur visent à attirer les attaquants en exposant des services apparemment vulnérables. Les honeypots côté serveur sont divisés en deux classes: les pots de miel à faible interaction et les pots de miel à haute interaction. Les pots de miel à faible interaction sont des systèmes simulés qui ne contiennent pas nécessairement de véritables services. Bien qu'ils soient très pratiques pour la collecte d'informations et la surveillance des attaques entrantes, leur capacité à capturer les détails de l'attaque est assez limitée. Par exemple, ils ne fournissent aucune fonctionnalité de backend ou de système d'exploitation car ils ne sont pas destinés à être réellement exploités. Par conséquent, les données collectées par les honeypots à faible interaction peuvent couvrir les phases de reconnaissance et d'exploitation, mais ne reflètent pas fidèlement le comportement des

attaquants et leurs véritables intentions. Les pots de miel à haute interaction sont utilisés à la place pour démêler les objectifs réels des adversaires. Ce sont de véritables systèmes dans lesquels les attaquants peuvent réellement exploiter une véritable application web et interagir avec le système d'exploitation sous-jacent par la suite. Le cas d'utilisation le plus courant du déploiement d'un tel système consiste à effectuer une surveillance à long terme des étapes de l'attaquant et à inspecter et analyser son comportement. Malgré leurs avantages par rapport aux systèmes simulés, les pots de miel à forte interaction présentent également des défis importants, comme mentionné dans les travaux précédents [CB13a]. En particulier, puisqu'il s'agit de systèmes réels sous le contrôle d'un attaquant, ils peuvent constituer un risque pour d'autres services. La virtualisation en elle-même ne peut pas contenir tous les scénarios possibles, car un serveur Web malveillant ne constitue pas seulement une menace pour lui-même mais aussi pour d'autres parties sur Internet. Puisque cette thèse se concentre sur le suivi des agresseurs et sur l'analyse de leur comportement, les pots de miel à forte interaction sont les mécanismes d'observation naturels adoptés pour cette thèse. Plus tard, nous discuterons plus loin de la façon dont nous surmonterons les défis posés par la gestion des pots de miel pendant de longues périodes et comment nous les adaptons en fonction de nos besoins.

A.0.1 Déclaration de problème

Cette thèse est centrée sur le problème de la collecte et l'analyse du comportement de l'attaquant en utilisant des pots de miel web. Plus spécifiquement, cette dissertation aborde trois problèmes principaux, résumés comme suit:

- Les méthodologies de détection existantes sont incapables de suivre le rythme actuel auquel les sites Web sont compromis. En fait, nous montrerons comment les artefacts clés utilisés par les attaquants peuvent rester non détectés pendant quatre ans. Ainsi, de nouvelles techniques sont nécessaires pour distinguer les sites Web potentiellement dangereux des sites Web bénins, de manière simple et automatisée.

- Grâce aux efforts des chercheurs précédents, il est désormais de notoriété publique que les sites Web sont exploités sur le Web. Cependant, la partie isolée du Web (aussi connue sous le nom de Dark Web) n'a pas reçu beaucoup d'attention de la part de la communauté de la sécurité en termes d'analyse de son paysage d'attaques. Comme les cybercriminels ont adopté le Dark Web comme plate-forme pour mener leurs activités illégales, y compris l'hébergement de logiciels malveillants [O'N] et l'exploitation de réseaux de robots [Bro10], it is still unclear how, il est difficile de savoir comment les adversaires mènent des attaques contre les services cachés hébergés sur ces réseaux privés.

- À ce jour, les chercheurs ont principalement concentré leur attention sur les techniques d'étude du code malveillant côté client, et n'ont examiné que récemment le code côté serveur du point de vue de l'analyse statique. Pendant ce temps, les équipes de réponse aux incidents doivent analyser les journaux du serveur Web et désobstruer manuellement les scripts pour tenter de comprendre les actions qui ont été effectuées dans le cadre d'une attaque réussie. Un tel processus d'analyse prend beaucoup de temps et est sujet aux erreurs - et il pourrait bénéficier de techniques automatisées similaires à celles que nous utilisons aujourd'hui pour analyser des échantillons binaires malveillants.

