ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2017-ENST-0039

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

présentée et soutenue publiquement par

Xiao HAN

le 25 Septembre 2017

Mesure et Supervision de la Sécurité du Point de Vue d'un Fournisseur de Services

Directeur de thèse : **Davide BALZAROTTI**

Jury

M. Engin KIRDA,	Professeur, Secure Systems Lab, Northeastern University	Rapporteur
Mme. Isabelle CHRISMENT,	Professeur, Équipe MADYNES, Telecom Nancy, Université de Lorraine	Rapporteur
M. Hervé DEBAR,	Professeur, Réseaux et Services de Télécommunications, Telecom SudParis	Examinateur
M. Nizar KHEIR,	Responsable Programme de Recherche, Thales	Examinateur
M. Adam OUOROU,	Directeur du Domaine de Recherche Confiance & Sécurité, Orange Labs	Examinateur

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

THÈSE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2017-ENST-0039

EDITE - ED 130

ParisTech Ph.D.

Ph.D. Thesis

to obtain the degree of Doctor of Philosophy issued by

TELECOM ParisTech

Specialisation in « Computer Science and Networking »

Publicly presented and discussed by

Xiao HAN

September 25th, 2017

Measurement and Monitoring of Security from the Perspective of a Service Provider

Advisor : **Davide BALZAROTTI**

Committee in charge

Engin KIRDA, Professor, Secure Systems Lab, Northeastern University	Reporter
Isabelle CHRISMENT, Professor, MADYNES Team, Telecom Nancy, Lorraine University	Reporter
Hervé DEBAR, Professor, Networks and Telecommunication Services, Telecom SudParis	Examiner
Nizar KHEIR, Research Program Manager, Thales	Examiner
Adam OUOROU, VP Trust and Security Research, Orange Labs	Examiner

THÈSE

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

ACKNOWLEDGMENTS

First, I would particularly like to thank my advisor, Davide Balzarotti, for his valuable guidance and availability for my Ph.D study even though most time I worked at Orange Labs instead of being at Eurecom. His advice and support helped me during all time of research, and also for the writing of the thesis.

I'm grateful to my supervisor, Nizar Kheir, for his support and for the numerious discussions that we had together throughout my entire Ph.D study. It was a great pleasure to work with him.

I would also like to express my gratitude to Prof. Engin Kirda, Prof. Isabelle Chrisment, Prof. Hervé Debar and Adam Ouorou, VP Trust and Security Research from Orange Labs for accepting to be the committe members for my Ph.D defense.

It's my pleasure to thank the present and past colleagues from the security department of Orange Labs, for their inspirations, encouragement and the coffee breaks. I also want to thank all fellows at Eurecom S3 group for the warm reception each time I stayed at Eurecom. I am also grateful to all of those who have provided the data for the work in this disseratation and Ph.D students that have designed and implemented the previous honeypot system.

My thanks also goes to Aurélien, Ayoub, Clément, Florent and Jean-Baptiste, for the CTF that we have played together.

I also want to thank all my friends in France, for their kindly support.

Special thanks to my wife, Xu, for her love and support.

I owe my deepest gratitude to my parents, for their encouragement, unconditional support during my studies.

This dissertation would not be possible without the funding of Orange Labs and French Ministry of Education.

ABSTRACT

Today Internet connects about half of the world populations, offering a variety of services – such as email, e-commerce, and online banking – which facilitate the life of billions of users. Unfortunately, the highly connected Internet and its increasing number of users have also appealed to attackers who constantly deliver advanced and sophisticated attacks. Meanwhile, defenders struggle to develop new solutions to detect attacks and protect legitimate services and users. In this context, the service providers may play a very important role in securing both their services, their customers, and the final users – a role that has been often neglected or under-estimated in existing security models and solutions.

In this dissertation, we explore several directions a service provider may follow, in addition to merely secure its infrastructure, to provide better security for its customers and also for other Internet users. More precisely, we leverage the valuable information providers have access to in order to measure and monitor a diverse set of security threats, including malware abuses, compromised instances hosting phishing kits, and external web attacks.

Recent anecdotal evidence shows that service providers, in particular cloud service providers, are routinely abused by malware writers. However, little attention has been paid to this phenomenon. We thus present a systematic large-scale study of about one million malware samples, and our results showed an increasing trend in the number of malicious dedicated cloud-based domains from 2008 to 2014. The existing security mechanisms adopted by service providers were insufficient to correctly measure and detect this type of abuses.

In the second part of the dissertation, we look at a provider ability to monitor attacks that are otherwise difficult to study for third-party researchers. In particular, we designed and implemented PhishEye, a system specifically designed to analyze phishing kit in an ethical way, which enabled us for the first time to understand the entire life-cycle of phishing attacks. Our results showed that most of the victims activity takes place in the short period after the phishing kit is installed, and before the security community even discovers its existence.

Lastly, we explore alternative techniques, in particular deception techniques, that a service provider may employ to add an additional layer of security for its customers. We present a comprehensive classification of existing techniques, allowing us to identify open research directions – including the design and modeling of deception techniques, their deployment, and the evaluation of such techniques. Furthermore, we designed and implemented two experiments to evalu-

ate the detection accuracy and the number of false alarms of deception techniques when they were used to protect web applications. In a red teaming experiment with deception techniques enabled, our approach was able to detect 64% of the participants who have successfully exploited at least one vulnerability. In the other long term experiment conducted in a production environment, zero false alarm had been triggered by 258 different users of the system.

LIST OF PUBLICATIONS

CONFERENCE AND JOURNAL PUBLICATIONS

1. Xiao Han, Nizar Kheir, Davide Balzarotti. *The Role of Cloud Services in Malicious Software: Trends and Insights*. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2015, Milan, Italy.
2. Xiao Han, Nizar Kheir, Davide Balzarotti. *PhishEye: Live Monitoring of Sandboxed Phishing Kits*. Proceedings of the 23rd ACM conference on Computer and communications security (CCS), 2016, Vienna, Austria.
   (Best European Student Paper Award)
3. Xiao Han, Nizar Kheir, Davide Balzarotti. *Deception Techniques in Computer Security: a Research Perspective*. ACM Computing Surveys (CSUR). *(Under submission)*
4. Xiao Han, Nizar Kheir, Davide Balzarotti. *Evaluation of Deception-based Web Attacks Detection*. Proceedings of the 4th ACM Workshop on Moving Target Defense. ACM, 2017.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Internet has rapidly evolved from a small regional network that interconnected a few academic and military institutions to a highly connected global network offering a variety of services – such as the World Wide Web (WWW), electronic mails, telephony, and file sharing. In 2017 [124] it was estimated that over 49% of the world population used Internet for various aspects of their life, including social interactions, electronic business, and telecommunication.

The first commercial Internet Service Providers (ISPs) appeared in the early 1990s to provide Internet access along with a limited number of services. Today, this offer has broaden to include a multitude of providers offering payment services, cloud computing, and online storage solutions for customer ranging from large enterprise to individual end users. To cope with this diversity of these offers, in this dissertation the term of *service providers* is not only used to describe ISPs, but also to refer to application, website, and public cloud service providers.

Unfortunately, the rapid expansion of Internet and its increasing number of users have also appealed to malicious users. Miscreants routinely target lucrative services (e.g., online banking), organization websites, and personal computer and smartphone, mostly looking for a financial gain. To reach this goal, cyber criminals often resort to a combination of zero day vulnerabilities, sophisticated malicious software (malware), phishing attacks, and web-based exploits. This has resulted in a continuous arm race with the security community engaged in a continuous struggle to develop new solutions to detect attacks and protect legitimate services. In this context, the service providers play a very important role in securing both the services and their users, a role that has been often neglected or under-estimated in existing security models and solutions.

1.1 CURRENT SECURITY MODEL

The security guarantees that are offered by each service provider may greatly vary depending on the type of service offered, its scale and also on country's regulations where the provider resides. This diversity of conditions makes it almost impossible to compare the security of different service providers. However, even though sometimes the security model is not clearly defined, service providers usually adhere to a generic *shared responsibility model*, first proposed by Amazon Web Services (AWS) [10] in 2014. Figure 1.1 illustrates a simplified

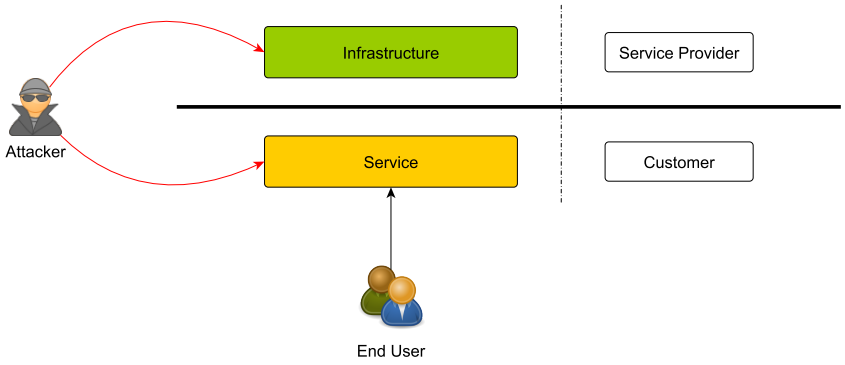


Figure 1.1 – Shared responsibility model

version where we distinguish four main actors: the *customer* who run a legitimate service that leverages the *service provider* infrastructure, the *end user* of this service and the *attacker* who may target both the infrastructure of the service provider and the legitimate service.

The shared responsibility model defines the security measures that the service provider and the customer may agree upon and operate respectively. The provider is responsible for managing the security of the infrastructure together with all the back-end software that supports the customer service. For instance, the service provider should protect the infrastructure against distributed denial of service attacks, and patch the known software flaws and vulnerabilities. On the other hand, the customer needs to take the necessary security measures to protect the running application of the end users, which are considered so far to be out of reach for the service provider.

While simple to understand, this model also presents some limitations. One straightforward example is the case of a miscreant who abuses a vulnerable customer service to perform malicious activities that threaten the security of other users on the Internet. In this scenario, the current shared responsibility model attributes the responsibility to the customer running this service. Yet, the provider is without doubt also responsible because its infrastructure is in fact abused to perform malicious activities. This phenomenon has further been confirmed by our study in Chapter 3. The shared responsibility model offloads the security burdens from the service provider to the customer, which is unfortunate as the provider has a vantage point to provide security measures and to better monitor the security and finally protect the customer.

1.2 THESIS OBJECTIVE

Currently, under the shared responsibility model a service provider has little incentive to provide proactively better security for its customers. This is confirmed by a recent study on web service providers [36], in which the authors found that most providers failed to detect even the most obvious signs that the customers applications had been compromised.

In this dissertation we want to explore what a service provider could do, in addition to merely focus on securing its infrastructure, to take advantage of its privileged position between the end users (or the attacker) and the target service, in order to provide better security.

For instance, a service provider has access to valuable information that are unavailable to its customers, including a global view over the entire infrastructure, the network traffic, and also has entire control over the software layers below the customer application. To show how this valuable information is current under-utilized to develop security mechanisms and study security phenomena, we identified three distinct research objectives:

O1. Service providers focus their effort to protect their own infrastructures against external threats. However, little is known about whether the service provider itself can be *abused* for malicious purposes – i.e., in the special case in which the role of the attacker and of the customer are played by the same actor. For instance, the customer may abuse a service to attack other machines on the Internet or she can host part of her malicious infrastructure on the premise of the service provider. While there are anecdotal evidence that services providers, and in particular cloud service providers, are actually being abused by malware authors, little attention had been paid to this phenomenon and a more rigorous study is needed to measure this emerging phenomenon.

O2. Over the past decade, the research community has put a considerable effort to study different online attacks, such as web exploits or phishing kits. Unfortunately, most previous studies were only able to perform experiments by collecting publicly available data (e.g., either by analyzing already reported infected pages or by crawling the web for signs of malicious content). This dramatically limited the scope of these studies, as the behavior of a service just after it was compromised, and before it was discovered by the security community, remained largely unknown. However, the infrastructure logs and the thorough analysis of incoming connections may provide a better view over certain type of attacks, compared to what has been reported by external researchers. In particular, in this objective we focus on the analysis of the lifetime of phishing kits (which

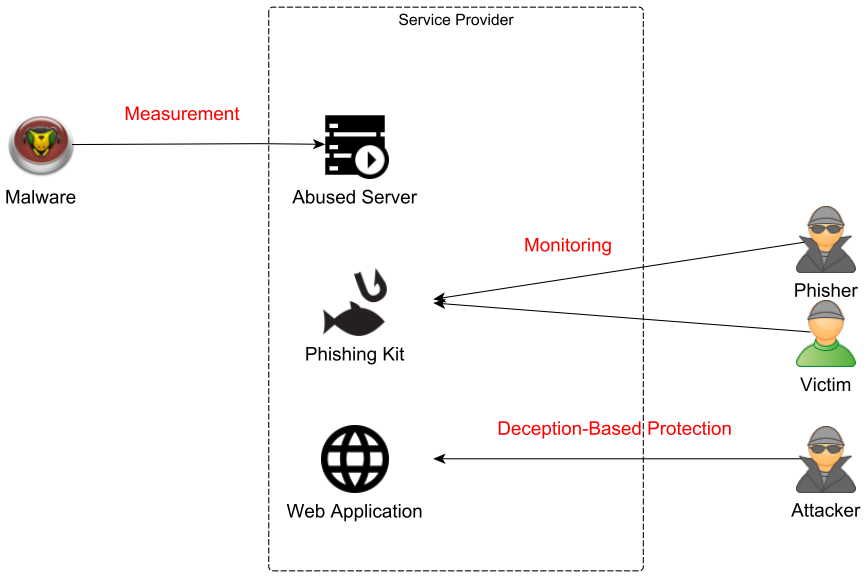


Figure 1.2 – Thesis overview

are very often installed to monetize compromised web applications), to pinpoint the differences between the view of the provider and the view of a third party researcher.

O3. For our final objective, we look at service providers that wants to add an additional layer of security to their customer applications, in particular in the common case of web applications. Traditional security solutions that fit this requirement, such as network monitoring and intrusion detection systems, generates too many false alarms or are unable to cope with advanced, previously-unknown attacks. Therefore, we want to investigate if *deception techniques* would be a better choice in this setting, and what are the scientific challenges that still prevent these solutions to be deployed on a large scale.

The goal of this dissertation is to advance the state of the art along these three different objectives, by conducting measurements and designing analysis and protection systems from the perspective of a service provider.

1.3 THESIS OVERVIEW

Motivated by the three objectives presented above, this dissertation presents a number of techniques that leverage the vantage point of a service provider to *measure the abuses* (Objective **O1.**), *monitor the compromise* (Objective **O2.**) and *enhance the protection* (Objective **O3.**) of online services (as illustrated in Figure 1.2).

For the first objective, we focused our effort on cloud services providers – as they are one of the most prominent type of service providers and their nature make them more prone to possible abuses. Cloud services offers additional advantages such as greater resiliency, hypervisor protection against network attacks, low-cost disaster recovery, on-demand security controls, real time detection of system tampering and rapid re-constitution of services [175]. Many of these advantages make the cloud attractive also to host malicious services and attackers often rely on cloud providers the same way and for the same reasons as legitimate customers [169]. In Chapter 3, we analyze the role cloud services play today in malicious software, presenting a systematic approach to measure the security of public cloud service provider against malware abuses. The results of our study showed that criminals obtained long-sustained malicious activities on public cloud, which makes prominent the need of novel monitoring mechanism for service providers as conjectured in objective **O2.**.

In the second objective we look at a provider ability to monitor attacks that are otherwise difficult to be fully understood. In this context, we focus on phishing attacks that compromise vulnerable websites hosted on the premise of a service provider to install phishing sites. Phishing attacks remain a major threat for Internet Service Providers (ISPs), cloud service providers, as well as for email providers, as confirmed by the fact that the number of unique phishing sites have reached an all-time record in the second quarter of 2016 [2]. However, previous studies conducted by researchers using external data (i.e., without access to the application logs or network traffic) have failed to monitor the entire life-cycle of phishing attacks. The work in this dissertation proposes a novel approach for an application service provider to monitor phishing attacks in order to gain an insight on the different actors and their behaviors, as depicted in Chapter 4.

In our final objective we explore alternative techniques a service provider can adopt to protect its customer applications. From a provider perspective, run-time protection is the only viable solution, due to its scalability and the fact that it does not require to touch or modify customers applications. Traditional runtime protection schemes rely either on signature-based application firewall or resort to anomaly detection [112]. Signature-based solutions are efficient but incapable of detecting unknown attacks, while it is challenging for anomaly-based approaches to balance the detection accuracy and the false positive rate – especially in the web domain. Due to these limitations, the work in this dissertation investigates the use of deception techniques that can be integrated obliviously above the target application by the service provider, promising extremely low false

positives rate combined with high detection of known and unknown attacks.

Although deception is not a novel concept, during our effort to summarize existing work on this topic, we found that we still lack a global understanding about deception techniques and their application in computer security. In particular, there is not yet a wide consensus among researchers about what are the mains goals and technical challenges to solve when deception techniques are adopted as a defense mechanism. In particular, the modeling, deployment, and evaluation of deception techniques were scarcely and poorly approached in the literature. To shed light on this topic, Chapter 5 presents a comprehensive survey on the use of deception techniques in computer security.

Finally, while deceptive schemes are well studied in certain domains, this is not the case for web applications. Previous work in the area [92] only studied the use of deception to delay known attacks. In this dissertation, we explore the use of deception to proactively detect web attacks in their early stage. This is an area that is recently attracting a lot of attention in the industry, but it is still largely unexplored from a research perspective and to the best of our knowledge there is no previous work that has designed experiments to evaluate the accuracy and the false positive rate while using deception to enhance the detection of web attacks. We present in Chapter 6 the design and implementation of two preliminary experiments on the application of deception to detect web attacks.

1.4 DOCUMENT OVERVIEW

This dissertation proposes a number of techniques to measure and monitor the security from the perspective of a service provider. Chapter 2 discusses the previous work related to this topic.

In Chapter 3 we propose a systematic approach to measure the role of cloud services in malicious software (Chapter 3). In particular, we investigate the way cyber-criminals abuse public cloud services to host part of their malicious infrastructures. We conducted a large scale analysis of all the malware samples submitted to the Anubis malware analysis system between 2008 and 2014. Our results reveal that cyber-criminals sustain long-lived operations through the use of public cloud resources, either as a redundant or a major component of their malware infrastructures. We also observe that the number of malicious and dedicated cloud-based domains has increased almost 4 times between 2010 and 2013. To understand the reasons behind this trend, we also present a detailed analysis using public DNS records.

Chapter 4 presents a novel approach leveraging the vantage point of a service provider to monitor attacks and compromise against web applications, with a focus on phishing kits (Chapter 4). We propose

a novel technique to sandbox live phishing kits to observe their behavior while protecting the privacy of possible victims. By using this technique, we performed a comprehensive real-world assessment of phishing attacks, their mechanisms, and the behavior of the criminals, their victims, and the security community involved in the process – based on data collected over a period of five months. Our infrastructure allowed us to draw the first comprehensive picture of a phishing attack, from the time in which the attacker installs and tests the phishing pages on a compromised host, until the last interaction with real victims and with security researchers.

Finally, we investigate how a provider can monitor and protect its customers from web attacks, by using a combination of deception techniques – which provide an interesting alternative compared to other traditional security mechanisms.

We start in Chapter 5 by presenting a survey of the current use of deception techniques in computer security, introducing a comprehensive classification of existing solutions. Furthermore, we analyze several open research directions, including the design of strategies to help defenders to design and integrate deception within a target architecture, the study of automated ways to deploy deception in complex systems and, most importantly, the design of new techniques and experiments to evaluate the effectiveness of the proposed deception techniques. Finally, we discuss the limitations of existing solutions and provide insights for further research on this topic.

In Chapter 6 we then present the preliminary design and evaluation of a deception-based web attack detection approach, which aims at detecting attacks at their early stage. During a one day red-team experiment where participants searched for vulnerabilities in a target web application, our system was able to detect 64% of the participants that have successfully exploited at least one vulnerability. We also conducted a long term experiment in a production environment over a period of seven months to evaluate the rate of false alarms. During the time frame of our test, the service was used by 258 different users, generating zero false alerts.

Finally, in Chapter 7 we draw the conclusions and discuss possible future work in the area.

2 RELATED WORK

The work in this dissertation covers the broad area of measurement and monitoring of security, and their application to detect and protect a service provider from a variety of threats – including malware abuse, compromised websites hosting phishing kits, and web attacks.

In this Chapter, we summarize previous studies that are related to the techniques we present in following parts of this thesis. More precisely, this spans three different areas. In section 2.1 we introduce related work on the measurement of malicious use of cloud services. In section 2.2 we discuss existing works that studied and monitored phishing attacks. Lastly, in section 2.3 we review previous surveys and classifications of deception techniques in computer security. Moreover, we overview related work that used deception-based technique in order to protect web application.

2.1 NEFARIOUS USE OF CLOUD SERVICES

The security of cloud services have been extensively studied in the literature. As discussed in [184], many studies have been dedicated to secure the virtual machine monitor in order to provide security for client software and data against exploits and side channels attacks. Other works have proposed to use virtual machine introspection to identify guest operating system [56] or to detect the presence of rootkit in the guest OS [86, 152]. Finally, many papers have proposed solutions to build of secure virtual network domain [34], secure storage [66], and secure boot [64] on cloud machines.

In [3], Aceto et al. surveyed the existing platforms and services for cloud monitoring. Previous work covered various aspects of cloud monitoring, including the performance, service level agreement, quality of services, capacity and resource management, and security. However, the abuse of cloud services remains one of the top nine critical threats to cloud computing [118]. Indeed, the threat of abuse and nefarious use is more of an issue for cloud service providers than cloud consumers. Unfortunately, from the perspective of a service provider, the measurement and monitoring of security is still insufficient to detect and protect against abuses, compromise, and attacks.

Side effects may exist when an attacker abuses cloud services. First, certain abuses may lead to side-channels attacks that leak customer information [153] or that allow an attacker to extract customer private keys [201]. Second, such abuses may be used as an amplifying factor to trigger last stage distributed attacks, for instance by sending a large

number of spam messages [31] or by joining distributed efforts to break cryptography keys [14].

Moreover, this kind of abuse allows miscreants to resort to cloud services to host part of their malware infrastructure [23, 60]. The term "malware" is generally used in the security community to designate unwanted software such as viruses, Trojan horses, ransomware, worms, and botnets – that exhibit harmful behavior to fulfill the attackers' intent [58]. Miscreants usually start by infecting the victim's system either through technical sabotage (e.g. by exploiting vulnerable network services or by performing drive-by download attacks targeting web browsers) or through social engineering methods.

Once the target machine has been infected, modern malware is usually equipped with communication and remote control mechanisms that provide the attacker with full control over the infected host. Consider the example of a bot, which is a type malware that is under the control of a malicious entity, also known as the bot-master. A bot allows the bot-mastere to deliver commands and control at distance the victim's system. In addition, bots may be capable of exfiltrating the victim's confidential data such as credit card number and online banking credentials to a remote server under the control of attacker. This kind of functionality requires specific infrastructure support, including a number of malicious servers to collect the uploaded information and orchestrate the *command and control* (C&C) communication.

To detect malicious C&C servers, previous works have focused on the analysis of the local area network, where they analyzed the spatial-temporal correlation of pre-programmed activities related to C&C, such as coordinated communication and propagation [68]. More recently, researchers shifted their focus to large scale Internet Service Provider (ISP) networks [24]. However, little has been known about the malware infrastructure, and in particular about the role of cloud services to host malicious software.

Prior abuse cases of public cloud providers have attracted a lot of interests in the recent years [23, 60, 77, 182]. For instance, cloud services are listed by Solutionary among the major components of modern cybercrime, and *"attackers seem to use these services the same way and for the same reasons as legitimate customers"* [169]. Despite this popularity, we are aware of only few research studies that managed to evaluate the real extent of this phenomenon.

Hamza et al. [71] presented a survey of possible techniques to abuse cloud services in modern cybercrime. The authors provided interesting insights on the way cyber-attacks were perpetrated from within cloud platforms, including examples such as host hopping attacks and abuse of privileges. However, this survey only focuses on strong attack signals, and does not consider other weak signals that determine possible ways in which the cloud services are used as part of the attackers command and control infrastructures.

In [132], Nappa et al. analyzed drive-by download attacks and exploited servers that were managed by the same organizations. They found that 60% of the service providers that hosted the exploit servers were indeed public cloud service providers. More interestingly, they evaluated the abuse report procedures implemented by public cloud service providers. The authors discovered that out of 19 abuse reports they submitted, only 7 were investigated by cloud providers. Moreover, the authors computed that it takes on average 4.3 days for a cloud provider to take down an exploit server after it has been reported. It is important to note that the authors of this study only focus on drive-by-download attacks that involve cloud services. Although drive-by-download servers constitute a major component of a modern malware infrastructures, in this dissertation we go beyond this unique use case to provide a more comprehensive assessment about the way cloud services are being integrated in malware infrastructures in modern cybercrime. We also try to understand whether clouds constitute core elements of the malware structure, or whether they are only used as redundant or failover components.

Canali et al. [36] proposed an active approach to evaluate the security mechanisms implemented by web hosting providers. They installed vulnerable web services on 22 distinct hosting providers, and triggered multiple attacks to leverage the reaction capabilities of these providers. To test the security mechanisms implemented by cloud service providers, in Chapter 3 we adopt a less intrusive approach where we only observe malware interactions with the cloud. In our study we only focus on Amazon EC2. While this choice may limit the extent of our observations, at the same time eliminate as much as possible the impact of rogue or other hosting providers that do not guarantee minimal security SLA requirements to their users. We believe that focusing only on the biggest cloud providers in terms of market share also shed light on the limits of current security and accountability mechanisms implemented by today's cloud providers.

Wang et al. [188] proposed a system that measures the churn rates in public cloud service providers (e.g. EC2 and Azure) in order to evaluate the efficiency of IP blacklists against cloud-based malicious activities. The authors actively probed the EC2 and Azure IP ranges, and proposed a clustering mechanism that groups together IP addresses implementing the same services. They also observed all web services hosted by cloud providers, spanning both benign and malicious activities. The results of their experiments showed only small amounts of malicious activity (mostly phishing and malware hosting) by comparing data from their system with public blacklists. In Chapter 3 we propose a complementary approach that observes only malware interactions with the cloud in order to leverage the true extent of the malicious activity hosted by public cloud providers.

More recently, Liao et al. [114] analyzed systematically the public cloud repositories that have been abused by miscreants for the purpose of illicit activities. The authors also characterized the effectiveness of long-tail search engine optimization spam that abused cloud hosting services [115]. Tajalizadehkhoob et al. [177] analyzed the data of C&C communication over a period of seven years and found that attackers have little preference for the providers where C&C domains have a relatively long lifespan.

Finally, in a recent paper published in 2017, Lever et al. [111] studied the network activities of about 26M malware samples collected from 2011 to 2015. They were able to confirm a similar trend and obtain results in line with what we present in Chapter 3, but on a larger scale that covered several cloud providers. They also found that potentially unwanted program (PUP) families were the ones among different malware families that used the most cloud infrastructure and content delivery networks (CDNs).

To conclude, the work in this dissertation presents the first systematic measurement of the security of public cloud services with respect to malware abuses. Recent studies focused on the abuses of cloud services in different point of view.

2.2 UNDERSTANDING PHISHING ATTACKS

The scientific literature includes a large number of papers related to phishing attacks. In particular, researchers have proposed many techniques to study, block, and take down phishing sites. However, there is not yet a comprehensive monitoring system that enables the observation of the entire life-cycle of a phishing kit. We classify existing studies into the following three categories: anatomy of phishing, anti-phishing techniques, and evaluation of anti-phishing techniques.

2.2.1 *Anatomy of Phishing*

The work most closely related to this dissertation was performed by Waston et al., who described two phishing incidents [190] that were discovered by the Honeynet Project [172]. Authors describe how phishers behave and the techniques used to set up the phishing sites. One of the two phishing kits received 256 incoming HTTP requests, but apparently no personal data was submitted by the visitors. Yet, authors had to shut down the honeypot because they did not have any system to prevent user data from being stolen by the attackers. Our work presented in Chapter 4 adopts a similar honeypot-based approach but focuses on providing an ethical system to study how real-world phishing attacks are structured. McGrath et al. [120] analyzed the way phishers performed their attacks, the characteristics of phishing links, the domains, and their hosting infrastructure. The

authors also estimate the lifetime of phishing domain names by using periodically collected DNS records. Moore et al. [126] present the evidence that miscreants use search engine ("Google Hacking") to compromise and re-compromise machines, which are further used to host phishing sites. In another work, Moore et al. [129] studied the temporal correlations between spam and phishing websites in order to understand the attackers behavior, and to evaluate the effectiveness of phishing site take-down. Sheng et al. [165] conducted a demographic analysis of victims' susceptibility to phishing attacks and discussed the effectiveness of educational materials.

A few studies have focused on estimating the success rate of phishing emails. Jagatic et al. first reported the success rate of simulated phishing emails [82], while Jakobsson et al. proposed ethical phishing experiments on a popular online auction website [85], in order to measure the success rate of simulated phishing emails. The reported success rate is respectively about 15% and 11% (compared to 9% we found in our study). However, these works measured the success rate based only on *simulated* phishing attacks.

Multiple studies measured the impact of phishing on potential victims. Moore et al. have empirically measured the lifetime of phishing sites and the number of victim responses [125]. Authors retrieved confirmed reports from PhishTank, and then relied on the records generated by Webalizer, a free web server log analysis tool, in case where it was already installed on the compromised websites. However, this tool saves merely the number of hits for a given web page instead of the unique number of visits, which makes the reported results a very rough estimation. The authors were also able to estimate the number of victims for 20 phishing sites, based on the assumption that only users who fall victim of a phishing would be redirected to a confirmation page after they have provided their credentials. Trusteer measured the effectiveness of phishing attacks based on the statistics gathered by a browser plugin over a period of three months [180]. The authors found that in 2009, 45% of observed victims who were connected to a phishing site disclosed their personal credentials. However, their study provides only a partial view of a phishing attack.

Finally, Cova et al. in [49] dissected freely available phishing kits and a few online phishing sites. They analyzed the target organizations, the techniques used to exfiltrate data, and the obfuscation methods implemented in the phishing kits.

2.2.2 *Anti-Phishing Techniques*

Phishing countermeasures have attracted a lot of interest from the research community. The proposed countermeasures can be grouped

into three categories: phishing page detection, blocking, and user training.

Most phishing detection techniques identify phishing pages by building a classifier using different heuristics based on URL features [62, 110] or on the web page content [144, 195, 203]. Some studies aim at detecting and blocking phishing attacks at different stages. For instance, Fette et al. [61] use machine learning to identify and block phishing emails. Several studies propose a browser plugin to protect users from phishing [40, 54, 88]. Another popular approach to block phishing is to compile and distribute blacklists, such as Google Safe Browsing and PhishTank. A number of studies have focused on education against phishing attacks and how to train users to identify phishing [102–104, 139]. Finally, only one study focused on identifying drop email addresses through the use of verified phishing websites from PhishTank and metadata provided by email providers [127]. However, this method introduces a significant latency compared to our approach since public blacklists may not be efficient enough to promptly detect live phishing kits.

2.2.3 *Evaluation of Anti-Phishing Techniques*

In 2006, a number of studies concluded that anti-phishing solutions [202], security indicators [55], and browsers toolbars [196] were ineffective in detecting phishing sites and protecting users.

In 2007, Ludl et al. [119] and Sheng et al. [166] specifically focused on the effectiveness of blacklists to prevent phishing, reaching contradictory results. In the first study, the authors collected online phishing URLs from PhishTank, and tested them against Google Safe Browsing along with the Phish Filter of Microsoft Internet Explorer. This study concluded that the blacklist approach is efficient in protecting users, especially Google which correctly detected almost 90% of the phishing URLs [119]. In the second study, Sheng et al. evaluated five blacklists (including the ones tested by Ludl et al.) with phishing URLs less than 30 minutes old, collected from the University of Alabama Phishing Team's email data repository. The paper found that those blacklists were inefficient as most of them detected less than 20% of the fresh phishing pages [166].

Egelman et al. conducted an empirical study to evaluate the effectiveness of web browser phishing warning and provided some guidances on how to enhance security indicators [59]. In 2014, Gupta et al. [69] evaluated the effectiveness of phishing education pages [102] to determine if they were effective at helping users identify phishing attempts.

To summarize, most existing studies evaluated the effectiveness of anti-phishing techniques only *after* the phishing page was reported publicly or privately. Our study assesses instead the effectiveness of

the blacklist approach right from the beginning of the phishing attacks. To achieve this goal, the work presented in this dissertation proposes a monitoring mechanism that takes advantage of the vantage point of a service provider, which enables the observation of the entire life-cycle of a phishing attack in the wild.

2.3 DECEPTION TECHNIQUES IN COMPUTER SECURITY

In this thesis, we survey existing deception techniques and propose a new classification along four dimensions. We also resort to deception techniques to achieve better protection for web applications. In the following, we first overview existing surveys and classifications on this topic, and leave a more complete overview of deception-related techniques to the survey presented in Chapter 5. Then we summarize previous work that used deception techniques to monitor, detect and mitigate web attacks.

2.3.1 *Previous Surveys*

In [170], Spitzner discussed the use of honeypot and honeytoken technologies as a way to protect against the insider threat. In [157], Rowe discussed the different possibilities to integrate deception in honeypot systems – such as decoy information, delays, fake error messages – and compared them to other opportunities for using deception in real computer systems. Voris et al. [187] discussed the multiple use cases where decoys may be relevant for computer security. Juels and Tech [91] discussed the use honey objects (a generic term they used to refer to multiple types of deception) to improve the security of information systems. Jogdand and Padiya [87] also analyzed IDS solutions and the way they may implement honey tokens, which are indeed a specific type of deception. In [46], Cohen et al. overviewed multiple issues that arise when applying deception in computer security, and further introduced their own framework for deception. The authors also discussed the major challenges to perform practical deceptions using this framework to defend the information system. Cohen also surveyed in [44] the historical and recent uses (until 2005) of honeypots and decoys for information protection, as well as the theory behind deceptions and their limitations.

Our work is different as it surveys the recent contributions, focusing on the technical challenges and evaluations of deception, rather than to study the history of this concept and how it found its way into computer security.

2.3.2 *Previous Classifications*

Early deception classification schemes followed a traditional military deception classification scheme. For example, Cohen [42] examined deception techniques based on the nature of these techniques, including the *"concealment, camouflage, false and planted information, ruses, displays, demonstrations, feints, lies and insight"*. The historical examination revealed that deception was far from being fully exploited in computer security. The same type of taxonomy has also been used in later works [159, 160].

Rowe and Rothstein [159] developed a theory of deception that is inspired from the computational linguistic theory of semantic cases. The semantic case roles may contain participant (agent, beneficiary), space (direction, location), time, causality (cause, purpose), quality (content, value), etc. The authors explained a deception operation as an action that modifies the values of the associated case role. Cohen [44] proposed a model of computer deceptions that groups deceptions by the hierarchical level of the computer at which they are applied, from the lowest hardware level to the highest application level. Information and signals between different levels can either be induced or inhibited in order to implement deception. Similarly, Almeshekah and Spafford [7] classified deceptions based on the system state and components where deception may be deployed. The top-level categories include the functionality of the system (system decisions, software and services, internal data) and state of the system (activities, configurations and performance).

Gartner [149] proposed a four layer deception stack including the network, endpoint, application and data layers. It further analyzed the deception techniques implemented by current commercial products. Both Gartner [149] and Almeshekah and Spafford [8] have examined the possibilities to deploy deception techniques during the different phases of a cyber attack, without any systematic classification.

Most previous classification considered only one dimension, such as the component or the granularity of each technique [7], or the layer where deception is applied [149]. These mono-dimensional classifying fail to capture other aspects of deception that are of equal importance, such as the threats covered by each technique, and the way the deception mechanism can be integrated inside a target system. To take all these different aspects into account, this dissertation introduces a new comprehensive classification system based on four orthogonal dimensions.

2.3.3 *Deception-Based Web Application Protection*

The literature is brimming with approaches to protect web applications [112] against attacks. Nevertheless, current web attacks detection techniques fail to reliably and proactively detect attacks in their early stage. Due to these limitations, complementary solutions such as deception techniques have been recently investigated by the research community [8, 13]. In this dissertation, we focus mainly on those solutions that adopt a deception-based web application protection scheme. We further group existing approaches into two categories, one used to enhance attack detection and the other used for the purpose of attack mitigation.

2.3.3.1 *Attack Detection*

Brewer et al. [30] proposed a web application that integrates decoy links. These links are invisible to normal users, but are expected to be triggered by crawlers and web bots that connect to the application. Similarly, Gavrilis et al. [65] presented a deceptive method that detects denial of service attacks on web services by using fake links hidden in the web page. In the same way, McRae and Vaughn [121] submitted honey accounts which contained decoy URL to phishing sites to track phishers while they viewed the honey accounts.

Another approach to deceive web-based attacks consists of using fake information disguised as web server configuration errors. Only malicious users are expected to manipulate or exploit these errors, which expose them to detection by the system. In this scope, Virvilis et al. [185] introduced honey configuration files, such as `robots.txt`, including fake entries, hidden links, and HTML comments that indicate honey accounts, in order to detect potential attackers. Other studies proposed decoy forms [97] and honey URL parameters [147] that display fake configuration errors in an effort to mislead attackers and protect the target system.

In [6], Almeshekah proposed centralized deceptive server which enables the implementation of deception to protect target web servers. In each web server, the proposed system hooks incoming requests and further sends their metadata toward the centralized server where a decision is taken on whether the system should respond with deception. The author further analyzed the performance overhead introduced by the system.

2.3.3.2 *Attack Mitigation*

Julian [92] proposed to alter the response time of a web-based search engine by injecting random delays in reply to malicious requests in order to confuse an attacker. Anagnostakis et al. [12] introduced *"shadow honeypots"* that extend traditional honeypots with

anomaly-based buffer overflow detection to protect web application. The shadow honeypot is a copy of the target application, with common context and application state. It is used to analyze anomalous traffic and to enhance the accuracy of anomaly detection.

Finally, Araujo et al. [16] presented a technique to transform traditional patches into *"honey-patches"*, designed to remove the vulnerability while at the same deceives the attackers into believing that the attacks have succeeded. On the detection of an attack targeting the known vulnerability of the web server, the system forwards the attacker to an unpatched but isolated instance of the same web server. Authors further extended their honey-patch system with an instrumented compiler that automatically clean credentials in the unpatched version of the web server [15].

The work in this dissertation differs from the above work in that we focus on attack detection instead of attack mitigation. Unfortunately, our survey emphasizes how most of the previous work in deception-based attack detection did not provide experiments and evaluations of the proposed techniques. Therefore, in Chapter 6 we present two experiments we conducted to evaluate the effectiveness and the false positive rate of deception-based web application protection.

3

ROLE OF CLOUD SERVICES FOR MALICIOUS SOFTWARE

In this chapter, we investigate the way cyber-criminals abuse service providers, and in particular public cloud providers, to host part of their malicious infrastructures. This includes using the provider to host exploit servers to distribute malware, C&C servers to manage infected terminals, redirectors to increase anonymity, and drop zones to host stolen data.

Public cloud services have rapidly expanded in the recent years, with almost half of US businesses now using cloud computing in some capacity [48]. The worldwide market of cloud services is projected to grow 18 percent in 2017 to reach 246.8 billion dollars [148]. Public cloud services offer a straightforward *pay-as-you-go* pricing model where users dynamically create virtual machines at will, provide them with public IP addresses and on-demand compute and storage resources, and then delete them without any sustainable cost. Major providers of public cloud services, such as Amazon EC2 [11] and Microsoft Azure [18], also propose scalable services and default configuration options that contributed to the wide adoption of cloud services.

Unfortunately, the rapid growth of cloud services has also attracted cyber-criminals, paving the way to an active underground economy. As a result, Los et al. [118] listed the abuse of cloud services among the top nine critical threats to cloud computing. In fact, public infrastructure-as-a-service clouds provide users with virtually unlimited network, compute, and storage resources. These are coupled with weak registration processes that facilitate anonymity, and so anyone with a valid credit card can easily register and use cloud services. For example, an early case of cloud service abuse was publicly uncovered in 2009, where a Zeus command and control (C&C) server was found to be hosted on Amazon EC2 [60]. More recent examples include the SpyEye banking trojan that was found to be using Amazon S3 storage [23], Android malware that exploited the Google Cloud Message service [182], and more advanced persistent attacks that used Dropbox and Wordpress services as a cover [77]. Despite these multiple examples, we are unaware of any existing study that measures the extent at which public cloud services are being abused by cyber-criminals. Such study would advise the service provider to better understand the threat, different actors involved and finally to implement necessary protection measures. More precisely, we do not know

if cyber-criminals use cloud-based servers only as redundant components of their malware infrastructure, or whether they *specifically* use cloud services to achieve a better sustainability. Besides, we do not know if public clouds add more resilience to malware infrastructures, what is the time it takes to detect a malicious server hosted on a public cloud, as well as the time required to take down this server after it was first discovered.

In this Chapter we present a framework to measure and analyze malicious activity that involves public cloud services. Unlike previous work that actively probed public cloud IP addresses [188] or only use passive DNS records [74], we directly collect malware communications by analyzing the network traffic recorded by the Anubis dynamic analysis system [20]. Anubis is a publicly accessible service that analyzes malware samples in an instrumented sandbox. As part of its analysis, the system also records which domains and IP addresses are contacted by each malware sample, and part of the data that is transferred through the connection. Unfortunately, malware can communicate with the cloud for multiple reasons, including malicious activities but also other innocuous connections which range from simple connectivity checks, to the use of public benign services. This greatly complicated the analysis, and required the development of several heuristics to discard malware samples using public services hosted on the cloud, as this cannot be considered an abuse of the cloud itself.

In our experiments, we analyzed the network communication of over 30 million samples, submitted between 2008 and 2014. Our system identified 1.08 million (roughly 3.6%) that connected to at least one publicly routable Amazon EC2 IP address. These IPs were associated to 12,522 distinct cloud-based domains. Interestingly, we observed that over the same period, only 32,225 samples connected to Microsoft Azure. Due to the relatively low number of samples that interacted with Azure, we only focused our study on Amazon EC2.

3.1 APPROACH

To identify malicious servers hosted on Amazon EC2, we first collected the range of IP addresses assigned to the cloud images, as reported by the Amazon website [1]. Moreover, to account for possible yearly changes, we also retrieved previous versions of the page from the web archive project [2]. We then extracted and analyzed the network traffic generated by all the malicious samples that have been collected and executed in Anubis, a popular malware analysis sandbox [20], over the past six years.

1. http://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html
2. https://archive.org/web/

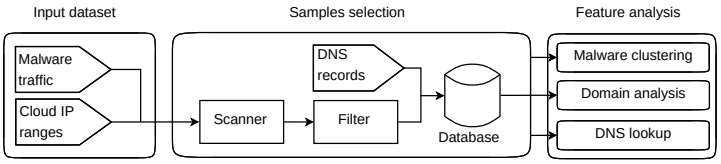


Figure 3.1 – Architecture of our platform

The main goal of our study is to verify the way miscreants make use of cloud services, whether they specifically target cloud infrastructures, and measure the time it takes for the provider to detect and drop malicious services hosted on EC2. To do so, our system tracks all domain names associated with the EC2 IP addresses that were contacted at least once by a malicious sample. Then, it further extracts and analyzes the DNS features and the content of network communications between the malware and the EC2 machines.