A.0.2 Contributions

Pour aborder les problèmes présentés dans la section 1.1, les contributions suivantes sont présentées dans cette thèse.

- Au chapitre 3, nous proposons pour la première fois une technique automatisée pour extraire et valider les Indicateurs de Compromis (IOC), qui sont des artefacts médico-légaux utilisés comme des signes qu'un système a été compromis par une attaque ou qu'il a été infecté. un logiciel malveillant particulier, pour des applications Web. Nous y parvenons en analysant les informations recueillies par un honeypot à forte interaction. Nos expériences montrent que notre système est capable de générer automatiquement des indicateurs de compromis web qui ont été utilisés par des attaquants pendant plusieurs mois (et parfois des années) dans la nature sans être détectés. Jusqu'à présent, ces scripts apparemment inoffensifs ont pu rester sous le radar des méthodologies de détection existantes - bien qu'ils aient été hébergés pendant longtemps sur des sites Web publics.

- Nous essayons de comprendre si la nature et le volume des attaques web ont un parallèle dans le Dark Web par rapport au Web traditionnel du chapitre 4. En particulier, en déployant un honeypot à forte interaction dans le réseau Tor pendant une période de sept mois, nous avons mené une étude de mesure du type d'attaques et du comportement des attaquants qui affectent ce coin encore relativement inconnu du Web. Nos résultats montrent que les applications Web sur Dark Web peuvent recevoir des attaques automatisées à partir du Web Surface avec l'aide de passerelles Tor qui agissent comme un service proxy. De plus, nous avons constaté que le comportement de l'attaquant implique plus d'activité manuelle au lieu de tirer parti des bots automatisés.

- Enfin, nous présentons le premier sandbox d'analyse de code PHP au Chapitre 5. Notre système se compose d'un interpréteur PHP instrumenté et d'un composant de relecture qui extrait automatiquement les requêtes HTTP requises du serveur Web cible et les utilise pour stimuler le code côté serveur imitant exactement l'action de l'attaquant. Cette combinaison permet une vue sans précédent sur toutes les étapes effectuées lors d'une attaque Web, qui sont capturées par notre sandbox et résumées dans un rapport clair. Nous avons validé notre système en utilisant un grand ensemble de données de plus de 8.000 sessions d'attaque réelles, et nous discutons de nos résultats et distillons quelques idées clés sur le comportement des attaques Web.

A.0.3 Organisation de ce manuscrit

Dans cette thèse, nous collectons et analysons le comportement des pirates et son impact sur les applications web et les serveurs web en utilisant des honeypots web côté serveur à forte interaction. Nous menons également des expériences pour évaluer la nature, le volume et le résultat des attaques Web. Le reste de la dissertation est organisé comme suit:

Chapitre 2 - Travaux connexes

Le chapitre 2 donne un aperçu de l'état de l'art dans ce domaine. Le texte fournit des informations sur les approches côté client, qui sont principalement pertinentes pour les chapitres 3 et 5, car elles couvrent la détection automatique des applications Web malveillantes et l'analyse des codes malveillants dans la nature. D'un autre côté, si les travaux précédents sur les pots de miel à haute interaction servent de toile de fond pour tous les chapitres suivants, ils sont particulièrement pertinents pour le chapitre 5, en raison de la technique dynamique d'analyse côté serveur pour étudier les attaques web. Enfin, le travail connexe se termine par un aperçu des études antérieures axées sur la mesure des caractéristiques du Dark Web, ce qui motive nos expériences présentées au chapitre 4.