A major challenge in our study is that domain names extracted from the Anubis database do not only include dedicated malicious servers, and so we cannot simply mark as suspicious every connection toward a cloud-based IP address. In fact malware often contacts other public and benign cloud-based services, such as IP lookup services, advertisement websites, and URL shortening. These services are not part of the malicious activity and therefore need to be identified and discarded from our subsequent analysis.

On the other hand, real malicious domains may have been *sinkholed* by security organizations at the time the malware was analyzed in Anubis. Malware will be thus redirected towards sinkhole services that are sometimes hosted on EC2, even though the original domains may have not been hosted on the cloud. Our system filters these cases and does not consider them as cloud-related malicious activities. Finally, in our experiments we discovered that many malware samples were *adwares* that leverage pay-per-install (PPI) services hosted on Amazon or other cloud providers. PPI services allow software publishers to pay affiliates by the number of installations that they perform on target machines. Caballero et al. analyze in [33] the *modus-operandi* of PPI services and measure their use for malware distribution. Although the use of PPI services to distribute malware still constitutes a malicious activity, PPI services are not malicious *per-se*, and so they need to be discarded from our dataset as we only focus in this chapter on dedicated malicious services that were hosted on EC2.

3.1.1 *Platform Description*

To setup our experiments, we designed and implemented the platform illustrated in Figure 3.1. Our system consists of two main components: the samples selection and the feature analysis modules. The

first extracts from the Anubis database all malware samples that exhibited at least one network connection towards the Amazon cloud. During the period of our study, we identified 1.08 million malware samples that satisfied this criterion. The samples selection module further discards samples that have contacted benign public services hosted on EC2, and *keeps only dedicated malicious services* as input to the feature analysis module. Finally, the feature analysis module classifies the remaining malware samples and analyzes their dedicated malicious services hosted on cloud.

3.1.1.1 *Samples Selection.*

The samples selection module aims at building a database of malware samples that, during their analysis, connected to malicious services hosted on EC2 – as well as the domain names or IP addresses that were associated with these services.

MALWARE SCANNER: this module first extracts from Anubis all malicious samples that interacted with EC2 machines. We seed this module with the list of publicly routable IP ranges that were associated to the Amazon cloud in the year in which the analysis was performed. During the six years of our study, we identified 1,079,318 distinct samples that connected to EC2.

The first thing we noticed in our experiments is that a large number of samples in our dataset were executables that leveraged pay-per-install (PPI) services hosted on EC2. PPI services have recently emerged as a key component of modern cybercrime and miscreants often refer to these services to outsource the global distribution of their malware. They supply PPI services with malware executables, which in turn charge them for successful installations based on the requested features for the desired victims. PPI service providers operate directly or through affiliate programs. They develop downloaders that retrieve and run the requested software (possibly malware) upon execution on the victim computer.

To identify PPI downloaders in our dataset, we refer to multiple public sources such as PPI forums [178] and public PPI web sites. The main challenge in our case was to identify the different PPI brands, since there are new brands that constantly appear over time. In order to address this challenge, we analyzed the public PPI services that were mostly contacted by the samples in our dataset, and we tried to infiltrate these services by supplying a small program we developed for distribution. By testing and manually reverse engineering the resulting installer we developed a set of 13 distinct network signatures that match the download URLs associated with different families of PPI services. By using these signatures on the malware traffic we could further discard their associated samples in our dataset. As illustrated in Figure 3.2, we were able to discard 1,003,289 PPI down-

PPI Domain name	Samples
getapplicationmy.info	116306
sslsecure1.com	71965
oi-imp1.com	68255
secdls.com	52857
oi-config1.com	43526
ppdserver.com	39434
optimum-installer.com	38777
optimuminstaller.com	35510
leadboltapps.net	31918
xtrdlapi.com	18615

PPI Domain name	Samples
torntv.net	16578
powerpackdl.com	15586
oi-config3.com	15578
webfilescdn.com	14050
torntvz.com	12440
premiuminstaller.com	11879
ppdistro.us	10463
bestringtonesmaker.com	10136
baixakialtcdn2.com	9946
oi-config2.com	9601

Table 3.1 – Top 20 PPI services in our dataset

loaders, which corresponds to up to 93.2% of our initial dataset. Table 3.1 summarizes the top 20 PPI domain names that were contacted by malware in our dataset and the number of samples that were associated with each service.

In addition to PPI downloaders, our dataset also includes benign files that were submitted for analysis in Anubis. In fact Anubis is a public service where Internet users freely submit suspect files for analysis. These files may turn out to be benign files that connect to benign cloud-based services and so they also need to be discarded from our dataset as they do not belong to the malware category. Since our dataset covers a period where the most recent samples are few months old, we use anti-virus (AV) signatures to identify and discard benign samples. We refer to public services such as VirusTotal[3] to scan our dataset, and we consider as benign files all samples that are detected by less than five AV editors. Our dataset finally includes 45,422 confirmed malicious malware samples. The remaining 30,607 samples (2.83% of the initial malware dataset) were discarded as we do not have enough confidence about the malicious nature of these files.

DOMAIN FILTER: The domain filter module further discards from our dataset all domains that are associated with benign cloud-based services. Although these domains supply public Internet services that can be used by malware, they are not part of dedicated malicious services. Out of the initial set of 12,522 distinct EC2-based domain names or IP addresses, the malware scanner discarded 8,619 associated to PPI services or that were also contacted by benign programs. The domain filter classifies the remaining 3,903 domains into four categories, as illustrated in Table 3.2.

3. http://www.virustotal.com

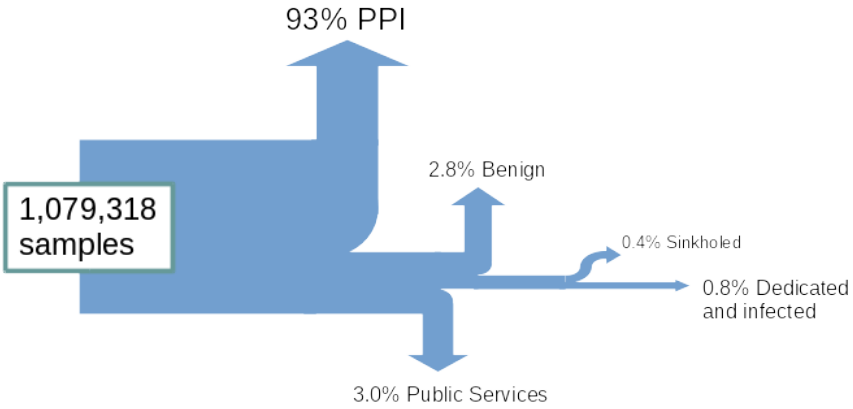


Figure 3.2 – Composition of our malware dataset

Service		Domain Names	Malware Samples
Public Services	Advertising	930	22,216
	File sharing	796	7,657
	Domain redirection	270	479
	Others	211	1,723
Sinkholed		26	4,249
Infected		22	231
Dedicated		1,648	7,884
		N/A	983
Total		**3,903**	**45,422**

Table 3.2 – EC2-based service categories

The first category includes public benign services that were contacted by malware. We found multiple examples in this category, including public IP resolvers (e.g. `hostip.info`), advertising and affiliate services, file sharing (e.g. `dropbox`), URL shortening (e.g. `notlong.com`), and multiple other free services (e.g. `about.me`, `spring.me`). To identify known public services in our dataset, the domain filter leverages multiple sources such as the Alexa list of top domains, public repositories that provide URL shortening services (e.g. `bit.do`) and file sharing [4]. We also refer to AV labels in VirusTotal in order to identify generic adwares. The domain filter module identifies as advertisement services all domains that were only contacted by Adwares samples. To be conservative, these domains were classified by our system into the public services category.

The second category includes domain names that have been sinkholed, and so they were redirected to sinkhole destinations that are hosted on the cloud. EC2 hosts multiple sinkhole destinations that are used to subvert BOT communications with their remote C&C

4. http://online-file-sharing-services-review.toptenreviews.com/

domains. These domains were not originally hosted on EC2, and so they need to be discarded from our dataset. We leverage the `X-sinkole` HTTP header[5] in order to identify sinkhole destinations in our dataset.

The last two categories include both dedicated malware domains and domains that were once infected and temporarily used as part of the malicious infrastructure. The separation between these two categories is more difficult and more prone to errors. Our system relies on multiple empirical observations in order to discriminate between the two cases. First, we assume that dedicated malware machines that were hosted on EC2 more than one year ago have all been detected or abandoned at the time we run our experiments. We show in Section 3.2 that this is a reasonable assumption, consistent with the average lifetime of dedicated malicious domains hosted on EC2. Based on this assumption, the domain filter module actively probes all the domains and if the domain is still in use and points to a populated web page, we classify it as an infected host. Unfortunately, domain name vendors often return a HTML page to sell expired domains. Therefore, to correctly identify these domains, we parsed the HTML response page using multiple keywords (e.g. '`domain expired`', '`domain for sale`') and we removed these domains from the infected domains category. On top of this first heuristic, we also leveraged the history of DNS requests towards expired domains in order to assess the average lifetime of these domains. We use for this purpose DNS records that we extracted from DNSDB, a passive DNS duplication service[6]. In this case, our assumption is that infected domains usually have a longer turnover than other dedicated malicious domains. In other terms, infected domains are expected to appear in DNS records a long time before the associated malware first appears in the wild. Dedicated domains instead, usually appear a short time before the malware is released and go offline a short time after the malware has been detected. By combining these two heuristics we were able to identify 22 infected services hosted on EC2 over the six years of observation. The remaining 1,648 domain names were identified by our system as being associated with dedicated malicious services.

Most of the connections were initiated using a domain name, but 983 malware samples directly connected to EC2-based IP addresses that were hard-coded in the malware itself, without any prior DNS request. To summarize, 8,867 of the 45,422 samples used at least one dedicated server hosted on Amazon EC2. For these, we also analyzed the content of their network communications. Almost 90.3% of these samples used the standard HTTP protocol (either GET, POST,

5. http://www.iss.net/security_center/reference/vuln/HTTP_Malware_XSinkhole.htm

6. https://www.dnsdb.info/

or HEAD methods). Few samples were IRC bots (19 distinct samples) and spam bots (136 distinct samples) that connected to malicious IRC and SMTP servers hosted on EC2. The remaining samples belonged to the Zeus version 3 and the Sality peer to peer (P2P) malware families, and were using the UDP protocol to connect to malicious P2P services hosted on EC2.

3.1.1.2 *Feature Analysis.*

The analysis module processes the output dataset provided by the feature extraction module in order to extract main trends. First, it clusters malware families according to their antivirus labels, in order to figure out whether there exists a general trend towards moving malware infrastructures into the cloud, or whether this phenomenon is limited to some specific malware families. Second, it analyzes the network activity of each malware sample, computing the distribution of IP addresses and the domain flux to tell if miscreants specifically target cloud services, or if they use these services as part of their redundant malware infrastructure. Third, the feature analysis module observes the average duration a dedicated malicious server remains publicly accessible on the cloud. This can be used to estimate how effective are cloud providers in detecting abuse of their services, and whether malware writers sustain long lived malicious activities through the use of public cloud services. The following section provides the details and the main results of our experiments.

3.2 EXPERIMENTS

The dataset provided by the feature extraction module (as described in Table 3.2) allows us to analyze both the malware families that are using EC2 cloud services in some capacity, as well as the distribution and lifetime of malicious domains that are hosted on EC2. Therefore, a first question that we would like to address in this section is whether the use of public cloud services is still limited to a small set of malware families, or whether it can be generalized to different families of malware. A straightforward approach to answer this question is to analyze the 8,867 distinct malware samples that we found to be connecting to dedicated malicious EC2 machines.

Since our dataset includes malware samples that are at least few months old at the time we run our analysis, we believe it is reasonable to use AV labels as a reference to understand and classify our dataset. More complex behavioral clustering mechanisms, as proposed for instance by Bayer et al. [19] and Perdisci et al. [146], could be applied to refine the classification. However, since we only need a broad understanding of the major malware families that use cloud services and we can tolerate few misclassification errors, a simple AV-based solution is better suited for our study.

AV label	# samples
Downloader Fosniw	1249
Worm Vobfus	909
Android DroidAp/SmsSend	634
Downloader Murlo/Renos	567
Backdoor QQRob	528
Downloader Small BKY	208
Delf Downloader	196
Trojan Injector	194
Downloader 8CCBF09D99CF	186
Clicker Agent	172

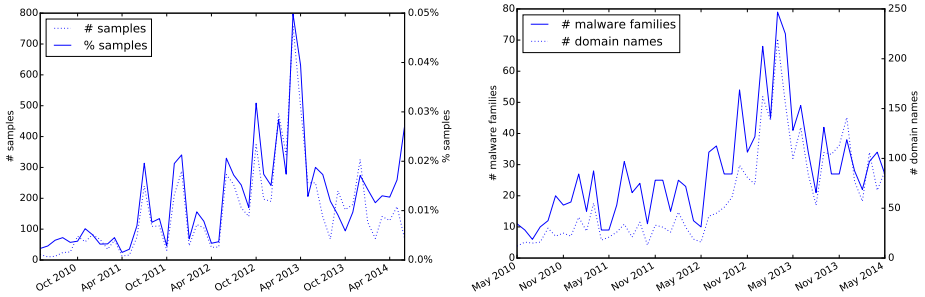
AV label	# samples
Trojan Kryptik	160
Ramnit	129
Downloader Banload/Zlob	128
Trojan Kazy	127
Downloader Virut/Virtob	127
Zbot	117
Malware SoftPulse	108
Downloader Karagany	90
Trojan Krap	89
Downloader Cutwail	80

Table 3.3 – Top 20 malware family

It is well known that different AV vendors assign different labels for the same malware sample. For example, the SpyEye malware can be identified by Kaspersky as `Trojan-Spy.Win32.SpyEyes`, and by McAfee as `PWS-Zbot.gen.br`. To limit the impact of such inconsistencies, we applied a majority voting to assign the labels to our dataset. In order to do so, we pre-process each label by splitting it in multiple elementary keywords according to non-alphanumeric characters We then discarded common prefixes such as `W32`, `Mal` and `Trojan`, as well as Generic malware identifiers, such as `Heur`, `Worm`, `Gen`, and `malware`. To handle malware aliases, we referred to multiple public sources such as the spywareremove website[7] to group together all aliases of a given malware family. For example, the labels `win32.spammy` by Kaspersky and `W32/Sality` by McAfee were identified as aliases for the same sality malware, and therefore grouped as part of the same family.

CLOUD-BASED MALWARE FAMILIES: We mainly focus in this chapter on malware that uses dedicated malicious services hosted on EC2. Therefore, we build clusters of malware families for our dataset including 8,867 distinct samples that belong to this category. Using our approach, we are able to identify 377 distinct malware families. As clearly illustrated in Table 3.3, which provides the list of top 20 malware families, we were not able to identify a predominant malware family that uses dedicated malicious cloud services. More interestingly, our dataset includes malware that uses different topologies, including also decentralized peer-to-peer networks such as the Sality malware. Clearly the use of dedicated malicious cloud services is not limited to a small set of malware families, but it could be generalized to all categories of malware.

7. http://spywareremove.com/



(a) Number and ratio of distinct malware samples that connected to dedicated malicious domains hosted on EC2

(b) Number of distinct malware families that used dedicated EC2-based malicious services and number of their associated malicious domain names

Figure 3.3 – Malware dataset analysis

TIME EVOLUTION: Since the hosting and usage of malicious services on public cloud infrastructures such as EC2 is not limited to specific malware families, our next goal is to identify if there is a clear trend on the amount of malicious software that make use of cloud services. Figure 3.3a illustrates the number of distinct samples that connected to dedicated malicious domains hosted on EC2 during the period of our observation. To account for changes in the overall number of submissions, the figure also shows the percentage of distinct samples compared to the total number of samples submitted to Anubis in the same period. Figure 3.3b shows instead the number of distinct malware families, and the number of their associated malicious domains that were found to be hosted on EC2 over the same period.

On average, the number of malware that uses dedicated cloud-based malicious services has grown by almost 4 times between 2010 and 2013. The overall trend also includes multiple peaks, that after a manual analysis resulted to be associated with multiple instances of malicious servers found to be temporarily hosted on Amazon EC2. While the fast growing number of malware samples that use cloud-based services may appear as a natural consequence of the general increase in the number of malware attacks [80], Figure 3.3a shows that this is not the case and that the ratio between these malware samples and the total number of malware submitted to Anubis has been increasing at the same rate. As illustrated in Figure 3.3b, this trend can be generalized to all malware families, which means there is a growing appetite towards using cloud infrastructures to host malicious servers. This could be due to multiple elements, including the fact that cloud services have been rapidly expanding in the past few years, and the fact that they are ease to access and still lack a strict accountability and control over their hosted machines [100].

3.2.1 *Role of Public Cloud Services in Malware Infrastructures*

In this section we describe the different ways malicious software makes use of public cloud services. In particular, we are interested in understanding whether miscreants specifically target cloud services or whether they use these services as small parts in a much larger redundant infrastructure.

For this purpose, we measured the ratio of remote malicious destinations that were hosted on EC2, compared to all malicious destinations contacted by the malware during the analysis. Then, for those malicious services that were hosted on EC2, we determined if they were hosted on EC2 only as part of a redundant mechanism. In this case, we extracted the DNS requests executed by the malware and we monitored the DNS records history using DNSDB service in order to compute the ratio of IP addresses that belong to the EC2 IP range, compared to all IP addresses associated with that malicious domain in other moment in time. This technique works particularly well in the presence of round-robin DNS and DNS fast-flux techniques that are often adopted by botnet herders. For instance, miscreants can associate different IP addresses with the same malicious domain name, where only some of these IPs may be hosted on EC2.

Figure 3.4 presents the average distribution of the ratio of remote malicious destinations that were hosted on EC2, compared to all malicious destinations contacted by all malware samples. We present our findings as a box plot where malware samples are classified according to their submission date to Anubis. The Y-axis characterizes the ratio of dedicated malicious domains that were hosted on EC2, with respect to all malicious domains contacted by malware. Since we included in this experiment only malware samples that used dedicated malicious services on the Cloud, the percentage is always greater than 0%. On the other hand, a malware would fit into the 100% category in case all dedicated malicious domains that were contacted by the malware were strictly found to be hosted on EC2.

As shown in Figure 3.4, miscreants mostly use public cloud services in order to host only certain components of their malware infrastructures. Note that while in 2010, and for malware that uses EC2 to host its dedicated malicious services, only few components of its malware infrastructures were found to be hosted on EC2 (less than 40% of remote malicious domains in average); the use of public clouds to host dedicated malicious services has rapidly evolved in the recent years, including malware samples that were found to be exclusively communicating with dedicated cloud-based malicious domains in years 2013 and 2014. In other terms, miscreants have been recently referring to public cloud services in order to setup and manage their entire malware infrastructure. Therefore, although the use of public cloud services is still limited to only specific compo-

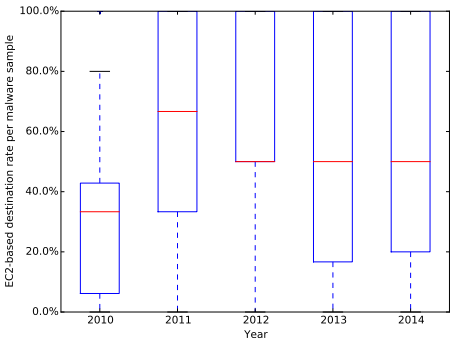


Figure 3.4 – Rate of dedicated malicious EC2-based domains contact per malware sample

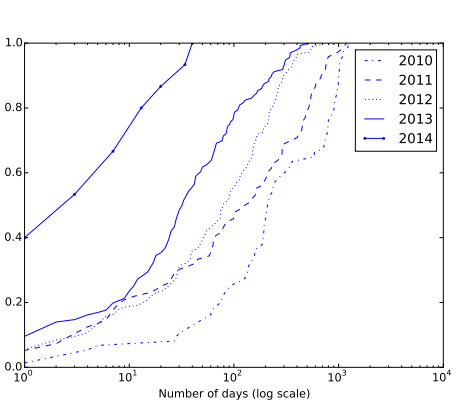
nents of malware infrastructures, we observe an increasing appetite for miscreants towards using public cloud services to setup and manage additional components of their malware infrastructures.

Since most miscreants refer to public cloud services to host only certain components of their malware infrastructure, the second question we would like to answer is whether they specifically refer to public cloud services for this purpose, or whether they use these services as redundant or failover components. We observed for this purpose the history of DNS records for all dedicated malicious EC2-based domains in our dataset until they were blacklisted, then we identified all IP addresses that were associated with these domains and their registrars in the DNSDB service. The results of our investigation were compelling. Out of the initial 1,648 dedicated malicious EC2-based domains that constitute our dataset, 1,620 domains (almost 98.3% of our dataset) were exclusively associated with IP addresses that belong to the EC2 IP range. Note that while 87.5% of dedicated malicious domains were associated with only a single EC2-based IP address, another 10.8% where found to be associated with multiple IP addresses that all belong to the EC2 IP range. In other terms, miscreants were specifically using public cloud infrastructures such as EC2 to host their dedicated malicious services.

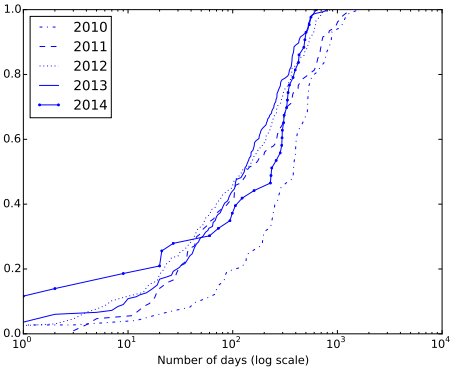
While the use of public cloud services to host dedicated malicious domains is still limited to only certain components of today's malware infrastructures, miscreants appear to be specifically targeting cloud infrastructures for this purpose, and do not use public clouds only as redundant components of their malware infrastructures.

3.2.2 *Dedicated Domains Lifetime Estimation*

In the last part of our study, we tried to estimate the average time that malicious domains persist on EC2 cloud. Our approach leverages the lifetime of the EC2-based malicious domains in order to estimate



(a) Time elapsed until a dedicated malicious EC2-based domain was first observed in Anubis



(b) Time until a dedicated malicious domain was no longer hosted on EC2, after it was observed in Anubis

Figure 3.5 – Lifetime of dedicated malicious EC2-based domains

whether the use of public cloud providers such as EC2 adds more resilience to malware infrastructures.

In the following, we refer to the lifetime of a EC2-based malicious domain as the duration when it was consecutively associated with EC2 cloud IP address in the passive DNS records. Note that the use of passive DNS service only provides an estimation of the real lifetime of these domains but this is an approximation that is often used for this type of measurements [113]. Since domains first appear in the passive DNS services when they are actively requested on the Internet, we consider that the use of this service provides a reliable estimation of the real duration in which a given domain remained active and accessible on the wild. In this section, we observe only dedicated malicious domains that were hosted on EC2, and that were contacted by our malware dataset collected over a period that ends by June 2014. Hence, we only consider historical DNS records associated with malicious servers that are no longer hosted on EC2 cloud at the time of writing. Note that these domains may be still accessible but no longer associated with any EC2 IP address.

We defined two metrics for our experiments. First of all, we measured the time between the domain first appeared in the passive DNS service and the time the malware was analyzed by Anubis. Second, we extract the time when a dedicated malicious domain is no longer associated with an EC2 IP address, after the malware was first submitted to the Anubis service.

The results of our experiment are summarized by the cumulative distributions that are illustrated in Figure 3.5. The graphs separately illustrate the results of our experiments for the last five years since 2010, in order to extrapolate some trends and assess the efficiency of security measures implemented by Amazon over time. The first graph shows that the distribution is clearly moving toward the top-

Id	Domain	IP address	First seen	Last seen	Duration (months)
1	09sp.co.tv	174.129.222.176	August 2010	October 2010	3
		174.129.242.247	January 2011	November 2012	22
2	47gr.co.tv	174.129.222.176	July 2010	July 2010	1
		174.129.242.247	February 2011	November 2012	21
3	dl.ka3ek.com	107.20.206.69	January 2013	November 2013	11
		54.209.129.218	January 2014	January 2014	1
4	hightool.com	107.20.206.69	January 2013	December 2013	12
		54.209.168.250	March 2014	September 2014	7
		54.208.247.222	September 2014	September 2014	1
5	hzmksreiuojy.com	54.241.7.53	April 2013	April 2013	1
		50.18.179.196	April 2013	October 2013	7
		50.17.195.149	July 2014	July 2014	1

Table 3.4 – Examples of domains that rotated their IP addresses on EC2 over time

left corner, meaning that each domain was observed in Anubis soon after it first appeared in the wild. For instance, while in 2011 around 50% of the domains were already present in the passive DNS service (and therefore active in the wild) for 100 days before some malware in Anubis contacted them, in 2014 they had only been active for two days. In other words, the security community became very efficient to promptly detect, collect, and submit malware samples.

Unfortunately, Figure 3.5b shows that the time these domains were hosted on EC2 *after* the malware was analyzed remained stable over the same period. While many factors are involved in this process, this seems to suggest that Cloud providers did not improve their ability to detect and report abusive behaviors. In other words, our observations suggest that the security mechanisms implemented by public cloud service providers have not contributed to reducing the lifetime of malicious domains hosted by these providers.

In order to confirm these findings, and since cloud providers may take-down malicious IPs and not their associated domain names, we analyzed the way malicious EC2 domains resolve to different IP addresses over time. We wanted to evaluate how long malicious machines remain active on EC2 before they are taken down by the cloud provider, and so miscreants may be forced into migrating their malicious domains towards other IP addresses. We monitored for this purpose all DNS records in the DNSDB service, searching for different IP addresses that were associated with every malicious domain in our dataset. Confirming our hypothesis, we found multiple instances of malicious machines that remained active on EC2 even for several months before they were migrated towards other IP addresses in the cloud.

As illustrated by the examples in Table 3.4, the first two malicious domains were associated with the same EC2 IP address for up to twenty consecutive months before they went out from the EC2 IP ranges. Interestingly, certain malicious domains, such as domains 1 and 2, as well as domains 3 and 4 in Table 3.4, were associated with the same IP address during the same period of time, which seems to indicate that miscreants may associate different domain names with their malicious machines in the cloud in order to obtain a better resilience against domain blacklisting. Moreover, they also seem to benefit from the flexibility offered by the cloud in order to migrate towards new IP addresses in EC2 as soon as their current IP addresses have disappeared from the active DNS records, which may suggest that their malicious machines have been identified and taken down by the cloud provider (EC2 in our study). In total, we observed similar behaviors in over 240 malicious domains in our dataset.

3.3 DISCUSSION

When we started our experiments, we were surprised to discover that over 3.5% of the malware samples in our dataset exhibited at least one network connection with a machine hosted on the Amazon cloud. However, as clearly depicted in Figure 3.2, the vast majority of these connections had nothing to do with the fact that criminals were intentionally using the Cloud as part of their infrastructure. In fact, once PPI and other benign services were filtered out, we discovered that less than 1% of the traffic toward Amazon involved a malicious EC2 machine.

Even though this number may seem incredibly small (roughly one every 3200 malicious samples), it is still relevant when scaled to the entire dataset containing tens of millions of malicious samples. Moreover, our experiments show that the use of public cloud services by malicious software has been increasing over the last six years, despite the measures taken by certain cloud providers to limit the extent of these abuses. It also seems that the use of dedicated malicious cloud services is not limited to a small set of malware families, but it can be generalized to most of the malware categories – as summarized by Table 3.3.

The final observation of our study is related to how the cloud providers, Amazon in our case, respond to this threat. Even though we did not have a direct way to observe their reaction, we were able to measure for how long – after the malware was publicly known – the malicious domains it contacted were resolving to IPs hosted on EC2. While the absolute value is not very important, the fact that it remained constant over the past four years seems to indicate that the cloud provider did not make any substantial improvement in detecting and taking down malicious machines.

3.4 CONCLUSION

Public cloud services have rapidly expanded in recent years, yet they have attracted cyber criminals because of the wealth of resources they make available, and the lack of accountability over the usage of these resources. In order to measure the extent at which public cloud services are being abused by cyber-criminals, we conducted a longitudinal study of malware in order to better understand the way it interacts with public cloud services.

In particular, in this chapter we study several characteristics of the traffic observed between malicious samples and the Amazon EC2 cloud. Based on our measurements, we discuss the evolution of this phenomenon over the past six years, and we present few key observations and insights into this growing problem.

We hope that our study can shed some light on a key component of the modern cyber crime infrastructure, and would provide useful input to devise appropriate mitigation strategies.

4

LIVE MONITORING OF PHISHING ATTACKS

In this Chapter we investigate how a service provider can take advantage of its position to better investigate online attacks. In particular, we chose to focus on phishing kits, as all existing studies on the topic were performed by third-party researchers and therefore were unable to capture the entire process and life cycle of the attack. To solve this problem, in this chapter we present a novel approach to sandbox live phishing kits that completely protects the privacy of victims. By using this technique, we perform a comprehensive real-world assessment of phishing attacks, their mechanisms, and the behavior of the criminals, their victims, and the security community involved in the process.

Despite the large effort and the numerous solutions proposed by the security community, phishing attacks remain today one of the main threats on the Internet [95]. They usually aim at deceiving users into visiting fake web pages that mimic the graphic appearance of real and authentic websites [84]. The main goal of an attacker, also known as phisher, is to collect sensitive user data such as login credentials, banking information, or credit cards numbers. The stolen data can then be monetized by leveraging hijacked accounts and performing fraudulent online transactions, or indirectly through the resale of the stolen information to other cyber-criminals, mostly on the Internet black market [78].

Phishing attacks constitute a major challenge for Internet Service Providers (ISPs), as well as for email providers, browser vendors, registrars, cloud service providers, and law enforcement agencies. To mitigate these attacks, a broad set of solutions have been proposed, tackling each of the three different stages that constitute a phishing attack [78]. At the first stage, they try to prevent phishing emails from reaching the end users by applying email filters [61], or by detecting [122, 144, 195, 197, 203], blocking, or taking down phishing web sites [125]. At a second stage, existing solutions focus on providing better user interfaces, such as browser plugins, that inform users about the reputation of a target web site, and notify users as soon as they are redirected towards a potentially malicious page [59, 196]. Finally, the last line of defense relies on proper education to help users recognize phishing sites [103, 104]. Despite this considerable effort, the phishing problem is far from being solved and a recent report by the Anti-Phishing Working Group (APWG) shows that the number of unique phishing sites was still increasing in 2015 [2], and that the

number of phishing reports the APWG receives has almost doubled between 2014 and 2015.

In order to discern phishing attacks, diverse efforts have been made by security researchers, and that involve different actors in the phishing ecosystem. However, previous studies focused mainly on the technical aspects of the problem, i.e., how phishers compromise vulnerable servers [190], and how phishing kits work [49]. Researchers also remotely analyzed existing phishing pages, based on ground truth datasets such as spamtraps [129] and phishing blacklists [125], in order to provide a real world assessment of the number of victims and to measure the efficiency and extent of take-down operations. Recently, Bursztein et al. [32] advanced our understanding about phishing by studying criminals incentives and the way they monetize stolen user credentials.

Unfortunately, previous studies have been confronted to two main dilemmas. First, most phishing kits were monitored only *after* they had been detected by public or private anti-phishing services. This drastically limits the extent of these studies, since an important part of the phishing life cycle (preceding any detection) has mostly remained unknown. Second, researchers have never observed the way real victims interact with phishing kits because of obvious ethical reasons. In this dissertation, we try to fill the gaps in the understanding of the phishing ecosystem by analyzing the attackers behavior and the way the potential victims interact with phishing kits in the wild.

As discussed in [128], *"a natural tension exists between conducting accurate, reproducible research and reducing the harm caused by the content that is being removed"*. Two are the main challenges that affect research on phishing attacks: "Should researchers notify affected parties in order to expedite take-down of phishing sites? Should researchers intervene to assist victims" [128]?

Bearing these issues in mind, this chapter proposes a new approach that lifts the barriers imposed by these ethical considerations in order to provide a first comprehensive real world assessment of phishing attacks, their mechanisms, and the behavior of all the actors involved in the process. Our approach leverages the valuable information to which a service provider have access, including the traffic log of the infrastructure. However, due to the privacy issue of a service provider while dealing with real customer data, we adopted a similar but realistic scenario to collect data where a web honeypot was used to attract real attackers into installing phishing kits in a compromised web application. This is inspired by the fact that, according to the APWG's Global Phishing Survey, 71.4% of the domains that hosted phishing pages were compromised domains [1]. We then present a novel sandbox technology designed to neutralize a phishing kit while maintaining it functional for a long period of time. Our approach is designed to strictly preserve the victim privacy, without interfering

with the attack process in order to make sure that attackers can compromise the honeypot, install phishing kits, and conduct functional tests without being alerted about the sandbox configuration.

Preserving the user privacy is a very challenging task. Most phishing kits instantly exfiltrate the newly collected victim credentials to the attacker, leaving no time to remove these credentials from our servers. Even worse, almost 97% of malicious phishing pages are accessible via unprotected HTTP connections [1], which may expose the cleartext compromised credentials to eavesdropping over the network. Finally, as discussed in section 4.5, some phishing pages reroute the HTTP requests to a different server directly controlled by the attacker [190], using the compromised application only to host the page but not to collect the data.

Our sandbox proposes a comprehensive solution to these problems, allowing us to collect real world data about the behavior of both attackers and victims, and to perform the first thorough investigation regarding phishing attacks. To the best of our knowledge, no previous work was able to monitor, in a white-box fashion, the lifecycle of a phishing kit.

Based on these elements, we discuss a number of interesting findings. For example, our experiments show that phishing kits are only active for less than 10 days since their installation and over this time most of them collect a limited number of user credentials (fewer than reported in past experiments). Therefore, attackers rely on compromised websites to install a large number of phishing kits in a sort of shot-and-forget approach, rapidly moving to new phishing pages as the old ones get blacklisted. We also confirm that Google Safe Browsing (GSB) and Phishtank are very effective tools to protect end users. However, our experiments show that both services tend to blacklist phishing URLs between 10 and 20 days after their first appearance, and this is often too late as most of the victims already connected to the page. Finally, we observed a considerable *flash crowd* effect once phishing URLs appear in public blacklists. If not properly modeled, this phenomenon can completely skew the analysis results, by confusing security researchers with potential victims.

To summarize, we make the following contributions:

— We present a novel approach to sandbox live phishing kits that completely protects the privacy of end users.

— We observed the interaction of attackers, victims, and security researchers with the phishing pages, reconstructing for the first time the entire lifecycle of a phishing kit.

— For the first time, we measure the impact of blacklisting services on phishing pages from the time in which they are first installed (and not from the time they are reported or discovered by security companies). We also discuss new techniques to use

Figure 4.1 – Typical Phishing Attack

the collected data to promptly identify the email address used by criminals to retrieve the stolen information.

Beyond these main findings, we also discovered two new phishing techniques that have never been reported before, and we conducted a thorough analysis of the modus-operandi of the corresponding campaigns.

4.1 BACKGROUND

In this section we provide a more detailed description of phishing attacks and their main actors. We then describe in more details the ethical issues we encountered during each phase of our experiments.

Anatomy of Phishing Attacks: A typical phishing attack, as depicted in Figure 4.1, consists of three main actors: a *phisher*, a set of potential *victims*, and possibly a number of *third party visitors* – such as researchers and security editors. Phishers mostly seek to compromise vulnerable web applications, install phishing kits that mimic victim web sites, and advertise the phishing URLs using, for example, spam emails and posts on social networks. The victims are the end users who receive these messages and connect to the phishing pages. Phishers usually seek limited interactions with their victims as their main goal is to hijack sensitive data without disclosing the real nature of their phishing pages. Therefore, they often redirect victims to the authentic website after they have provided their credentials, or they redirect them towards error pages to make them disconnect from the phishing site. As soon as a victim connects to a phishing page, she can either realize the real malicious nature of the site and disconnect, or she can be fooled by the legitimate-looking page appearance and attempt to login, thus providing her credentials. Finally, the third party category includes visitors from security companies, public crawlers, and independent researchers. They usually examine and monitor the phishing pages only after the presence of the phishing kit is included in popular blacklists (e.g., PhishTank), as discussed in details in section 4.5.2. The behavior of third party visitors can be very similar to the behavior of real victims, which makes it difficult to separate these two actors within the same experimental setup.

Ethical Considerations: Researchers have already proposed the use of honeypot systems as a tool to analyze phishing attacks [190]. How-

ever, conduct live phishing experiments and evaluations inside honeypots has always been confronted with ethical considerations that have severely limited the scope of all previous research studies about phishing attacks. The result is that previous experiments were often limited to the analysis of how phishers compromise the honeypot, as well as the static analysis of the collected phishing kits. Unfortunately, honeypots have never been used to study the behavior of the victims as they connect and interact with the installed phishing pages. As soon as victims are being enrolled into a honeypot experimental setup, ethical considerations cover all aspects related to the protection and perfect secrecy of user identities, as well as the secrecy of their credentials in case where they may be exposed during the attack. In order to properly address this important ethical problem, one should have a better understanding about the key steps of a phishing attack where the user identity or credentials may be exposed. We divide these ethical issues into two main categories, depending on whether they may be addressed on the server or on the client-side.

On the server side, the phishing kits aim at collecting user submitted data. The stolen data may be either locally stored on the honeypot server until it is retrieved by the phisher, or it may be instantly sent to the phisher by email or direct HTTP connections. Therefore, it is important to prevent any user data to be locally stored on the server, or even processed by the phishing application. Moore et al. used public accessible data collected on vulnerable web servers, including statistics of the web page hits and credentials that the attackers had collected and forgotten to remove from the vulnerable server [125, 126]. Following the publication of their study, more ethical issues concerning the user credentials were exposed, most of which being still unanswered [128].

Apart from the above ethical problems on how to prevent information from being stored on the server, ethical issues also exist on the client-side. In fact, phishing kits may collect user credentials by posting them to other malicious servers that are under the direct control of the attacker. Moreover, 97% of the phishing pages are accessible via clear-text HTTP connections [1], thus exposing the victim's sensitive data to eavesdropping over the network. Therefore, and even though no sensitive data would be hosted on the remote server, there may still be a considerable risk because of the clear-text sharing of user credentials on the network.

Finally, it is important to note that our institution does not have an IRB, but we asked advice from the company legal department, and were granted permission to perform the research.

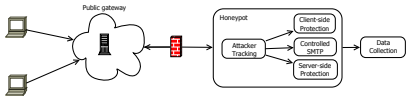


Figure 4.2 – High Level System Overview

4.2 DATA COLLECTION

We leverage an existing honeypot infrastructure [35] as a basis to implement our system. As shown in Figure 4.2, our architecture consists of two main components: a public proxy gateway and a private backend server that implements the main honeypot applications. The gateway was hosted on a number of public hosting providers, including Amazon EC2, which previous research has revealed to be a popular target for attackers looking for machines to compromise and conduct malicious activities on the Internet [72]. The proxy server does not host any content and acts purely as a reverse proxy to forward all HTTP connections through a secured VPN channel towards the honeypot server hosted in our premises. For security reasons, outgoing traffic from the honeypot is dropped at the firewall – except for DNS queries and for a limited number of verified SMTPS connections (as explained in Section 4.3.3). On the honeypot server, we configured 18 vulnerable PHP pages that can be exploited by attackers and allow them to upload files and execute shell commands. Note that the honeypot is configured in such a way that attackers cannot modify these PHP pages.

The data collection module periodically retrieves the data collected on the honeypot server, including the server access logs and the uploaded files (such as web shells, phishing kits, defacement pages, exploit kits, and hacking tools).

Elimination of Other Malicious Files: The identification of phishing kits is performed through a number of heuristics, complemented by a manual classification. For instance, most phishing kits contain a large number of files and resources required to replicate the targeted website [49], and therefore isolated files are unlikely to be used for phishing. We also adopted a number of keywords to determine the content of the files and identify possible targets. However, it is very difficult (and outside the scope of this chapter) to setup an accurate filter that can precisely separate phishing applications from other malicious files uploaded to the server (we refer the reader to existing work on this specific problem [203]). In our study we decided to adopt a conservative approach and we manually analyzed all the files that did not match our filters in order to be sure that *only* phishing pages were hosted on our honeypots. We removed on a daily basis other malicious files including web shells, exploit kits, and drive-by download from the honeypot. As part of this activity,

we may have erroneously removed some small phishing kits, but we believe that it is better to be conservative and avoid exposing users to other dangerous threats.

Data Exfiltration by Client-Side Side Channels: We also verified that the phishing kits did not leverage other side-channels to capture and successfully exfiltrate the credentials from the victim browser. We found three PKs that had these functionality and used obfuscated JavaScript code disguised as a HTML `img` tag to send the user credentials directly to a remote server. However, our client-side protection described in Section 4.3 acts directly on the typed password, before the malicious JavaScript retrieves it.

Overall, through an accurate user inspection, we were able to confirm that no user credentials were disclosed during our experiments.

4.3 SANDBOX AND PK NEUTRALIZATION

This section presents an overview of our platform and describes the properties and design configuration that enable us to address the ethical issues outlined in section 4.1. It also illustrates the deployment scenario, including the system components and data management procedures. Finally, it describes the low-level implementation details and discusses the potential limitations of our approach.

4.3.1 *Design Goals*

In order to address the ethical issues and to enable a sanitized honeypot platform that preserves the privacy and security of any potential victim data, our system achieves the following design requirements:

Client-Side Data Mangling: To prevent sensitive data from being sent out of the user terminal, our system injects into all phishing pages a JavaScript component whose purpose is to replace any posted information with random data before it is sent over the network. The injected code protects all potential victims as long as they have not explicitly deactivated JavaScript in their browsers. For those users who may have deactivated JavaScript, our system also injects a HTML noscript tag that redirects user to an error page so that they would disconnect from the honeypot.

Server-Side Data Randomization: As an emergency backup in case JavaScript is enabled on the victim terminal but the injected code fails to replace the posted information, our honeypot server relies on a custom Apache PHP module that filters all incoming data before it reaches the phishing kit. In particular, our solution replaces on the fly all user data that reaches the server with random fake data, making

sure that no sensitive information may ever reach the phishing kit, nor it can be locally stored on the honeypot.

HTTP Redirection Disruption: To make sure that our honeypot may not be used by the attacker as an elementary component of a broader malicious redirection chain [113], our system uses static analysis techniques to detect and disable any form of web redirection, and so to make sure that no users may be redirected from the honeypot towards any other malicious website under the control of the attacker.

Concealed Instrumentation: One of the main goals of our system is to protect the users while being perfectly transparent for the attacker. Therefore, the honeypot is designed to identify attackers and monitor their procedures and mechanisms without revealing the real nature of our experiment. In particular, we noticed that attackers usually test their kits by inserting fake credentials in the phishing page and verifying that such credentials are correctly exfiltrated to their preferred drop-zone. In order to ensure that these testing operations are successful, our system needs to selectively disable the client- and server-side randomizations, and to allow phishing kits to send emails (which is the most popular exfiltration technique implemented by the phishing kits [49]) when attackers decide to test their pages.

4.3.2 *System Overview*

As illustrated in figure 4.2, our honeypot implements five main functionalities, including *the attacker tracking module, the client-side protection module, the server-side protection module, the controlled SMTP module, and the data collection and processing module*.