Chapitre 3 - Indicateurs Web de compromis

Dans ce chapitre, nous présentons l'utilisation de WIOC (Web Indicators of Compromise) et une technique automatisée pour les extraire automatiquement des machines compromises. Le chapitre commence à partir de l'observation, dérivée de plusieurs années de fonctionnement d'un honeypot d'application Web, que des composants à l'apparence innocente peuvent être utilisés comme un levier pour localiser les pages compromises. Nous expliquons ensuite comment ces éléments peuvent être utilisés comme WIOC et intro- duire des cas d'utilisation possibles. Nous décrivons ensuite les caractéristiques que nous avons identifiées pour distinguer les indicateurs valides des indicateurs non valides et présentons l'évaluation réalisée sur un certain nombre d'expériences en direct. Le travail basé sur ce chapitre a été publié dans la 25ème conférence internationale de World Wide Web (WWW) en 2016 [CBB16].

Chapitre 4 - Attaque le paysage dans le sombre Web

Ce chapitre décrit la conception et le déploiement d'un honeypot à forte interaction dans le réseau Tor pendant une période de sept mois. Notre objectif était de comprendre si les menaces auxquelles les applications Web traditionnelles sont exposées ont un parallèle dans le Dark Web. En particulier, le chapitre discute des principales différences, en termes d'avantages et de désavantages, entre le déploiement et la maintenance d'un pot de miel sur le Web traditionnel et l'utilisation d'une infrastructure similaire à celle d'un service caché de Tor. Nous détaillons les différentes stratégies publicitaires et leur impact sur la collecte des données et discutons enfin des résultats de nos expériences. Le travail présenté dans ce chapitre a été publié dans le 32ème symposium ACM SIGAPP sur l'informatique appliquée (SAC) en 2017 et a remporté le prix du meilleur article pour la piste du logiciel système et de la sécurité [CBB17].

Chapitre 5 - Analyse automatique des attaques Web à l'aide d'un sandbox PHP

Dans ce chapitre, nous présentons une nouvelle approche dynamique pour analyser les applications Web côté serveur malveillantes. Nous expliquons d'abord les défis de la compréhension des attaques contre les applications Web et comment les techniques d'analyse statiques et manuelles actuelles peuvent prendre beaucoup de temps et provoquer des erreurs. Cela sert de motivation pour le recours à l'analyse dynamique et la conception d'un sandbox dédié pour l'analyse des attaques web côté serveur. Le chapitre se termine par l'élaboration des résultats obtenus en rejouant plus de 8 000 attaques, ce qui nous a aidé à démontrer comment le comportement des attaquants peut différer en fonction de l'environnement cible.

Chapitre 6 - Conclusions

Enfin, nous terminons la thèse en résumant les chapitres précédents, en passant en revue leurs principales contributions, en esquissant les futurs travaux possibles dans la région.

List of Publications

Conference and Journal Publications

1. O. Catakoglu, M. Balduzzi, D. Balzarotti Automatic Extraction of Indicators of Compromise for Web Applications. In *25th International World Wide Web Conference (WWW 2016)*

2. O. Catakoglu, M. Balduzzi, D. Balzarotti Attacks Landscape in the Dark Side of the Web. In *ACM Symposium On Applied Computing (SAC 2017)*

3. G. Pellegrino, O. Catakoglu, D. Balzarotti, C. Rossow Uses and Abuses of Server-Side Requests. In *Research in Attacks, Intrusions and Defenses (RAID 2016)*

Bibliography

[201] Black Hat USA 2014, *You Don't Have to be the NSA to Break Tor: Deanonymizing Users on a Budget*. 28

[acu] *Acunetix Ltd, Web Vulnerability Scanner*, http://www.acunetix.com/vulnerability-scanner/, Accessed: 2016-09-26. 70

[ahm] *Ahmia*, https://ahmia.fi/, Accessed: 2016-09-26. 61

[Aka17] Akamai, *akamai's [state of the internet] / security, q1 2017 report*, 2017. 11, 99

[AKM13] Luca Allodi, Vadim Kotov, and Fabio Massacci, *Malwarelab: Experimentation with cybercrime attack tools.*, CSET, 2013. 26