As a core component of the system, the attacker tracking module is responsible for distinguishing attackers who connect to the honeypot from other benign users such as victims and third party visitors. This allows us to apply different access control policies whose purpose is on one hand to enable attackers to verify and test their phishing kits, and on the other hand to prevent any sensitive data posted by the victims from being captured by the attacker. This module assigns the role of the attacker to the user who installed the phishing kit on the compromised application. The attacker tracking module further keeps track of all attackers who connect to the honeypot using a history of attacker IP addresses and their associated user agents, and provides this information as input to the subsequent protection and data collection modules.

The client-side and server-side protection modules use the list of attackers provided by the tracking module as input in order to identify potential victims and to prevent their data from being exported or hijacked from the user terminal. If the visitor is not an attacker, the module inspects each outgoing HTTP response and injects in each

page a HTML noscript tag and a static JavaScript file (more details in Section 4.3.3), which hooks the data submission and replaces the user's data with fake information.

The server-side protection module performs two functions: *backend data randomization* and *malicious redirection disruption*. In the first case, it acts as a second line of defense in addition to the client-side protection. It operates on the server side, and it replaces the incoming data with random values before it is passed to the web application. The server-side protection module also intercepts and blocks all HTTP, HTML, and JavaScript redirections that point to other remote websites in order to make sure that the honeypot server cannot be used as a stepping stone within a broader malicious redirection chain. It detects such redirections through performing static rule-based analysis over the content of each HTTP response provided by the server. Our system allows only redirections to other pages hosted in our honeypot, and automatically intercepts and drops any other destination.

Finally, as most phishing kits exfiltrate the stolen data by emails, and since our honeypot is configured to drop outgoing SMTP connections to prevent attackers to send spam, we also implemented an SMTP module that allows each attacker identified by the tracking module to send a configurable amount of messages (two in our experiments) to test a freshly-installed phishing kit.

4.3.3 *Implementation*

The experimental setup on the honeypot leverages two Apache HTTP modules, namely `mod_php` and `mod_security`. We configure and extend these two modules in order to incorporate our system functionalities. In the following, we describe in details our system implementation.

mod_php provides PHP support to the Apache HTTP server, and was extended to implement the attacker identification and server-side protection modules. In particular, we hook the PHP function `rfc1867_post_handler`, which is actually the form-based file upload handler, and we save the IP address of the attacker who has successfully uploaded a file in a privileged location. To implement our server-side protection module, we modify the functions `php_std_post_handler` and `php_default_treat_data`, which provide respectively the handler for HTTP POST and GET requests. These handlers whitelist the 18 pre-installed vulnerable pages to ensure that attackers can reach and use the honeypot. For requests toward other pages, the handlers perform the attacker verification and further replace the data with fake ones when the request is not originating from an attacker. The handlers identify specific types of information (such as login, email, and credit card fields) by using a pre-defined set of keywords and regular expressions, and overwrite these fields with randomly gener-

ated data that reproduces the same format. In other words, emails are replaced with seemingly valid (but non-existent) email addresses, credit-card numbers by other fake numbers, and so on. If the system is unable to recognize the type of a field, its value is replaced with a random string of alphanumeric characters.

mod_security is a web application firewall that provides attack detection, traffic monitoring, logging and real-time analysis [1]. We extend the core functionalities of this module in order to implement the client-side data mangling, the malicious redirection disruption, and the data collection modules. The client-side data mangling behaves almost the same way as for the server-side data modification. The only difference is that it injects a HTML noscript tag and a static JavaScript code in the HTTP response in case the destination has not been identified by the attacker tracking module. The static JavaScript code, executed on the remote victim browser, modifies any data provided by the user before it is sent over the network. To achieve this, the injected JavaScript first replaces the native form submission function *submit()* with a custom handler that dispatches automatically a *submit* event on the page. Then it adds a *submit* event listener that hooks the form submission and modifies the submitted data. Note that the static JavaScript is prepended to the HTML page so that it is always executed before other JavaScript. During our experiment, we did not observe any PK that tried to detect this kind of hooking. If JavaScript is deactivated on the victim browser, the HTML noscript tag redirects the victim to an error page to prevent him from disclosing his credentials. Moreover, in order to prevent potential victims from being redirected towards other remote malicious destinations, we configure `mod_security` to drop all HTTP responses that contain redirections. Note that malicious redirections are only disabled for potential victims that cannot be identified by the attacker tracking module. The HTTP redirection can be detected by combining the response status code and the *Location* field in the HTTP header. To identify HTML and JavaScript redirections, our module parses the content of the response body and analyzes its HTML and JavaScript code. The last configuration option that we have added to the `mod_security` module is the ability to collect files uploaded to the honeypot. We configure the `SecUploadDir` and `SecUploadKeepFiles` functions in order to save a copy of the uploaded files into a protected location that is invisible to the attackers.

Controlled SMTP: The honeypot is configured to send all emails with a free email service provider through sSMTP [2]. To prevent attackers from sending spam emails from our honeypot, we limit to 2 the number of emails that each attacker can send. The original binary of

1. https://www.modsecurity.org/
2. http://linux.die.net/man/8/ssmtp

sSMTP is replaced with a modified version that checks the presence of a specific flag in the arguments passed by the caller. Only the PHP mail handler is configured to call the sSMTP with the desired specific flag. Moreover, we modify the PHP mail handler so that it keeps only the first recipient when the destination contains multiple recipients.

Experiment Limitations: To mitigate denial of service attacks or botnet scans, we rate-limited to 5 the number of concurrent connections from the same IP address. We opted for this configuration because we detected multiple scan attempts to our honeypot using a common proxy server, and that most of these attempts did not lead to any attack. Moreover, we believe that it is highly unlikely to have more than five victims connecting simultaneously from the same IP address and it is equally unlikely to have more than five simultaneous attackers interacting with our honeypot using the same proxy IP address. Therefore, we believe that this rate-limiting has a marginal impact on our experiment.

Our strict attacker identification process would inevitably prevent attackers from submitting fake credentials to the site in case they use a combination of IP address and user agent that is different to the one used to upload the PK. However, we believe that this solution offers the best trade-off to achieve a complete protection of the user privacy. Nonetheless, based on the number and diversity of the collected kits and the potential victims, we believe that our experiment is sufficiently representative of existing phishing attacks in the wild, as discussed in section 4.4.4.

4.4 PHISHING ATTACK GLOBAL PICTURE

We collected 643 unique phishing kits, which were uploaded on our honeypot over a period of five months from September 2015 to the end of January 2016. Out of this initial dataset, 474 kits (74% of our initial dataset) have been correctly installed by 471 distinct attackers. The remaining kits were likely automatically uploaded by exploitation bots but never unpacked nor configured by the attackers. The installed phishing kits targeted 36 distinct organizations, mostly online banks, but also social networks and e-commerce portals. The five most frequent targets were Paypal (375), Apple (26), Google (10), Facebook (9) and the French online tax payment system (6).

This section presents an overview of our main findings, including the way attackers setup and operate their phishing attacks, the behavior of victims as they interact with the phishing kits, and our assessment for the lifetime of live phishing kits on the Internet. We initially focus on aggregated statistics and on understanding the big picture, and then we discuss the technical aspects related to how we identify victims, and how we separate them from attackers and third party visitors.

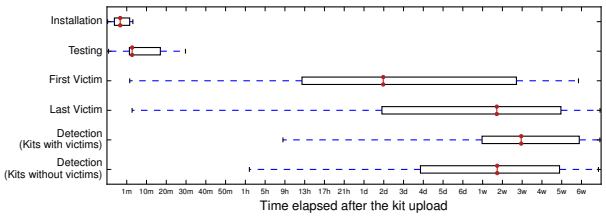


Figure 4.3 – Phishing Attack Timeline

Figure 4.3 illustrates an aggregated timeline of all phishing attacks we observed in our study. On the graph, the Y-axis shows the major phases of an attack, and the X-axis indicates the time elapsed since the phishing kit was first uploaded to the honeypot. During our experiment, and as shown in Figure 4.3, we split the lifecycle of an attack in five separate phases: installation, testing, interaction with the first and last victims, and detection by popular blacklists.

The *installation* phase includes the actions performed by the phishers to unpack, install, and set up the phishing kit on the compromised machine. The *testing* phase describes how phishers test and verify the correct behavior of their newly installed kits. The *first* and *last victims* capture the time elapsed until we observe respectively the first and last connections from a victim. Lastly, the *detection* phase covers the time at which the phishing URL is added to public phishing blacklists (Google Safe Browsing and/or PhishTank in our experiment). All results are illustrated using a box plot that captures the distribution of the data spanning from the first quartile to the third quartile, with the red line showing the median value.

4.4.1 *Attackers Behavior*

Based on the referer field, 29% of attackers located our honeypot using search engine queries, mostly on Google (28%), and Yahoo (0.8%). Interestingly, over 40% of the attacks came with a Facebook-related referer, which seems to indicate that attackers are increasingly using social networks to share data (e.g., the location of web shells) or simply forge the referer field to make their traffic look more legitimate.

While most of the attackers installed only one copy of the phishing kit, some (26%) have installed their phishing kits on multiple subdirectories. After the installation was completed, 70% of the attackers visited the phishing pages and 58% of them submitted fake credentials to verify whether the kit was operating correctly. This confirms our initial hypothesis that a phishing sandbox needs to protect the victims but let the attacker freely play and test the installed pages, as described in Section 4.3. Without this feature, almost 60% of the uploaded kits would have been abandoned by the attackers.

As illustrated in Figure 4.3, phishers installed the uploaded kits very quickly on the compromised web server, just few minutes after the kit was uploaded. The testing phase was also performed immediately after the installation. However, the few minutes of delay and the irregular patterns suggest that this was performed manually and not through automated bots. Quite interestingly, after the testing phase was completed, we never observed any connection to the phishing pages from the same IP address.

In the second stage of a phishing attack, attackers would make use of standard social engineering techniques in order to drive potential victims into connecting to their phishing pages. During the entire period of our study, we observed only one attacker who has tried to use the compromised machine to also send the phishing emails (all messages were blocked by our firewall). This seems to suggest that attackers have decoupled the process of compromising public servers and installing phishing kits, from the process of sending phishing messages. A possible explanation is that using different infrastructures is more robust and decreases the probability that the compromised server is detected by security solutions.

4.4.2 *Victims Behavior*

To study the behavior of victims as they connect to the phishing pages, we first need to separate them in our dataset from other actors such as public crawlers and third party visitors. We leverage multiple heuristics and empirical observations that we describe as follows. First of all, users who were assigned the role *attacker* by the attacker tracking module do not belong to the victims category. We also discard public crawlers by looking at the user-agent header field in incoming HTTP requests – since it is reasonable to assume that a victim would not spoof its browser user-agent to mimic a search engine.

The most challenging part of our study was to separate victims from other third party visitors, such as researchers and security editors who may find the phishing URLs on public blacklists and connect to the honeypot to verify the content of the phishing pages. First of all, the source IP address may reveal valuable information that enabled us to classify users as either potential victims or third party visitors. For example, after verification in the whois database, we found that a large number of connections to our honeypot originated from university researchers (e.g. IP ranges belonging to Boston University, University of Pennsylvania, Carnegie Mellon, and Massachusetts Institute of Technology) and security companies such as Kaspersky, Symantec, Bluecoat, and Fortinet. To be conservative, we consider all these users as third party visitors (even though this can misclassify students who fell victim of the phishing attacks) and we remove them

from the victims' category. This approach does not allow us to identify other third party visitors, such as independent researchers and curious individuals who found the URL on public blacklists or hacking forums. To identify this specific category of users, we leverage a number of additional heuristics, such as the fact that researchers may connect at regular intervals to verify whether the phishing pages are still available, or they spend a considerable amount of time investigating different sub-pages or other resources used by the phishing kit. Based on these observations, we filter out a large number of users from the victims category. We believe that our cleaning approach was very conservative and had potentially over-estimated the number of third-party researchers and reduced the number of victims. However, as we explain in more details later, in our experiments we noticed that a large amount of incoming HTTP requests were not from real victims and therefore would inflate the results if included in our analysis.

After our aggressive filtering, we counted a total number of 2,468 victims who have connected to 127 distinct phishing kits. Although the total number of victims seems to be relatively small compared to previous work that measured the impact of phishing attacks, our study has the merit of providing the first fine-grained assessment of the number of victims for a phishing attack. In fact we addressed two main limitations that have contributed in the past to overestimate the number of victims for a phishing attack.

First of all, we observed a spike in the number of (third party) visitors just after a phishing page first appears on public phishing blacklists (details in section 4.5.2). While such crowd phenomenon is a natural consequence of blacklisting a phishing page, previous studies did not separate such connections from the set of real victims, which could have contributed to largely overestimating the real number of victims. Using our sandbox configuration we were able to identify and clearly separate this phenomenon, whose impact is further explained in details in Section 6.

Second, our honeypot configuration also enabled us for the first time to observe the victims submitting their credentials to the phishing page. This is made possible as we analyze the behavior of victims on a per-user basis, and so we can verify whether each victim has performed any HTTP POST request, which suggests that the user has submitted data to the phishing page. Over the period of our study, we found that 215 users (9% of the total) have indeed posted their credentials to the phishing page. Note that since our system replaces automatically the submitted data with fake values, we are however unable to estimate the number of victims who send fake credentials to the phishing page.

Finally, the geolocation of the victims did not provide any particular insight, except for confirming that PKs are often targeted to a

particular audience (e.g., French citizens for the French tax payment system), and in fact many PKs received the majority of their victims from a single country.

4.4.3 *PK Lifetime*

We measure the effective lifetime of a phishing kit as an indicator of the time interval during which a phishing kit remains operational on the Internet. We consider a phishing kit to be operational as long as new victims connect to the phishing page. Note that even though the phishing kits are kept online on the honeypot server throughout the duration of our study, we did not observe new victims after the phishing URLs were blacklisted by a large enough number of anti-phishing services and browser plugins.

A main challenge when measuring the effective lifetime of a phishing kit is that it would be very difficult to capture the exact time for the last victim who connect to the phishing page. In particular, we may still observe few users connecting to a phishing kit days after the phishing kit has been abandoned by the attacker, thus causing us to overestimate the lifetime of a phishing kit. To address this challenge, and so to eliminate such outlier observations, we cut the tails of the distribution and only consider the time interval during which we observed 90% of victims connections. As shown in Figure 3, the *first victim* captures the time elapsed until we observed 5% of victims connecting to a given phishing page, and the *last victim* captures the time elapsed until we have observed 95% of victims. Using this definition, we observed during our study that the first victims connect to a phishing page in average two days after the page was installed by the attacker. The last victims (at the 95% threshold) connect in average to the phishing page after 10 days. This gives us an estimated lifetime of eight days.

4.4.4 *Effectiveness of Phishing Blacklist*

To evaluate the reaction of the security community against phishing attacks, we leverage two anti-phishing services: GSB and PhishTank. We chose these two actors first because GSB is integrated by default into Chrome and Firefox, which together account for 87.1% of the market of Internet browsers [3], and second because PhishTank has been extensively used as a source feed in many previous research studies [119, 125, 202]. Our main goal in this section is to measure the time it takes for an anti-phishing service to blacklist phishing URLs since the corresponding kits were first uploaded to the honeypot. To do so, and throughout the duration of our experiment, we periodi-

3. http://www.w3schools.com/browsers/browsers_stats.asp

cally check, multiple times per day, both blacklists for all phishing URLs that were installed in our honeypot.

While almost all phishing pages that were installed on the honeypot (98% of them) were correctly detected and blacklisted by the two phishing blacklists, we observed an average detection latency of 12 days. More precisely, we split this value in two separate categories: phishing kits for which we observed victims were blacklisted in average 20 days after installation, while kits with no victims were blacklisted in average 10 days after their installation. As illustrated in Figure 4.3, the detection latency varies quite a lot depending on the phishing pages and the evasion techniques used by the attackers, as further discussed in Section 4.5.2. More importantly, 62% of the phishing kits were blacklisted only after 75% of victims had already connected to the corresponding phishing page. While these observations seem to indicate that the anti-phishing services were not effective against a certain category of phishing attacks, they were indeed very proactive against another 27% of the phishing kits during our study, as they blacklisted these URLs even before 25% of victims have yet connected to the phishing page.

Another interesting aspect is the fact that GSB initially blacklisted only individual phishing kits. However, after many phishing pages had been reported for the same domain name, GSB changed its policy and started blacklisting entire directories subtrees in those domains. This approach may have proactively blacklisted other kits that were installed within these same directories.

4.4.5 *Measurement Bias*

Clayton et al. [41] draw the attention to the potential measurement bias in this type of studies. For instance, PhishTank only contains 40% of all phishing URLs, but 100% of all PayPal phish. Therefore, if the composition of the PhishTank dataset is not understood, the focus on attacking PayPal will be overestimated. In our study, we evaluate the effectiveness of the PhishTank dataset against the 471 phishing kits installed in our honeypot. Even though an important amount (375, i.e., about 80%) of these PKs involved PayPal, only a small portion (about 1%) of them have been reported and thus included in the PhishTank dataset. This in return corroborates to some extent the diversity and the representativeness of our dataset. We thus argue that our measurement is relatively representative.

Another concern may be related to the large number of PKs that received no victims. It is hard to know the real reason, but we do not believe that the fact that attackers could have recognized the honeypot is the main explanation behind this phenomenon (even though in some cases it could certainly be the case).

Drop Techniques	Live Kits
Email	443
File	30
POST	2
MAILTO	1

Table 4.1 – Drop mechanisms of the live phishing kits

In 34% of the PKs that did not receive any victims, the system did not receive any follow-up connection after the PK was uploaded. These PKs are mostly uploaded by automated exploitation bots without any human activity. In this case, it is possible that the attackers had many available targets that were compromised by their bot, and simply did not choose to use our system. In the remaining 66% of the kits with no victims, the attacker connected to the PK and explicitly tested it by submitting some credentials. Our logs did not report anything anomalous and the PK correctly sent the email with the testing credentials to the attacker, but then received no victims. Even though we do not know the reason, it is possible that no victims clicked on the phishing link, or that the phishing emails were largely stopped by antispam solutions. Overall, since we observed that PKs are often fire-and-forget pages uploaded in large numbers and used only for few days, it can be normal that some of them are never used and some have zero-success rate.

4.5 CASE STUDIES

In this section we provide details about four interesting cases we observed in our experiments including a new phishing kit dropping technique, a blacklist evasion technique, the time distribution of victims as they connected to the phishing pages, and we discuss a possible way to use our system to detect drop email addresses from live phishing kits.

4.5.1 *Dropping Techniques*

Most of the phishing kits contain the complete phishing web sites in a ready-to-deploy package. One of the main function of these kits is to automatically send the collected information to the attackers. In order to assess the technical evolution of the phishing kits since a previous study [49] conducted in 2008, we analyze the mechanisms that the phishing kits use to exfiltrate information.

As illustrated in Table 4.1, the vast majority of kits use email accounts to send data to the attackers. Few kits save directly the collected information on the server, and only two send it to a remote

server using a HTTP POST request. Interestingly, we observed during our study that attackers experimented a new drop technique that aimed at sending the information in a HTML form through an email directly from the end user terminal. In order for this to work, attackers configured the HTML form action to `mailto` the values directly to their email address. Even though this technique is already well documented [4], this is the first time that phishers were observed to implement it inside a phishing kit. However, note that this method is very unreliable and only works with a particular combination of Internet Explorer and Outlook Express.

To further discover new techniques adopted by phishers, we analyze all the external links included in the kits uploaded to our honeypot. We found that 10 kits made use of resources directly fetched from the content distribution network (CDN) of the target organization. More interestingly, we also found that another 98 kits contained code to contact other computers likely under control of the phishers. These were either carefully disguised backdoors to exfiltrate the stolen credentials on a second channel (which was blocked by our firewall), or ways to retrieve information provided by the attackers, typically blacklists of endpoints or user-agents to block from viewing the page.

4.5.2 *Blacklist Evasion*

During our experiments we experienced an unexpected service outage caused by disk space exhaustion. Further investigation revealed that the exhaustion was caused by a phishing kit whose entry page, namely `index.php`, automatically creates a random subdirectory, copies the content of the entire kit inside it, and then redirects the visitor to the newly generated random location. The following PHP code presents an example of such behavior:

```
$random=rand(0,100000000000);
$md5=md5("$random");
$base=base64_encode($md5);
$dst=md5("$base");
$src="New Folder";
recurse_copy( $src, $dst );
header("location:$dst");
```

After the PK was installed, the phisher visited the entry page, which generated a copy of the kit at a random location. Interestingly, the phisher could distribute either the newly generated link to conceal the original phishing page location, or the link to the entry page so that each visitor would be automatically redirected to a different location. To understand the intended function of this phishing kit, we leverage the server access log to observe the first victims of this

4. http://www.html-form-guide.com/email-form/email-form-mailto.html

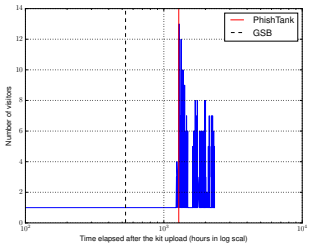


Figure 4.4 – Visitor time distribution of the kit with blacklist evasion technique

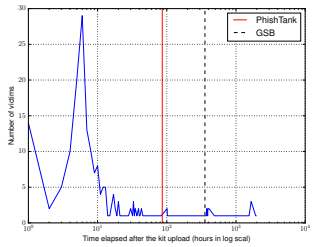
kit. We found that the phisher conducted a phishing campaign with a link to the entry page of the phishing kit, as the first victims landed directly to the entry page. This approach may seem controversial at a first glance, as the phishers exposed the real link to the phishing kit instead of hiding it from being blacklisted. In order to understand the reason behind this, we examine how PhishTank and Google Safe Browsing react to such phishing kit.

PhishTank publishes phishing reports submitted by different users, which allows us to retrieve the reported links and further compare them with the original link to the entry page. Unfortunately, most of the users had been fooled by this technique, and reported the randomly generated locations instead of the entry page, as illustrated by the following anonymized web server access log:

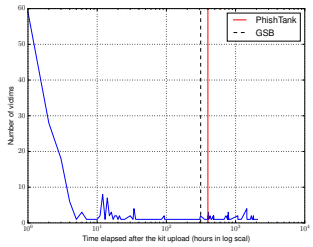
```
[12/Nov/2015:18:57:41] 14.xx.xxx.198
GET /kit/ 302
User-Agent: curl/7.25.0
[12/Nov/2015:19:01:35] 213.xx.xxx.100
GET /kit/8c5fcf4518e94a9f272d60ee75c309a7 301
User-Agent: Mozilla/4.0
[12/Nov/2015:19:20:45] 213.xx.xxx.100
GET /kit/8c5fcf4518e94a9f272d60ee75c309a7/redirection.php 200
User-Agent: Mozilla/4.0
```

The user first leveraged a command line tool (curl) to visit the link distributed by the phisher, which referenced the entry page of the phishing kit. As a consequence, a new phishing link has been generated, which however has not yet been visited until the reporter verified it a few minutes later with an IP address different to his first connection. The reporter was then redirected to the phishing page under a random location, which was also the link reported to PhishTank.

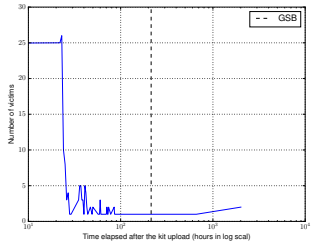
This PK confirms the existence of the crowd effect described in the previous section. In fact, while real victims would visit different randomly generated URLs, we observed hundreds of incoming connections toward the same link blacklisted by PhishTank. In Figure 4.4, the X-axis presents the number of hours in log scale that have elapsed after the kit was uploaded to the honeypot. The Y-axis describes the



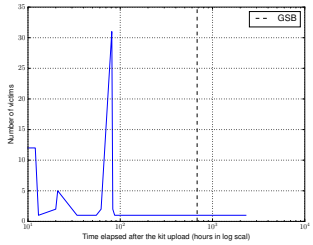
(a) Kit 1 that is detected by GSB and PhishTank



(b) Kit 2 that is detected by GSB and PhishTank



(c) Kit 3 that is detected by GSB



(d) Kit 4 that is detected by GSB

Figure 4.5 – Victims time distribution for the most significant phishing kits

number of visitors that have connected to the phishing kit. The vertical lines correspond to the time upon which Google Safe Browsing or PhishTank has detected the given phishing kit. During the first few days the phisher was able to attract few victims, who connected directly to the entry page. After the random link was published by PhishTank we received many connections from researchers and security companies, which connected to the reported link instead of the entry page of the phishing kit. However, our technique was able to correctly remove these visits from the victim set.

Google Safe Browsing publishes only the MD5 digest of the phishing links, which are represented as host-suffix/path-prefix expressions. These expressions can match arbitrary URLs as long as they have the required host suffix and path prefix. This approach helps to protect against sites where the attacker uses randomly generated sub-paths to evade blacklists [5].

4.5.3 *Victim Time Distribution*

To illustrate and further confirm our measurement of the effective lifetime of phishing kits discussed in Section 4.4.3, we plot the victim

5. https://code.google.com/p/google-safe-browsing/wiki/SafeBrowsingDesign

time distribution from four of the most significant phishing kits, as presented by Figure 4.5.

In general, we observe two different distribution shapes: (1) a skewed right distribution where the majority of victims connected to the phishing kit within a short period of time after the kit was uploaded on the honeypot and (2) a bimodal distribution where two groups of victims connected to the phishing kit during two different periods. Kits 1-3, as presented in Figures 4.5a, 4.5b and 4.5c belong to the first category, having a single group of victims within a short period, followed by a long tail of very few victims. The phishers of the three kits performed probably only one phishing campaign after which they abandoned these kits. Different to the previous kits, Kit 4 in Figure 4.5d belongs to the second category, with two distinct groups of victims. The first group connected to the phishing kit within 24 hours, while the second burst arrived after exactly 70 hours (about 3 days). This is probably a consequence of two different phishing campaigns.

As shown in Figure 4.5, even if phishing kits remain online for a long period of time, they are only active for a short duration after the kit is uploaded to a compromised server. Besides, this effective lifetime is largely unaffected by the time at which Google Safe Browsing (GSB) or PhishTank blacklist the phishing pages, since these services usually become effective only after most victims have already visited the phishing pages. This victim time distribution further supports the way we measure the effective lifetime of the phishing kits described in Section 4.4.3.

4.5.4 *Real-time Email Detection*

Our results show that GSB and PhishTank are not fast enough to blacklist new phishing kits, which leaves victims on their own to identify and protect against phishing attacks. We believe that honeypot systems, as the one described in this chapter, could be deployed to provide an early detection mechanism to promptly identify drop email addresses used by the attackers. These addresses can be further used by the email service providers to disable the email accounts to prevent phishers from retrieving the stolen credentials.

Our custom PHP interpreter records constantly the attempts to send an email even when the attacker makes use of the error control operators (@) [6] to silence all error messages. Each attempt is logged along with the file path of the script that tried to send an email and the destination address. Our system can detect the drop email as soon as an email is sent to the phisher, which may be either triggered by the phisher who verifies the good behavior of the phishing kit or by the first victim who submits some data.

6. http://php.net/manual/en/language.operators.errorcontrol.php

In order to assess the effectiveness of this approach, we manually check the emails sent to the attackers to estimate how many attacker email accounts could have been detected during our study. In total, we have found 68 distinct drop email addresses in the sent box – so each email address has been used in average by two different phishing kits. Unfortunately, only four of these addresses were disabled or were unreachable at the moment when the phishing kits attempted to send the first stolen credentials. This shows that email providers already disable drop email accounts. However, this process can greatly benefit from using a more systematic infrastructure (like the one presented in this chapter) to collect more addresses in an efficient manner.

4.6 CONCLUSIONS

In this chapter we present the design and implementation of a honeypot system especially designed to analyze and disarm phishing kits. Our system provide a new way for an application service provider to monitor the security of its customer applications without compromising the user privacy. Using this infrastructure, we conducted a five-month experiment to understand and measure the entire life cycle of this type of attack.

Different from previous works, our approach is able to measure the effective lifetime of phishing kits, which starts immediately after the kit installation. We are also the first to clearly distinguish the victims from the attackers and the other third party visitors. Our results show that less victims divulge their credentials compared to previous studies conducted in 2009 [180], maybe due to an increased user education in the past seven years against this threat.

5

DECEPTION TECHNIQUES IN COMPUTER SECURITY: A RESEARCH PERSPECTIVE

In this Chapter we look at alternative techniques a service provider can deploy to add an additional security layer in front of its customers applications. In particular, a recent trend both in academia and industry is to explore the use of deception to achieve proactive attack detection and defense – to the point of marketing intrusion deception solutions as zero-false-positive intrusion detection. However, there is still a general lack of understanding of deception techniques from a research perspective and it is not clear how the effectiveness of these solution can be measured and compared with other security approaches. To shed light on this topic, this Chapter presents a comprehensive survey on the use of deception techniques in computer security.

In recent years we have witnessed a surge in advanced cyber attacks, that are rapidly increasing both in number and in sophistication [176]. This threat affects both enterprises and end-users alike, and inflicts severe reputation, social, and financial damage to its victims [1]. Unfortunately, despite the numerous efforts to raise awareness against cyber attacks, attackers are still able to infiltrate their target systems, leveraging multiple attack vectors such as zero-day vulnerabilities, flaws in software configuration, access control policies, or by social engineering their target into installing or running malicious software.

To thwart this vicious trend, the security community has proposed and developed numerous solutions to enhance the security of networks and information systems. Current solutions cover all traditional aspects of security, including intrusion prevention [163], system hardening [135], and advanced attack detection and mitigation [17]. These traditional measures, although essential in any modern security arsenal, cannot provide a comprehensive solution against Internet threats. Hence, complementary solutions have been recently investigated, with the aim to be more proactive in anticipating threats and possibly warn against attacks in their very early stages. In this scope, deception techniques have attracted a considerable amount of interest among the research community [8, 13]. These techniques, initially inspired by the use of deception in the military domain, consists of orchestrating a *"planned set of actions taken to mislead attackers*

1. https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/the-impact-of-targeted-attacks

and to thereby cause them to take (or not to take) specific actions that aid computer-security defenses" [199].

Most recently, during the French President election campaign, the digital team of Mr. Macron created fake accounts with a large number of fake documents [137]. They submitted such fake accounts to phishing attacks, and have successfully delayed the advancement of the attackers by the large quantity of fake documents they created. Interestingly, attackers rushed to modify some of the leaked documents, leaving some traces in their meta-data that provided information about their identity.

During our efforts to summarize existing work on deception and its application in computer security, we realized that many security solutions already implement deception up to a certain level, as also mentioned by Cohen [42]. The diversification of military deception and its application in the cyber space has been also actively developing over the last few decades [83, 160]. Nonetheless, from a research perspective, we still lack a global understanding about deception techniques and their application in information and communication systems security. In particular, there is not yet a wide consensus among the research community about what are the main goals and technical challenges to achieve when using deception techniques as a defense mechanism. The real benefits for such techniques are also not clearly emphasized, nor the way deception techniques may complement other traditional security solutions. The modeling, deployment, update, and evaluation of deception techniques are also scarcely addressed in the literature – often resorting to qualitative statements and thus making difficult to compare such techniques with other approaches.

This Chapter discusses all these issues by providing an overview of deception techniques in computer security, divided in four separate categories:

CLASSIFICATION AND CURRENT APPLICATIONS: A first question to address when referring to deception techniques is why should we use such techniques, and how do they complement other traditional security mechanisms [7, 8]. Only two decades ago, researchers were still wondering whether deception may be acceptable from a legal and ethical perspective [42]. Today, it is obvious that such questions are no longer an issue, opening the door to a large number of proprietary (e.g TRAPX [2], CANARY [3], ThreatMatrix [4]) and open source (e.g. DCEPT [5], Canarytokens [6]) solutions that explicitly mention and use deception for com-

2. https://trapx.com/
3. https://canary.tools/
4. https://attivonetworks.com/product/deception-technology/
5. https://github.com/secureworks/dcept
6. https://github.com/thinkst/canarytokens

puter security defense. A main goal of this Chapter is thus to propose a comprehensive classification, survey the current applications of deception in computer security, and shed light on the key technical domains where deception has been implemented.

MODELING: Deception modeling refers to the set of rules and strategies that drive the defender into designing and integrating deception within the architecture and workflow of a target system. During our survey of scientific literature, we identified multiple deception strategies that leverage different properties of the target system, and different assumptions about the attacker behavior. For example, Cohen et al. [45, 47] leveraged the use of attack graphs, which provides a compact representation of known attack scenarios and the way attackers can break into a target system. The authors integrated deception techniques in different position of the graph, in order to lure attackers and to drive them towards fake targets. In [37], authors proposed an alternative approach, leveraging models based on game theory, in order to describe different interaction sequences between attackers and the target system, and to apply deception while maximizing the defender's gain.

In this Chapter, we survey existing deception modeling techniques and we discuss their main benefits and limitations. In particular, we observe that most deception modeling techniques either cover only specific attacks and use cases, or they try to adapt and implement high-level strategies whose applicability remains questionable.

DEPLOYMENT: Another key aspect of deception is the way it is deployed inside a target environment. The mode of deployment refers to the technical environment where the deception is being implemented inside a target system [160]. In this scope, deception may be either implemented as a standalone solution (e.g. honeypots), or fully integrated within the target system (e.g. fake documents [186]). Moreover, deployment also includes the placement strategies (either manual or automated) that place deception at specific locations within the architecture and workflow of a target system. To be effective, deception elements needs to be generated to achieve perfect realism with regard to real data and system integration. However, it is still unclear how the deployment and placement strategy affects the effectiveness of the solution. Moreover, while the importance of retraining machine learning models or update attack signatures is well known in the intrusion detection domain, the importance of updating the deception element is still largely unexplored.

EVALUATION: The last aspect that we cover in this survey is the experimental setup and the way previous work evaluate the pro-

posed deception techniques. We discuss the main challenges researchers are faced with when evaluating deception, and discuss how these problems may drastically limit the soundness of obtained results (both including the efficiency and accuracy of the proposed solutions). Other solutions are typically evaluated by measuring their detection and false alarm rates – both of which are difficult to assess in the deception domain.

Finally, we provide insights and key features that should be taken into account in the future when evaluating deception techniques.

Motivated by these key questions, this chapter presents a systematic organization of previous work on deception techniques and their application in the cyber security domain. It discusses the main technical challenges that were addressed using these techniques and emphasizes the gaps and the pieces of the puzzle we still miss in this complex field of research.

5.1 DEFINITION & SCOPE

5.1.1 *Deception techniques: concept and terminology*

One of the first documents to mention the term *deception* as a way to enhance the security of computer systems is "The Cuckoo's Egg" book [173], published by Cliff Stoll in 1989. The author describes how he set up a fictitious system environment, including a user account populated with a large number of fake documents to deliberately attract and delay an attacker, while tracking his incoming connections to reveal his origin and identity. Such fictitious environment, designed to capture the interaction with an attacker, later became popular as a *honeypot* [105]. In 1994, a system called Tripwire [99] offered a new approach for file integrity checking by planting fictitious files and placing them under a file integrity monitor to detect intruders [98]. The concept of planted fictitious files have then evolved into a diversified set of solutions including *honeyfiles* [198], and *decoy* documents [27].

In the following years, deception has been routinely used to protect computer systems and networks. Honeypots became a popular solution for the monitoring, analysis, understanding, and modeling of attackers' behavior [42, 133, 150, 172]. In 2003, soon after the first computer honeypot was released, the concept of *honeytoken* was proposed by Augusto Paes de Barros [143]. Then Spitzner [171] broadened this concept to cover "*a digital or information system resource whose value lies in the unauthorized use of that resource*" and brought it to the attention of a larger public. He illustrated practical examples of honeytokens, including bogus social security numbers and fake credentials, which have later inspired many follow-up contributions such as honey pass-

words [90], honey URL parameters [147], database honey tokens [22, 38] and honey permissions [93].

The use of deception techniques for computer security has been widely driven by the analogy with more conventional deceptive military tactics. In this scope, the concept has been broadened towards intrusion detection (IDS) and prevention (IPS) systems. Instead of indicating to the intruder a violation of a security policy, deception consists in responding to attackers with some pre-defined decoy actions, such as fake protocol messages [67], response delays [92], and crafted error messages [123]. For example, instead of blocking a detected intrusion on the web server, Katsinis and Kumar [96] examined the use of a deceptive module that analyzes incoming HTTP requests and advises deceptive responses in case of an ongoing attack.

It is worth mentioning that during our efforts to summarize existing work on deception techniques, we observed that many traditional security solutions already implement deception up to a certain level. For example, widely known techniques such as software diversity [106], as well as anti-forensics techniques [63], are deceptive in nature, as they aim to hide and conceal system information against unauthorized access. Similarly, security scanners and crawlers mostly behave in a deceptive way in order to mimic normal users and hide their real purposes. Malware analysis systems also try to deceive malicious binaries by executing them in an instrumented sandbox as if they were running on a real victim terminal. To further complicate this already multifaceted domain, the existing literature often adopts a confusing terminology – interchangeably using terms such as concealment, fakes, baits, decoys, traps, and honey(pots | tokens | files | accounts | ...).

5.1.2 *Scope of this survey*

The first part of this survey aims at providing a multi-dimension classification of deception techniques in computer security. It reaches this goal by organizing the existing deceptive approaches along four orthogonal directions, as discussed in Section 5.2. The second part of the survey emphasizes instead the research gaps, by focusing on open problems in the modeling, deployment, and evaluation of deception techniques.

Unfortunately, the concept of deception is too broad to cover in a single document – as it manifests in different forms in almost all security domains. Therefore, in the rest of the document we will only focus on those techniques that are used to deceive an attacker while he is interacting with his target. This definition (sometimes called *Intrusion Deception*) covers only those solutions that can be applied to secure real systems. It rules out instead other loose forms of deception adopted in the security field (such as faking the user agent of a web scanner) as well as standalone deceptive systems used to

monitor the overall threat and attack landscape (such as standalone honeypots [134]).

5.2 CLASSIFICATION

In this section we introduce a classification of deception techniques along four orthogonal dimensions, including the unit of deception, the layer where deception is applied, the goal of the deception solution, and its mode of deployment.

5.2.1 *Multi-Dimension Classification*

Because of the wide spectrum of deception techniques and the different ways they may apply to computer security, it is very difficult to classify all of them along a single dimension that successfully incorporates the different research aspects. Traditional military deception classifications (such as [42]) and semantic classifications (such as [159]) allowed us to understand the use of deception techniques from different angles, which however are difficult to match with the needs of computer security. Moreover, most previous classifications considered only one dimension, such as the component, the granularity of each technique [7], or the layer where deception is applied [149]. Unfortunately, these mono-dimensional classifications miss other aspects of deception that are of equal importance, such as the threats covered by each technique, and the way they can be integrated inside a target system.

To address these limitations, we introduce in this chapter a new classification system based on four orthogonal dimensions. In our system, categories are mutually exclusive, which means that each existing deception technique may be classified within only one single four-dimensional vector Our classification is also complete, so all existing deception techniques find a place in our schema. The four dimensions that we use for our classification are defined as follows.

Goal of deception captures the main purpose a specific deception technique is trying to achieve. This could be either to improve and complement attack detection, to enhance prevention, or to mitigate successful attacks. The first category includes those solutions designed to detect an attack, typically because it interacts with active traps or because it uses passive decoy information that are intentionally left accessible to be discovered by the attacker (e.g., decoy documents [21]). The second category covers instead those mechanism that aim at confusing or distract attackers from the real targets before an attack occurs (e.g. the deployment of deceptive network topology [179]). Finally, the last category targets on-going attacks and tries to reduce their

damage, for instance by replying in a delayed manner [92] or by redirecting attackers to a safe copy of the target system [12, 16].

Unit of deception refers to the granularity of the decoy asset that is used to implement deception. We use the same definition of granularity introduced by Almeshkah et al. [7], which includes the following units of deception: *decision* (e.g., by accepting a connection towards an unused IP address [26, 116]), *response* (e.g. a forged network response [26]), *service* (e.g., a decoy service [43]), *activity* (e.g., a decoy computation activity [101]), *weakness* (e.g. a simulated vulnerability [16]), *performance* (e.g. a delayed response [92]), *configuration* (e.g. a fake network topology [179]) and *data*.

We further refine the data unit into separate types, such as *parameter* (e.g., a honey URL parameter [147] or a honey form field [97]), *file* (e.g., honey files [198] and decoy documents [21]), *account* (e.g. honey account [28, 39]), *user profile* (e.g., honey profiles [191]), *source code* (e.g., decoy source code elements [145]) and *database record* (e.g., database honey token [38]).

Layer of deception indicates the layer in which the deception element is applied. While there are different ways to organize the layers of an information system, we use a standard classification divided in network, system, application, and data layers. The network layer covers deception techniques that are accessible over the network, and that are not bound to any specific host configuration. The system layer covers host-based deception techniques. The application layer covers deception techniques that are linked to specific classes of applications, such as web applications or databases. Finally, the data layer covers deception techniques that leverage user-specific data, such as fake accounts or fake documents.

Deployment of deception characterizes the way a deception technique may be integrated within a target system. Possible deployment modes include (1) *built-in* deceptions solutions that are integrated in the system at the design phase (e.g., in the source code [92]), (2) deceptions that are *added-to* the system during operation (e.g., documents inserted in a file system [186]), (3) deceptions that are set *in-front of* a target system (such as a proxy or gateway [30]), and finally (4) *isolated* solutions separated from the target system (e.g., fake accounts of a decoy server [28, 39]).

Table 5.1 presents a survey of previous work on deception and classifies them over the four aforementioned dimensions. Note that few studies that are referenced in Table 5.1 do not explicitly mention deception in their core contributions. Moreover, since deception is a generic concept that may apply to different aspects of security, certain studies introduce deception as an *add-on* functionality within a

more comprehensive security framework. Table 5.1 lists all existing work as long as their contribution fits with our definition of deception introduced in Section 5.1.

5.2.2 *Overview of Intrusion Deception Techniques*

During our survey, sometimes we found it particularly challenging to associate a given publication to a specific deception category, since many techniques may have been presented in a single work. Therefore, to keep a consistent classification of existing work on deception, we associate each publication to its main deception technique, or to the one that has not been introduced before in case of multiple contributions.

For sake of clarity, since our classification leverages four distinct dimensions, we organize this section based on the layer of deception. But we discuss all four dimensions as part of our classification.

Layer I: Network

The network-based deception techniques we observed in the literature were mostly designed to address three categories of threats: *network fingerprinting*, *eavesdropping*, and *infiltration and attack propagation*.

SCANNING & FINGERPRINTING – These typically occurs during the early stage of an attack, in particular the reconnaissance phase, and enables an attacker to acquire information about the network topology and available assets by fingerprinting and scanning the network.

One of the first deception techniques that offered to interfere with the reconnaissance phase in order to obfuscate its results relied on sinkholing attack traffic, by redirecting malicious traffic to a set of fake machines that mimic the behavior of real terminals on the network [26]. Such decoy machines are commonly known as network tarpits [116]. They create sticky connections with the aim to slow or stall automated scanning, and to confuse human adversaries. Shing [167] further brought a few improvements to this idea by tuning the different options of the sticky connections to conceal network tarpits and prevent them from being easily recognized.

Another way to lure attackers and to randomize the outcome of their fingerprinting attempts was proposed by Le Malécot in [109]. They authors introduced a technique to skew the topology of the target network through random connection dropping and traffic forging. For example, Trassare [179] defeated a particular type of scan, traceroute probes, by misleading an attacker through constantly revealing a false network topology.