[AYY+17] Mitsuaki Akiyama, Takeshi Yagi, Takeshi Yada, Tatsuya Mori, and Youki Kadobayashi, *Analyzing the ecosystem of malicious url redirection through longitudinal observation from honeypots*, Computers & Security **69** (2017), no. Supplement C, 155 – 173, Security Data Science and Cyber Threat Management. 23

[BDM10] Alberto Bartoli, Giorgio Davanzo, and Eric Medvet, *A framework for large-scale detection of web site defacements*, ACM Transactions on Internet Technology (TOIT) **10** (2010), no. 3, 10. 23

[BKV13] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna, *Delta: Automatic identification of unknown web-based infection campaigns*, Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (New York, NY, USA), CCS '13, ACM, 2013, pp. 109–120. 23

[BKV15] ———, *Meerkat: Detecting website defacements through image-based object recognition*, Proceedings of the 24th USENIX Conference on Security Symposium (Berkeley, CA, USA), SEC'15, USENIX Association, 2015, pp. 595–610. 24

[Bro10] Dennis Brown, *Resilient botnet command and control with tor*. 16, 55, 106

[car] *Tor Project. Did the FBI Pay a University to Attack Tor Users?*, https://blog.torproject.org/blog/did-fbi-pay-university-attack-tor-users. 28

[CB13a] Davide Canali and Davide Balzarotti, *Behind the scenes of online attacks: an analysis of exploitation behaviors on the web*, 20th Annual Network & Distributed System Security Symposium (NDSS 2013), 2013, pp. n–a. 13, 15, 26, 56, 57, 58, 61, 64, 67, 75, 77, 78, 92, 101, 105

[CB13b] Davide Canali and Davide Balzarotti, *Behind the scenes of online attacks: an analysis of exploitation behaviors on the web*, Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS), NDSS 13, January 2013. 35, 37, 40, 46

[CBB16] Onur Catakoglu, Marco Balduzzi, and Davide Balzarotti, *Automatic extraction of indicators of compromise for web applications*, Proceedings of the 25th International Conference on World Wide Web, WWW '16, 2016. 18, 35, 56, 108

[CBB17] Onur Catakoglu, Marco Balduzzi, and Davide Balzarotti, *Attacks landscape in the dark side of the web*, Proceedings of the 32nd Annual ACM Symposium on Applied Computing (SAC), SAC 17, ACM, April 2017. 19, 55, 109

[CBC+17] Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli, *Deltaphish: Detecting phishing webpages in compromised websites*, pp. 370–388, Springer International Publishing, Cham, 2017. 23

[CBMR] Vincenzo Ciancaglini, Marco Balduzzi, Robert McArdle, and Martin Rösler, *Below the Surface: Exploring the Deep Web [Technical Report]*, http://www.deepweb-sites.com/wp-content/uploads/2015/11/Below-the-Surface-Exploring-the-Deep-Web.pdf. 28, 55

[CCVK11] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel, *Prophiler: a fast filter for the large-scale detection of malicious web pages*, Proceedings of the 20th international conference on World wide web, ACM, 2011, pp. 197–206. 35

[CGZ+11] Kevin Zhijie Chen, Guofei Gu, Jianwei Zhuge, Jose Nazario, and Xinhui Han, *Webpatrol: Automated collection and replay of web-based malware scenarios*, Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ACM, 2011, pp. 186–195. 15, 27, 104

[CKV10] Marco Cova, Christopher Kruegel, and Giovanni Vigna, *Detection and analysis of drive-by-download attacks and malicious javascript code*, Proceedings of the 19th International Conference on World Wide Web (New York, NY, USA), WWW '10, ACM, 2010, pp. 281–290. 15, 24, 25, 104

[CWS17] CWSandbox, *Understanding The Sandbox Concept of Malware Identification*, 2017. 77

[DMKS+14] Giancarlo De Maio, Alexandros Kapravelos, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna, *Pexy: The other side of exploit kits*, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2014, pp. 132–151. 26

[EAM+15] Birhanu Eshete, Abeer Alhuzali, Maliheh Monshizadeh, Phillip A Porras, Venkat N Venkatakrishnan, and Vinod Yegneswaran, *Ekhunter: A counter-offensive toolkit for exploit kit infiltration.*, 2015. 26

[EV14] Birhanu Eshete and V. N. Venkatakrishnan, *Webwinnow: Leveraging exploit kit workflows to detect malicious urls*, Proceedings of the 4th ACM Conference on Data and Application Security and Privacy (New York, NY, USA), CODASPY '14, ACM, 2014, pp. 305–312. 24

[EVW11] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam, *Malicious website detection: Effectiveness and efficiency issues*, SysSec Workshop (SysSec), 2011 First, IEEE, 2011, pp. 123–126. 75

[FB] Thomas Fox-Brewster, *Tor Hidden Services And Drug Markets Are Under Attack, But Help Is On The Way*, http://www.forbes.com/sites/thomasbrewster/2015/04/01/tor-hidden-services-under-dos-attack/. 29

[FCKV09] Sean Ford, Marco Cova, Christopher Kruegel, and Giovanni Vigna, *Analyzing and detecting malicious flash advertisements*, Proceedings of the 2009 Annual Computer Security Applications Conference (Washington, DC, USA), ACSAC '09, IEEE Computer Society, 2009, pp. 363–372. 24

[GL09] Salvatore Guarnieri and V Benjamin Livshits, *Gatekeeper: Mostly static enforcement of security and reliability policies for javascript code.*, USENIX Security Symposium, vol. 10, 2009, pp. 78–85. 75

[Goo15a] Google Inc., *Safe Browsing API*, https://developers.google.com/safe%2Dbrowsing/, 2015. 35, 42

[Goo15b] ———, *VirusTotal*, https://www.virustotal.com, 2015. 42

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten, *The weka data mining software: An update*, SIGKDD Explor. Newsl. **11** (2009), no. 1, 10–18. 46

[HKB16] Xiao Han, Nizar Kheir, and Davide Balzarotti, *Phisheye: Live monitoring of sandboxed phishing kits*, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 1402–1413. 27

[HYH+04] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo, *Securing web application code by static analysis and runtime protection*, Proceedings of the 13th International Conference on World Wide Web (New York, NY, USA), WWW '04, ACM, 2004, pp. 40–52. 83

[HYL13] Adam Kliarsky Hun-Ya Lock, *Using IOC (Indicators of Compromise) in Malware Forensics*, Tech. report, SANS, 2013. 35

[IBC+12] Luca Invernizzi, Stefano Benvenuti, Marco Cova, Paolo Milani Comparetti, Christopher Kruegel, and Giovanni Vigna, *Evilseed: A guided approach to finding malicious web pages*, Proceedings of the 2012 IEEE Symposium on Security and Privacy (Washington, DC, USA), SP '12, IEEE Computer Society, 2012, pp. 428–442. 21, 35

[IHF08] Ali Ikinci, Thorsten Holz, and Felix C Freiling, *Monkey-spider: Detecting malicious websites with low-interaction honeyclients.*, Sicherheit, vol. 8, 2008, pp. 407–421. 35

[JKK06a] N. Jovanovic, C. Kruegel, and E. Kirda, *Pixy: a static analysis tool for detecting web application vulnerabilities*, 2006 IEEE Symposium on Security and Privacy (S P'06), May 2006, pp. 6 pp.–263. 83

[JKK06b] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda, *Pixy: A static analysis tool for detecting web application vulnerabilities*,

Security and Privacy, 2006 IEEE Symposium on, IEEE, 2006, pp. 6–pp. 26

[Joh17] Johnny Long, *Hackers For Charity*, 2017. 89

[JSU] JSUNPACK, *A Generic JavaScript Unpacker*. 77

[JYX+11] John P John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martín Abadi, *Heat-seeking honeypots: design and experience*, Proceedings of the 20th international conference on World wide web, ACM, 2011, pp. 207–216. 37, 58, 67

[KAL+15] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas, *Circuit fingerprinting attacks: Passive deanonymization of tor hidden services*, 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 287–302. 28