Another type of threat, OS Fingerprinting, allows an attacker to gain valuable information about available Operating Systems (OS), and so to identify potential flaws and vulnerabilities. To conceal OS-

Reference	Technique	Unit	Layer	Goal	Deployment
[116]	Network Tarpit	Decision	Network	Mitigation	Added-to
[26]	Network Tarpit	Decision	Network	Mitigation	In-front of
[167]	Network Tarpit	Decision	Network	Mitigation	Added-to
[109]	Traffic forging	Response	Network	Mitigation	In-front of
[179]	Deceptive topology	Configuration	Network	Prevention	In-front of
[168]	OS obfuscation	Response	Network	Mitigation	In-front of
[28]	Honey accounts	Configuration	Network	Detection	Isolated
[39]	Honey accounts	Configuration	Network	Detection	Isolated
[47]	Deceptive attack graph	Configuration	Network	Prevention	Added-to
[45]	Deceptive attack graph	Configuration	Network	Prevention	Added-to
[43]	Decoy services	Service	Network	Prevention	Added-to
[150]	Decoy services	Service	Network	Detection	Added-to
[161]	Deceptive simulation	Activity	Network	Prevention	Added-to
[101]	Decoy computation	Activity	System	Prevention	In-front of
[183]	Deceptive simulation	Service	System	Mitigation	Added-to
[189]	Multi-layer deception	Multiple	Multiple	Detection	Added-to
[130]	OS obfuscation	Parameter	System	Mitigation	Added-to
[158]	Fake honeypots	Configuration	System	Mitigation	Added-to
[93]	Honey permissions	Configuration	System	Detection	Built-in
[162]	Decoy network device	Service	System	Detection	Added-to
[123]	Software Decoys	Response	Application	Detection	In-front of
[16]	Honey patches	Weakness	Application	Mitigation	In-front of
[50]	Software trap	Weakness	Application	Detection	Added-to
[92]	Delayed response	Performance	Application	Mitigation	Built-in
[12]	Shadow honeypots	Service	Application	Mitigation	In-front-of
[30]	Decoy hyperlinks	Parameter	Application	Detection	In-front of
[65]	Decoy hyperlinks	Parameter	Application	Detection	Added-to
[121]	Honey URL	Parameter	Application	Detection	Added-to
[147]	Honey URL parameters	Parameter	Application	Detection	Built-in
[97]	Decoy form field	Parameter	Application	Detection	In-front of
[185]	Honey accounts	Configuration	Application	Detection	Added-to
[25]	Honey password	Account	Data	Prevention	Added-to
[90]	Honey password	Account	Data	Detection	Added-to
[89]	Honey encryption	Account	Data	Prevention	Built-in
[29]	Honey accounts	Account	Data	Detection	Isolated
[4]	Honey accounts	Account	Data	Detection	Isolated
[138]	Honey accounts	Account	Data	Detection	Isolated
[198]	Honey files	File	Data	Detection	Added-to
[107]	Honey files	File	Data	Detection	Added-to
[38]	Database honey tokens	Record	Data	Detection	In-front of
[22]	Database honey tokens	Record	Data	Detection	Added-to
[142]	Database honey tokens	Record	Data	Detection	Built-in
[145]	Decoy source code	Code	Data	Detection	Added-to
[27]	Honey files	File	Data	Detection	Added-to
[21]	Honey files	File	Data	Detection	Added-to
[186]	Honey files	File	Data	Detection	Added-to
[136]	Honey files	File	Data	Detection	Added-to
[117]	Honey files	File	Data	Detection	Added-to
[94]	Honey web pages	File	Data	Detection	Added-to
[191]	Honey profiles	Profile	Data	Detection	Added-to
[174]	Honey profiles	Profile	Data	Detection	Added-to
[51]	Honey profiles	Profile	Data	Detection	Added-to

Table 5.1 – Detailed overview of deception techniques

related information and prevent it from being fingerprinted by an attacker (e.g. using the Nmap tool), multiple deception techniques have been proposed – that offer to mimic the network behavior of fake operating systems [130, 168] in an effort to mislead potential attackers. The use of deception techniques against this category of threat mainly aims at confusing attackers and delaying their advancement.

EAVESDROPPING Network eavesdropping is a challenging attack to detect using traditional detection systems since the attacker usually remains silent on the network. Deception techniques have been introduced as an alternative way to protect against such attacks. A deception technique that is particularly relevant in this area is the use of honey accounts. These are usually dummy credentials that are shared over the network in order to be captured and used by elusive attackers, thus revealing their presence in specific networks such as wireless [28] and Tor [39] networks.

INFILTRATION & PROPAGATION Cohen et al. in [45, 47] conducted red teaming experiments on deception, through leveraging structured attack graph representations, in order to drive attackers into following fake attack paths that would distract them from their real targets. Similarly, HoneyD [150] and the Deception ToolKit [43] offer to spawn multiple fictitious services and network IP addresses in order to fool the attackers and make them attack false targets. In addition to their application in traditional networks, deceptive simulation techniques [161] have been also applied in the context of industrial control systems. These tools offer to monitor the network topology and to create fake but indistinguishable attack targets that are aimed to fool attackers and conceal potential victims.

Layer II: System

Deception techniques that have been implemented at the system layer were mainly aimed at addressing two categories of threats, namely *external attacks* and *insider threats*.

SYSTEM COMPROMISE Wang et al. [189] suggested the use of deception techniques in an effort to enhance the detection of system compromise attempts by introducing a multi-layer decoy framework that included decoys for user profiles, files, servers, and network / system activity. These decoys were supposed to conceal the real assets of an organization and protect them against targeted attacks. In a similar approach, Rrushi [162] proposed the use of a decoy Network Interface Controller (NIC) for Windows operating systems. This decoy interface was intentionally set in order to lure and detect malicious software that may be running on the system. The idea is that benign software is not expected to use the decoy interface, which is

how the authors were able to detect other malicious applications running on the system.

As opposed to the previous techniques that were mainly aimed at enhancing detection, Rowe et al. [158] adopted a proactive deception approach that aims at preventing system compromise attempts. They suggested the use of fake honeypots that make ordinary but critical systems appear as real honeypots, which may confuse an attacker and turn him away from the compromised system. Similarly, Urias et al. [183] offered to clone and migrate machines that are suspected of being compromised, and to place them in a deceptive environment where network and system configurations are duplicated to mimic the real network environment. Lastly, Kontaxis et al. [101] proposed to duplicate multiple times the entire application server to generate decoy computation activities.

INSIDERS To detect and mitigate insider threats, Kaghazgaran and Takabi [93] offered to extend role-based access control mechanisms with honey permissions. These are fake permissions in the sense that they assign unintended access permissions to only fake versions of sensitive system assets. By monitoring attempts to access or modify such fake assets, the authors are able to detect insiders who triggered these malicious attempts. Note that many other techniques applied at the data layer, such as decoy documents and fake source code, have also been proposed to deal with insider threats, and we discuss them in the last part of this section.

Layer III: Application

State of the art deception techniques applied at the application layer mainly address two threat categories, which are host-based software compromise and remote web-based attacks.

SOFTWARE COMPROMISE Deception techniques have been extensively used in the literature as a way to protect software from commonly known vulnerabilities. They usually consist of deceiving attackers by either pretending fake (non-existent) vulnerabilities, or by randomly responding to common vulnerability scan attempts. For example, a straightforward deceptive response would be to simulate system saturation by randomly adding delays in order to deceive potential adversaries [92]. Michael et al. [123] introduced the notion of intelligent software decoys that detect and respond to patterns of suspicious behavior (e.g. the interaction between a worm and the system component that it tries to infect), and maintain a repository of rules for behavior patterns and decoying actions. Araujo et al. [16] converted software patches into fake but valid-looking vulnerabilities (aka "honey-patches") that drastically limits the attackers capability to determine the successfulness of their attacks. In this scope, and upon detection of an attack exploiting the fake vulnerability, the sys-

tem seamlessly forwards the attacker to a vulnerable decoy version of the same software. Authors further extended their honey-patch system with an instrumented compiler that automatically redacts secrets in order to elude attackers and to reduce their chances of finding existing and unpatched vulnerabilities [15]. In [12], Anagnostakis et al. introduced shadow honeypots that extend honeypots with anomaly-based buffer overflow detection. The shadow honeypot is an instance of the target application, and shares its context and internal state. It is used to process anomalous traffic and to enhance the accuracy of anomaly detection.

Finally, Crane et al. introduced in [50] software traps, that are dissimulated in the code as gadgets, and detect return-oriented programming attacks. These traps detect and notify an ongoing attack as soon as they are manipulated by the exploit.

WEB ATTACKS Brewer et al. [30] proposed a web application that embeds decoy links. These links are invisible to normal users, but are expected to be triggered by crawlers and web bots that connect to the application. Similarly, Gavrilis et al. [65] presented a deceptive method that detects denial of service attacks on web services by using decoy hyperlinks embedded in the web page.

Another approach to deceive web-based attacks also consists of using fake information disguised as web server configuration errors. Only malicious users are expected to manipulate or exploit these errors, which expose them to detection by the system. In this scope, Virvilis et al. [185] introduced honey configuration files, such as `robots.txt`, including fake entries, invisible links, and HTML comments that indicate honey accounts, in order to detect potential attackers. Other studies proposed decoy forms [97] and honey URL parameters [147] that display fake configuration errors in an effort to mislead attackers and to protect the target system.

Layer IV: Data

This section discusses state of the art techniques that use fake or decoy data to deceive attackers. They mainly protect against four categories of threats, namely *identity theft*, *data leakage*, *privacy violation* and *impersonation*.

IDENTITY THEFT Honey accounts have been used in the literature to track phishers [121], detect malware [4, 29] and also provide possibilities for researchers to study the malicious activities performed on stolen webmail accounts [138]. In the same vein, Lazarov et al. [107] created five fake Google spreadsheets containing decoy banking information and wire transfer details, in order to shed light on the way that cyber-criminals use these fake spreadsheets. There are also other approaches that applied deception techniques to protect stolen user passwords. For example, to protect against situations where hashed

user passwords have been leaked, Juels et al. introduce in [90] honeywords (false passwords) in order to conceal true authentic passwords. Bojinov et al. [25] introduced instead the concept of a theft-resistant password manager that executes on the client side, and that randomly generates new password instances. Finally, Juels et al. [89] proposed "honey encryption", which creates a ciphertext that, when decrypted with an incorrect key or password, result in a valid-looking decoy message.

DATA LEAKAGE AND INSIDERS To mitigate or report a data leakage, multiple studies [21, 27, 186] suggested the use of decoy documents that implement honeytokens, and beacon alerts that call home when they are opened. Alternatively, honey tokens that mimic sensitive information have been also integrated within databases (e.g. [22, 38]) – mainly as a way to detect insiders who scan the database to obtain an unauthorized access to data. In the same vein, Park et Stolfo [145] generated fake but believable Java source code to detect the exflitration of proprietary source code.

PRIVACY VIOLATION Honey files that raise an alert when accessed or modified have been used in the literature in order to detect privacy violations for documents shared on web hosting providers [136] and Peer-to-Peer networks [117].

More recently, Kapravelos et al. [94] proposed honey web pages that adapt the page structure and content according to a browser extension's expectations. These honey web pages allowed the authors to identify malicious behavior in the form of privacy violation in browser extensions.

IMPERSONATION De Cristofaro et al. [51] created and promoted 13 honey Facebook pages to study the fake likes that they would receive. Honey profiles have also been deployed on social networks in order to identify accounts that are operated by spammers [174, 191].

5.3 MODELING

In this section, we look at theoretical models that had been proposed for deception techniques in computer security. In particular, we have identified two main axes of deception modeling in the literature. First, many works have proposed models to support planning and integration of deception in a target infrastructure or system. These models mainly propose some sort of methodology (that can either be process, a probabilistic model, or a practical model based on attack graphs) to design where, when and how deception may be integrated in computer security. Second, few works have tried to understand the affects of deception in computer security by modeling

the interaction between attackers and deception-enabled defenders using game theory.

5.3.1 *Deception Planning*

We group the proposed models to plan deception into three categories by the modeling method that are 1) process model that defines the desired process and how they should be performed, 2) probabilistic model that uses probability to evaluate the benefits and cost of deception, and 3) practical model that used attack graphs to build deception.

5.3.1.1 *Process Model*

Yuill [199] presented a process model covering four major steps: 1) deception-operation development, 2) deployment, 3) target engaged and 4) continuation decision and termination. The first step involves the planning of goals, the identification of possible targets, the creation of the deceptive element, and the preparation of the mechanism to engage the target. The next step consists of deploying the deception scenario at the location that is visible to the potential target. The target is engaged once he perceives and believes the deception scenario, and subsequently takes the planned action, which may be reported by properly designed feedback channels. Finally, a decision on whether to continue or terminate the deception is taken based on the results and the efficiency of the deception operation.

Similarly, Almeshekah and Spafford [7] proposed a model that includes three general phases: design, implementation and integration, and monitoring and evaluation. The first phase consists of specifying the strategic goals, defining the way that the target may interact with deception, creating the deception story, and identifying the feedback channels. Then defenders should integrate deception within a real environment, instead of being a separate disjoint component. Finally, the previously established feedback channels should be carefully monitored in order to enhance deception. Heckman et al. [75] presented a similar but iterative deception life cycle management process while De Faveri and Moreira [52] proposed a similar but adaptive process model. Hassan and Guha [73] proposed an abstract model based on a state machine with four states (default, ready, production, and determine) to provide a basic elementary view of the aforementioned models.

De Faveri et al. [53] designed a goal-based approach to incorporate deception in the software development process in three separate phases. The first phase consists of modeling the system architecture and its goal in a specific domain of application. Then the security requirements are identified and a threat model is produced. The last step requires the application of the deceptive solution based on previ-

ously built models. The applied deception model is actually similar to the aforementioned process models.

Finally, Yuill et al. [200] proposed a model that describes deceptive methods by the way that defenders may use to disable the attackers to discover a hidden asset. More precisely, the process model affects the attackers ability or behavior, by altering the results of their direct observation, the findings of their investigation, and the information they learned from other users.

5.3.1.2 *Probabilistic Model*

Rowe [155] provided a probabilistic model of attacker beliefs in false excuses such as system crush, communication breakdown, and also the attacker's suspiciousness about whether he is being deceived. This model is helpful for defenders to plan when and how to deceive, while monitoring the attacker's belief in the proffered excuses. Rowe [154] also proposed a cost-effective probabilistic model to assess the cost and benefits while planning deception. Wang et al. [189] modeled the design of the multilayered deception system including decoys of user profile, files, servers and network or system activity as a problem of optimization which aims to reduce the cost of deception deployment and the loss in case of a successful compromise.

5.3.1.3 *Practical Model*

Cohen et al. [45, 47] modeled the process and path that attackers might use to compromise a computer using attack graphs. By introducing false targets in the attack graph, they modeled how to drive the attackers away from real targets.

5.3.2 *Interactions between Attackers and Deception Techniques*

To the best of our knowledge, only two studies have modeled the affects of deception with respect to attackers using game theory. Carroll and Grosu [37] modeled how deception affects the attack-defense interactions based on a game theory, where the players (defenders and attackers) have incomplete knowledge of each other. In this game, defender can deploy two deception defenses by either concealing a legitimate server as a honeypot, or by making a honeypot to look like a legitimate server. Gutierrez et al. [70] presented a primer of modeling the interaction between adversaries and system defenders using hypergames. Using this approach, the authors were able to model misperceptions resulted from the use of deception.

5.4 DEPLOYMENT

Most deception techniques take advantage from their interactions with the remote attacker. Hence, the efficiency of such techniques hinges largely on their credibility, their coverage, and the location in which the deceptive elements are placed inside the target environment. In this section, we discuss these fundamental properties and we survey how these requirements were fulfilled in previous work.

In particular, the rest of the section focuses on the following four aspects: 1) how deception is deployed and integrated into a target environment, 2) how previous work approached the problem of optimal placement of deception elements, 3) how to achieve realistic and plausible deception, and finally 4) how to monitor, update and replace deceptive elements already deployed on the field.

5.4.1 *Mode of Deployment*

We start our deployment study by looking at how deception techniques are positioned with respect to the target environment under protection. The techniques that we introduced in the previous section often adopt different deployment strategies, based on their granularity and on the objectives the defender wants to achieve when adopting a given deception technique. According to the taxonomy we presented in Section 5.2, we classify the different modes of deployment into four categories, depending on whether the deception is built inside the target system, it is placed in-front of it, it is later added-to the system, or finally it comes as a isolated standalone solution. Unfortunately, not all existing deception techniques come with a clear or unique deployment strategy. Therefore, this section covers only those techniques whose deployment may be clearly identified with respect to the target system.

Built-In

Only few studies in the literature suggest integrating deception already at the system design phase. In this scope, honey encryptions [89] offer to protect messages by creating dummy honey tokens and instilling them in the system by design. Similarly, Kaghazgaran [93] proposed to embed, during the policy setup phase, honey permissions inside role-based access control models. In the same category, some other deception techniques have been integrated directly in the source code of an application. Examples include the injection of random delays in order to modify the normal behavior of a web-based search engine [92], honeytokens added during the creation of a new database using aspect-oriented programming [142], and setting honey HTTP request parameters in a web application in order to lure attackers by exposing dummy URL attributes [147].

In-Front of

Deception techniques that are implemented in-front of a target system mainly consist of interposing the deception elements between the attacker and the system. This can be achieved by multiple ways, depending on whether the deception is integrated at the network, at the system, or at application layer. For instance, at the system layer the deployment requires to intercept the execution flow of a process or application, while at the network layer it is necessary to intercept the traffic, often using reverse proxies and trusted certificates (in case of encrypted connections).

We identified two distinct methods in the literature that offer to modify the execution flow of an application in order to deploy deception. First, Michael et al. [123] instrumented the software component to implement decoy actions. Other work leveraged specific application features. In particular, Cenys et al. [38] used the Oracle database to intercept insert, select and update operations and to integrate honeytokens. Similarly, the modular design of the Apache web server enables to register a hooking module that can implement deception techniques [6, 30, 97].

At the network layer, deception is often used to mitigate network scanning and fingerprinting. These solutions are usually deployed on specific network devices, such as routers [179], gateways [26, 101, 168], firewalls [109], or on dedicated monitoring modules (anomaly detection [12]). Note that it may also be possible to achieve similar results without intercepting the traffic. For example, the authors of [116, 167] proposed techniques to monitor and reply to unanswered address resolution protocol (ARP) requests in order to produce false network topologies, and to reduce the attacker's ability to scan the network.

Added-To

This category includes techniques that are added or integrated into the system at runtime. Many existing approaches leverage this mode of deployment, and use deception as a way to complement other traditional security mechanisms. In particular, decoy assets (IPs, services, applications, vulnerability or data) are usually integrated in the infrastructure in order to mislead attackers and turn them away from other sensitive assets. To build such decoy assets, previous work adopted different approaches, such as the use of honey patches that simulate fake vulnerabilities [16], simulation [161], or by duplicating legitimate services [183]. Another interesting trend consists of adding artifacts to a real benign service in order to make it appear for attackers as a honeypot [158].

Previous studies have also suggested to use honey files to detect privacy violations, by placing them on online sharing platforms and

cloud hosting services [117, 136]. In the same vein, other studies suggested to use the same concept of honey files in order to detect data leakage within corporate or private networks [21, 27, 186, 189].

The concept of (honey-)assets has also been extended to honey passwords [25, 90], honey profiles [51, 174, 191], and even honey hyperlinks in the context of web applications [65], in order to detect malicious usages and abuses of sensitive applications and data.

Standalone

This category includes deception techniques that are isolated from the target system. While honeypots are the most common type of isolated deception system, other methods have been also explored in the literature, such as the use of honey accounts to drive attackers into connecting to a decoy service that is isolated from the real authentic service [28, 39]. Onaolapo et al. performed a real world experiment in which they (deliberately) leaked fake webmail accounts that are isolated from any other benign accounts on the webmail service [138]. During their experiments, the authors had been able to monitor abuses of leaked webmail accounts and to learn more about the tactics used by attackers.

5.4.2 *Placement*

We now look at different strategies that have been proposed to support the placement of deceptive components. This covers the way existing deception techniques are deployed inside a target system (either manually or automatically), and the way these techniques are displayed to attract and deceive attackers. We organize this section into three parts, based on the ultimate objective of deception: attack prevention, detection, or mitigation.

5.4.2.1 *Attack Prevention*

The application of deception techniques for intrusion prevention currently takes two different aspects. On the one hand, it deters an attacker by displaying, for example, false targets among the path that attackers may go through [45]. In this case, the placement of false targets is guided by the knowledge of existing attack graphs.

On the other hand, deceptive techniques may also be used as *chaff*, to confuse an attacker by hiding important information in a large amount of data. For example, to protect real user passwords Bojinov et al. [25] offer to conceal them through dissimulating a large number of other decoy passwords. Similarly, Kontaxis et al. generate decoy computing activities [101], and Rrushi et al. generate decoy network traffic [161], in order to respectively dissimulate sensitive applications and network connections, so they may be less likely to be identified

and compromised by an attacker. In this case, decoys are usually placed alongside the valuable assets.

5.4.2.2 *Attack Detection*

Many deception placement strategies have been previously explored in order to enhance attack detection. We may classify these strategies into two different categories, depending on whether deception is being integrated inside the target system, or whether it is publicly exposed in order to enhance threat intelligence and to anticipate unknown attacks.

INTEGRATED DECEPTION PLACEMENT In the first category, Gavrilis et al. [65] use decoy hyperlinks in order to protect a website against flash crowds. The authors develop a placement strategy where they represent the target website as a undirected graph. The graph nodes represent the individual pages of the website, and the edges represent the hyper links between the pages. The attacker is modeled as a random walker, and the strategy consists of finding the optimal subset of decoy hyperlinks that minimizes the survival probability of the random walker in the graph.