[KKK+17] Kyungtae Kim, I Luk Kim, Chung Hwan Kim, Yonghwi Kwon, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu, *J-force: Forced execution on javascript*, Proceedings of the 26th International Conference on World Wide Web (Republic and Canton of Geneva, Switzerland), WWW '17, International World Wide Web Conferences Steering Committee, 2017, pp. 897–906. 25

[KLZS12] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, *Rozzle: Decloaking internet malware*, 2012 IEEE Symposium on Security and Privacy, May 2012, pp. 443–457. 24, 25

[KM] Vadim Kotov and Fabio Massacci, *Anatomy of exploit kits*, Engineering Secure Software and Systems **7781**, 181–196. 26

[Kor06] Jesse Kornblum, *Identifying almost identical files using context triggered piecewise hashing*, Digital Investigation **3, Supplement** (2006), no. 0, 91 – 97. 42

[KSC+13] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna, *Revolver: An automated approach to the detection of evasive web-based malware*, Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13) (Washington, D.C.), USENIX, 2013, pp. 637–652. 24, 25

[Lew] Sarah Jamie Lewis, *OnionScan Report June 2016 - Snapshots of the Dark Web*, https://mascherari.press/onionscan-report-june-2016/. 28

[Man15] Mandiant, *OpenIOC – An Open Framework for Sharing Threat Intelligence*, http://www.openioc.org, 2015. 35

[MAS16] Sally M Mohamed, Nashwa Abdelbaki, and Ahmed F Shosha, *Digital forensic analysis of web-browser based attacks*, Proceedings of the International Conference on Security and Management (SAM), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016, p. 237. 27

[MC09] Tyler Moore and Richard Clayton, *Evil searching: Compromise and recompromise of internet hosts for phishing*, Financial Cryptography and Data Security, Springer, 2009, pp. 256–272. 23

[Mea15] Meanpath, *Meanpath Web Search API*, https://meanpath.com/, 2015. 41

[MKC15] Srdjan Matic, Platon Kotzias, and Juan Caballero, *Caronte: Detecting location leaks for deanonymizing tor hidden services*, Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, ACM, 2015. 28

[mod] *ModSecurity: Open Source Web Application Firewall*, https://www.modsecurity.org/, Accessed: 2016-09-26. 60

[MSSV09a] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker, *Beyond blacklists: Learning to detect malicious web sites from suspicious urls*, Proceedingsof theSIGKDD Conference. Paris,France, 2009. 22

[MSSV09b] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker, *Beyond blacklists: learning to detect malicious web sites from suspicious urls*, Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 1245–1254. 75

[MSSV11] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker, *Learning to detect malicious urls*, ACM Trans. Intell. Syst. Technol. **2** (2011), no. 3, 30:1–30:24. 22

[mWBJ+06] Yi min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King, *Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities*, In NDSS, 2006. 24

[Naz09] Jose Nazario, *Phoneyc: A virtual client honeypot.*, LEET **9** (2009), 911–919. 75

[NIK+12] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens,

and Giovanni Vigna, *You are what you include: large-scale evaluation of remote javascript inclusions*, Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 736–747. 22

[NPAA15] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad, *Webwitness: Investigating, categorizing, and mitigating malware download paths.*, USENIX Security Symposium, 2015, pp. 1025–1040. 27

[NPLN14] Christopher Neasbitt, Roberto Perdisci, Kang Li, and Terry Nelms, *Clickminer: Towards forensic reconstruction of user-browser interactions from network traces*, Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 1244–1255. 27

[NWS+16] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder, *A survey on honeypot software and data analysis*, arXiv preprint arXiv:1608.06249 (2016). 15, 104

[O'N] Patrick Howell O'Neill, *Bank thieves are using Tor to hide their malware [News]*, http://www.dailydot.com/crime/bank-malware-tor2web/, Accessed: 2016-09-26. 16, 55, 106

[Phi15] PhishTank, *PhishTank Website*, http://www.phishtank.com/, 2015. 35

[PLZ+16] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel, *Website fingerprinting at internet scale*, Proceedings of NDSS 2016, 2016. 28

[PMM+07] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu, et al., *The ghost in the browser analysis of web-based malware*, Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, 2007, pp. 4–4. 22

[RLZ09] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn, *Nozzle: A defense against heap-spraying code injection attacks*, Proceedings of the 18th Conference on USENIX Security Symposium (Berkeley, CA, USA), SSYM'09, USENIX Association, 2009, pp. 169–186. 24, 25

[SASC10] Jack W. Stokes, Reid Andersen, Christian Seifert, and Kumar Chellapilla, *Webcop: Locating neighborhoods of malware on the web*,

Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More (Berkeley, CA, USA), LEET'10, USENIX Association, 2010, pp. 5–5. 22

[SC14] Kyle Soska and Nicolas Christin, *Automatically detecting vulnerable websites before they turn malicious*, Proc. USENIX Security, 2014. 22

[SDA+16a] Oleksii Starov, Johannes Dahse, Syed Sharique Ahmad, Thorsten Holz, and Nick Nikiforakis, *No honor among thieves: A large-scale analysis of malicious web shells*, Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2016, pp. 1021–1032. 26, 87

[SDA+16b] ———, *No honor among thieves: A large-scale analysis of malicious web shells*, Proceedings of the 25th International Conference on World Wide Web, WWW '16, 2016. 56

[SKV13] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna, *Shady paths: Leveraging surfing crowds to detect malicious web pages*, Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 133–144. 35

[SLZ16] B. Stock, B. Livshits, and B. Zorn, *Kizzle: A signature compiler for detecting exploit kits*, 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2016, pp. 455–466. 24, 25

[SN] Amirali Sanatinia and Guevara Noubir, *Honions: Towards detection and identification of misbehaving tor hsdirs*, https://www.securityweek2016.tu-darmstadt.de/fileadmin/user_upload/Group_securityweek2016/pets2016/10_honions-sanatinia.pdf. 28

[SRBS17] Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos, *The onions have eyes: A comprehensive structure and privacy analysis of tor hidden services*, Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017, pp. 1251–1260. 29

[SS02] David Scott and Richard Sharp, *Abstracting application-level web security*, Proceedings of the 11th International Conference on World Wide Web (New York, NY, USA), WWW '02, ACM, 2002, pp. 396–407. 83

[Ste10] Stefan Esser , *Evalhook - Decoding a User Space Encoded PHP Script*, 2010. 77

[Sym17] Symantec, *Internet security threat report*, apr 2017. 11, 99

[TACB16] Flavio Toffalini, Maurizio Abba, Damiano Carra, and Davide Balzarotti, *Google Dorks: Analysis, Creation, and new Defenses*. 12, 14, 64, 101, 103

[TSOM16] Teryl Taylor, Kevin Z Snow, Nathan Otterness, and Fabian Monrose, *Cache, trigger, impersonate: Enabling context-sensitive honeyclient analysis on-the-wire.*, NDSS, 2016. 25

[Vis] VisAdd, *Advertisement Solution*, `http://visadd.com`. 50

[W3T17] W3Techs, *Usage statistics and market share of php for websites*, aug 2017. 76

[WBJ+06] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King, *Automated web patrol with strider honeymonkeys*, Proceedings of the 2006 Network and Distributed System Security Symposium, 2006, pp. 35–49. 24, 35

[WKM+14] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl, *Spoiled onions: Exposing malicious tor exit relays*, International Symposium on Privacy Enhancing Technologies Symposium, Springer, 2014. 28

[ZH13] Peilin Zhao and Steven C.H. Hoi, *Cost-sensitive online active learning with application to malicious url detection*, Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA), KDD '13, ACM, 2013, pp. 919–927. 22

[ZNG14] Jialong Zhang, Jayant Notani, and Guofei Gu, *Characterizing google hacking: A first large-scale quantitative study*, International Conference on Security and Privacy in Communication Systems, Springer, 2014. 58