Another example includes the placement of honeyfiles inside a target file system. First, Bowen et al. [27] manually placed the honey files at selected locations in the file system. Alternatively, Voris et al. [187] developed an automated placement strategy that first searches the target file system to locate folders that have been recently used, and also the folders of large number of files sharing similar features such as extensions. These folders are further selected as favorite locations to place honey files. The same authors further distinguished two modes of automatic deployment: integrated and separated [186]. In an integrated deployment, decoy documents are co-located with real documents, while in a separated placement, decoy documents are deployed in isolated subfolders within the same root directory. Finally, Whitham [193] suggested a more aggressive placement strategy, offering to place honey files in all directories of the file system.

Finally, researchers have also studied the placement of decoy permissions within role-based access control policies[93]. The placement strategy consists first in identifying high risk permissions. These are further duplicated as honey permissions, and their corresponding objects are duplicated as decoy objects. The authors select the sensitive roles in the policy whose risk exceeds a predefined threshold, and assign the honey permissions to these roles for monitoring and detection.

ISOLATED DECEPTION PLACEMENT Isolated deception techniques follow different placement strategies, which may also depend on the defender's objectives and the monitored attacks. In particular, honey

profiles are being commonly created on social networks, and used in order to monitor and detect spam and infection campaigns [174, 191]. Different strategies can be used to advertise and share these decoy profiles, including the use of dedicated advertisement services, and other underground services [51].

On the other hand, honey files have been manually leaked on public hosting services [136] and file sharing networks [117]. This approach is commonly used to attract potential attackers, and as early warning system in case of unauthorized access to personal user data. Moreover, honey accounts are actively provided to be collected and exfiltrated by malware, to study the goals of the attackers [4, 29]. Similar information has been also deliberately distributed on malicious hacking forums to attract attackers and infiltrate criminal groups [138]. Finally, honey URLs are used when testing and connecting to phishing sites in order to track the activities of the attacker [121].

Apart from honey files, the placement strategy has been however under-studied for the majority of deception techniques. For instance, previous work has discussed how to implement honey tokens in a database [38], but it is still unclear how these tokens should be placed – e.g., in a separated fake table or alongside legitimate data. A similar problem exist for honey parameters placed in a web application to enhance attack detection. Honey parameters can be added in any field of HTTP protocol and in the HTML source code, which results into a large possibility of possible placements. Moreover, a web application usually consists of multiple and various services. Currently, there is no systematic approach to place honey parameters to achieve optimal attack detection. The fundamental difficulty in achieving this is the lack of methods to describe the web application logic. As Li and Xue [112] have pointed out, there is actually *"an absence of a general and automatic mechanism for characterizing the web application logic"*, which is however a prerequisite to achieve an effective deception placement.

5.4.2.3 *Attack Mitigation*

When deception techniques are used for the purpose of attack mitigation, the placement strategy is generally in line with the type of attack whose damage needs to be reduced. For instance, upon the detection of a network scan, mitigation techniques such as network tarpits [116, 167], and traffic forging [109] can be put in place. To mitigate software exploits, the safe duplication of target application enables to redirect attackers [12, 16], and even contain them [183]. Note that some techniques, such as chaff, can be deployed both for attack prevention and mitigation. For instance, if an attacker is able to steal the password file, populating it with a large number of fake entries [25] could somehow mitigate the password theft attack.

5.4.2.4 *Summary*

Most previous work focus on the presentation of a technique or on new deception elements, but rarely explain where and how such elements should be placed in the system to protect. When they do, it is unclear if the proposed placement provides any guarantee of being better than other approaches. In fact, without a clear methodology to test and measure the accuracy of deception techniques, a problem we will discussed in more details in Section 5.5.1, it is impossible to compare different placement strategies and conclude which one is better among several options.

5.4.3 *Realistic Generation*

A key property of any deception element is that it should be able to deceive an attacker into believing that it is real. Therefore, we now look at different approaches that have been proposed in the literature to create plausible and realistic deception elements. We divide again this section into four parts, based on the layer where deception is applied.

5.4.3.1 *Network Layer*

A common approach to generate fake network activity is to load and replay real network traffic, after replacing confidential information with decoy data. For example, Bowen et al. [28] suggested to simulate decoy network communication by recording real network traffic, and inject into it decoy information such as fake IP addresses and payloads. Kontaxis et al. [101] also discuss the generation of decoy HTTP traffic. In this case the authors introduce templates to describe protocol messages and their parameters, and then generate random permutations and modification of these parameters in order to create decoy HTTP connections. Shing [167] tuned the parameters of sticky connection to produce more realistic traffic in order to evade the detection proposed by Alt et al. [9]

5.4.3.2 *System Layer*

In general, there are two approaches to reproduce realistic system behaviors, either by *duplication* or by *simulation*. The first approach consists of duplicating a legitimate server [101, 183] or a target application [12, 15] while removing possible sensitive information. The second solution relies instead in simulating the appearance, the exact behaviors, and the overall activity of an entire process and equipment [161].

5.4.3.3 *Application Layer*

Different mechanisms have been proposed in the literature to generate realistic deception at the application layer. We can roughly classify these mechanisms into three categories, based on the goal of deception.

First, deception may consist of purposely modifying the behavior of an application in response to specific requests or attacks. In this scope, Julian [92] proposed to alter the response time of an application to confuse an attacker by adding artificial delays proportionate to the average time the same application would take to process that specific user request.

The second category consists of altering the structure of an application in order to showcase a fake attack surface. For example, to render a decoy server less suspicious, Rowe [156] generated fake, yet realistic file systems using pieces extracted from a real file system. The author added random modifications, such as changes to the folder tree, creation of fake directories, or generation of fake files by combing pieces of real files. In the same category, Whitham [193] describes the generation of honey files by combining statistics (such as file names, sizes, access control attributes and modification dates) from public files accessible on the Internet with statistics of usage behavior for the target system.

The third category includes techniques that consist of creating fake content in order to attract and detect attackers. To generate such realistic honey file content, Bowen et al. [27] instrumented genuine and common financial documents such as invoices and tax forms by adding honey accounts, realistic (but fake) names, addresses and other user familiar information. Whitham et al. [194] proposed a more elaborated approach that consists of using natural language processing to generate realistic honey-text content. The authors first selected a median size file from the target directory to generate the template using language processing tags. They then collected characteristics from the target file system that are used to redistribute the original words. Finally, they placed the honey file into selected places inside the target directory.

5.4.3.4 *Data Layer*

The generation of realistic fake data mostly overlaps with the generation and simulation of artificial data (see for instance [79] for an overview of this area). Given the large scope of these techniques, we limit ourselves in this survey to those methods that have been previously used to generate honey tokens for deception.

In [192] the authors introduced a method to create honey tokens that represent fake US citizens. They first observed the statistic features, values, frequencies and dependencies of representative sam-

ples. Then they generated fake personal information according to previously obtained characteristics. Moreover, to generate decoy passwords, authors in [25] converted real user passwords into a set of semantic rules. For example, the following rule "$l_3 d_1 s_2$" refers to a password that includes a word of three letters, one digits, and two special characters. The authors further used a dictionary to generate similar passwords that match the previously obtained rules. Another approach by Juels and Rivest [90] offer either to randomly change the last few characters, digits, or to use a probabilistic model that characterize real passwords.

On the other hand, Bercovitch et al. [22] presented a method to automatically generate honey tokens in a database. First, the authors built various rules to characterize the database structure and relationships. Then, they generated fake, but realistic tokens that satisfied the previously-generated rules. Finally, they assigned a confidence score and ranked the honey-tokens based on their similarity with the real database content. Alternatively, Alese et al. [5] offer to simply shuffle the records of a real database in order to generate fake honey tokens.

5.4.3.5 *Summary*

As discussed above, the realistic generation of honey-tokens is relatively well studied for few traditional domains – such as network traffic, files and file systems, passwords, and database environments. However, the application of deception techniques has recently broadened in scope to cover many other areas, such as web applications and cloud images, where a proper generation strategy has not yet been discussed.

5.4.4 *Monitoring*

A successful deployment of deception techniques drives attackers to interact with them, thus revealing their malicious activities. To collect and analyze such activities, a monitoring mechanism is indispensable. We discuss in this section whether and how the deception is monitored, and whether the deception can be maintained and updated over the time.

Monitoring is more relevant to deception techniques that are used to enhance attack detection rather than for those aiming at prevention and mitigation. We group existing work that have clearly defined their monitoring approach into two categories, based on whether the system that raises the alerts is integrated into the technique itself or is instead implemented as a separate component. Also note that there are two distinguishable phases of attack detection, namely the triggering and monitoring of the alerts.

5.4.4.1 *Integrated Alert*

In this category, Bowen et al. [27] proposed to insert a uniquely identifiable token hidden in the decoy document, which will be sent silently toward a remote server to trigger an alert. Two examples have been presented: a remote image embedded in MS Word document, and a snippet of JavaScript code included in a PDF file [136].

Another example is the use of decoy URLs or hyperlinks that when they are included in a web page automatically fetch some content from a backend server. The use of such links always require a web server that these links point to, which can be the same as the legitimate server if decoy URLs are integrated within the target web application [30, 65]. Otherwise, a separate web server is necessary [121] to handle the deception elements. In any case, the monitoring is usually performed off-line by parsing the access log of the web server.

Finally, a software trap [50] triggers an alert when is exploited by an attacker. It also needs an extra handler to detect and respond to the attackers. However, no details of the way that the handler monitor the software trap was provided by the authors. Likewise, deception techniques such as honey URL parameters can be implemented in the source code together with the monitoring mechanism [147]. Authors proposed to use centralized log management systems to identify and prevent attackers.

5.4.4.2 *Separated Alert*

The first version of honey files proposed by Yuill et al. [198] relied on a centralized monitoring approach. Honey files were deployed on a file server where all file access operations werer monitored. In case of an access to a honey file, an alert is sent from the file server. In the same vein, Wang et al. [189] implemented a system service on Microsoft Windows operating system that registers and monitors honey files and triggers an alert when an access to a honey file is detected.

Bowen et al. [27] proposed another type of honey files that contain honey accounts from prevalent web services, in particular Gmail. The authors then adopted a set of custom scripts to monitor and gather information about the account activity to detect attackers. A similar solution of including honey accounts in honey files also been reused in other works [107, 117]. Honey accounts usually requires a dedicated module to monitor and detect attackers that use them. Chakravarty et al. [39] created manually honey accounts on their decoy servers where they monitored the unauthorized accesses. Honey encryption [89] and honey passwords [90] enforced the generation of honey passwords at the moment of user account creation. In order to detect the misuse of generated fake passwords, they require online

monitoring at the server side. Finally, honey permissions [93] also require an extra module to monitor their use.

Lastly, to detect the selection of a particular honey token in a database, Cenys et al. [38] also suggested the use of a specific handler module.

SUMMARY Independently from the way the actual alert is triggered, most deception techniques require an extra module or even a dedicated server to monitor and detect attacks. However, there is not yet a comprehensive monitoring system that incorporates all the different deception techniques to build a multi-layer proactive deception-based threat detection system. Moreover, it's still unclear for the users how they should integrate such system with respect to traditional defense mechanisms, and in particular with other existing monitoring systems.

Finally, a very important aspect to consider is the update and re-deployment of deception elements. After a deception technique has been deployed, if its not constantly modified attackers can learn its nature and location and simply avoid it in their future attempts to break into a system. This sets intrusion deception apart from other detection mechanisms, where the knowledge of the defense solution does not necessarily undermine its effectiveness. Despite its fundamental importance and the fact that the re-deployment problem has already been mentioned by other studies [97, 189], this aspect has been almost completely ignored by the research community. In fact, if we know well how to update signature-based solutions or re-train model-based approaches, we know almost nothing on when, how, and how often a set of deceptive elements should be replaced with new ones. We do not even know what is the actual impact, in terms of reduced effectiveness, of not updating a deception deployment for long periods of time.

To the best of our knowledge, only one work exists in this space, in which Whitham [193] implemented a form of continuous management in the case of honey files. The system was designed to randomly retire honey files, update their timestamp attributes, and create new honey files to maintain a desired ratio in the system.

5.5 MEASUREMENT & EVALUATION

This section covers the different techniques and existing experiments that have been used to evaluate the efficiency and coverage of intrusion deception. In as early as 2006, Cohen discussed the existing experiments that evaluate deception, and came to the conclusion that most of them do not cover all the relevant aspects [44]. In particular, the author stressed the fact that more elaborated experiments were required to better understand the issues behind the application of deception for cyber-defense. In this section, we review the previous

experiments that offer to measure and evaluate deception techniques, with an emphasis on recent contributions that were not covered by Cohen's book [44].

Note that during our effort to summarize existing experiments, we observed different objectives for the evaluation process itself, depending on the design criteria and the main properties the authors wanted to achieve by using deception. For example, Yuill et al. discussed properties such as the *"plausibility, receivability, verifiability, efficiency, and implementation"* [199]. Bowen et al. further introduced a new set of properties for deception, including *"believability, enticement, conspicuousness, detectability, variability, non-interference and differentiability"* [27]. In 2014, Juels et al. [91] added two additional properties, namely the *"indistinguishability"* and *"secrecy"* of deception techniques, and suggested to use them to tune the decisions of an access control system against potential attackers.

While all these properties play an important role to achieve an efficient deception, most of them are difficult to formalize and measure, which makes their evaluation very challenging. We discuss in this section how previous work managed to evaluate these properties, and the main lessons that we learned from those experiments.

The rest of the section is organized into four distinct categories, that respectively cover the following aspects: 1) the way previous works evaluated deception placement strategies, 2) the way they evaluated the plausibility and realism of deception, 3) the way they measured the efficiency and effectiveness of deception, and finally 4) the way they evaluated the accuracy of deception and its false positives rate.

5.5.1 *Evaluation of Deception Placement*

We discuss in this section the way how previous work managed to evaluate the proposed deception placement strategies. We structure those evaluations depending on whether deception was used to enhance intrusion prevention, detection, or mitigation.

5.5.1.1 *Attack Prevention*

To the best of our knowledge, their is not yet a structured approach in the literature to evaluate the deception placement strategies when it comes to intrusion prevention. Previous work mainly evaluated the additional effort introduced by the presence of deception, and the attacker's extra effort to compromise the target system [25]. Alternatively, other contributions suggested to evaluate the indistinguishability property for a deception technique. It consists of evaluating the level at which a decoy deception technique may confuse potential attackers, and the way it may be discerned from other genuine assets or services to be protected [161]. Nonetheless, previous experiments were mainly aimed at evaluating the intrinsic properties of deception,

but not the different placement strategies and the way they can be leveraged in order to achieve an optimal attack prevention.

5.5.1.2 *Attack Detection*

We observed two alternative approaches in the literature to evaluate the effect of deception techniques and their impact on attack detection.

On the one hand, Garvilis et al. [65] introduced a theoretical approach to embed decoy hyperlinks in a web site in order to detect Denial of Service (DoS) attacks. To evaluate the placement strategy for those hyperlinks, the authors introduced a probabilistic method that leverage a graph-based representation of the target web site. The nodes in the graph capture the individual web pages, and the edges capture the hyperlinks between the pages. Attackers are treated as random walkers in the graph, and the evaluation model computes the probability to detect an attacker with respect to the number and placement of decoy hyperlinks in the graph. Using their theoretical evaluation model, the authors were also able to compare their approach to other decoy placement strategies, such as for example the one based on genetic algorithms.

As opposed to this theoretical approach, other contributions in the literature experimentally evaluated the effect of deception placement strategies, mostly using groups of students or volunteers. For instance, Ben Salem and Stolfo [21] set up an experiment that involved 13 computer science students divided into 4 groups. The authors deployed a system that includes honey files, and run the system during one week to observe how students would interact with the system and whether they will be able to unveil the decoy files. They observed that the location of a honey file in the system drastically impacts the number of users who will be tricked into accessing and opening the decoy document. Similarly, Voris et al. [186] evaluated their deception tool by operating an experiment where decoy files were automatically placed on a benign user system, and a few volunteer users were asked to attack the system. They found that their automated decoy placement achieved similar results compared to a manually designed placement.

5.5.1.3 *Attack Mitigation*

The placement of deception techniques when it comes to attack mitigation is tightly coupled to the detection system that triggers the mitigation, such as anomaly detection [12]. Therefore, previous work has mostly focused on evaluating the efficiency of detection. So far, none has evaluated the efficiency of the deployed mitigation actions, mainly relying on expert knowledge for validation.

Reference	Experiment subject	Metric
[65]	[none]	detection probability
[21]	52 students	detection number
[186]	Students	comparison with manual placement

Table 5.2 – Evaluation of deception placement for attack detection

Reference	Layer	Experiment subject	Metric
[28]	Network	15 students	accuracy
[167]	Network	tarpit detector	rate of detection
[179]	Network	traceroute	response of scan
[161]	Network	students	signal detection theory
[101]	System	[none]	control flow graph similarity
[156]	System	[none]	statistic similarity
[92]	Application	4 students, 4 colleagues	evaluator reaction
[194]	Application	[none]	file content similarity
[192]	Data	100 testers	number of detection
[22]	Data	109 students and researchers	honeytokens quality

Table 5.3 – Evaluation of deception generation

5.5.1.4 *Summary*

Sadly, we are aware of only three controlled experiments that were dedicated to the measurement of decoy placement, as illustrated in Table 5.2. Nevertheless, many studies evaluated implicitly this property while they were evaluating the effectiveness of deception, as discussed in section 5.5.3.

5.5.2 *Evaluation of Deception Generation*

The part mostly pertains to the intrinsic and inner features of deception. Its evaluation relates to the human perception of the generated deception techniques, and whether they are discernible from other genuine assets of the target system. Table 5.3 lists the existing approaches in the literature that performed an evaluation of the deception generation approach, including the metric used for the evaluation and the individual subjects involved in the experiment. Note that most experiments have been performed in controlled and instrumented environments, involving mostly volunteer human subjects. This section discusses the different evaluation strategies and the obtained results.

5.5.2.1 *Network Layer*

We observed in this layer two experimental approaches to evaluate the deception generation mechanism, including an automated, tool-based approach and a manual, human-based approach.

In the first category, Trassare [179] proposed a network-based approach that deceives malicious traceroute attempts in order to conceal the topology and assets of a physical data communications network. To evaluate the generated deceptive topology, the author conducted experiments using the traceroute tool against different network topologies. The results of the experiments showed that the generated deceptive topology drastically limits the ability of traceroute probes to scan the network. Similarly, Shing [167] measured the efficiency of deceptive network tarpits against an automated detector [9]. The author found that only 2.8% of the tarpit hosts had been identified as decoys by the detector.

The second category covers human-based evaluation attempts. In this area, Bowen et al. [28] evaluated the believability of the generated decoy traffic by using 15 human subjects, including students in network security, researchers and security professionals. The participants were asked to manually discern the decoy traffic from the real one. During the experiments, even the most skilled participants successfully discerned in average 49.9% of the decoy traffic, which is equivalent to the probability of guessing randomly. Alternatively, Rrushi introduced in [161] a network-based deceptive approach that is aimed at protecting industrial control systems. By generating decoy network traffic, the proposed solution enables a defender to intervene in the process of an attacker to select a target, and thus steer it towards simulated fake devices. To evaluate the solution, the author asked students to try to discern genuine assets from other decoy assets in the target deceptive system using signal detection theory. The results of those experiments showed that decoy traffic drastically increased the uncertainty of students when picking and attacking their targets.

5.5.2.2 *System Layer*

To measure the similarity between a decoy and a genuine file system, Rowe [156] used 36 statistical metrics such as the mean, standard deviation, minimum and maximum values in order to characterize the structure and type of filenames, file types, size, the distribution of date, and the directory shape. The author found that the generated fake files remain statistically discernible from other real files in the file system, but that the difference is small enough that it remains difficult for individual users to discern decoy and benign files. Kontaxis et al. introduced in [101] a deceptive system that uses fake computation activities to prevent attackers from accessing unauthorized confidential information in the cloud. To evaluate the efficiency of the generated scheme, the authors proposed a binary instrumentation tool that compared the similarity between the control flow graphs for both replicated decoy servers and benign servers. In their evaluation,

the authors have shown that it is difficult to notice the difference between legitimate and generated activities. Adversaries would have either to take the risk to trigger an alert, or investigate significantly to identify data that they are interested in, which may reveal themselves.

5.5.2.3 *Application Layer*

In [194], Whitham et al. proposed an automated tool to process the content of benign documents in order to generate fake documents with the same structure and semantic. To evaluate their approach, the authors introduced a similarity score that leverages the percentage of common words among a decoy and genuine documents. The similarity is expected to be the highest between the decoy document and its originating genuine document, and the lowest between the decoy document and other documents in the file system.

In a different use case, Julian [92] conducted an experiment where users were asked to perform normal and harmful queries against a web application that add deceptive delays. The author measured the reaction of test subjects to determine whether the subjects were able to discern deceptive from other regular processing delays for the application. During the experiment, all users had been misled by the delaying tactic.

5.5.2.4 *Data Layer*

White conducted in [192] two sets of experiments that involve 100 participants. These participants were asked to discern real data from other automatically generated decoy content. In the first experiment, participants were unaware of the existence of honey tokens. When they were given a list of 20 genuine data, they believed that in average 3.67 of them were indeed decoys. Afterwards, participants correctly found in average almost the same number (3.36) of fake data within 3 lists of data in which generated decoys have been inserted. A second experiment revealed that the participants failed to correctly select the honey tokens, even when they were aware that the samples did include for sure decoy elements. The correctly selected number of honey tokens was similar to what would have been obtained if the participants had selected randomly.

Similarly, Bercovitch et al. [22] evaluated the quality of generated honey tokens by showing both genuine data and generated examples to 105 individual subjects. The authors used the results of this study to tune their likelihood-based rating and to select undetectable honey tokens.

5.5.2.5 *Summary*

The most commonly used type of evaluation to measure the realism of deception consists of using a human evaluator to judge the

Type	Reference	Experiment subject	Metric
Controlled	[47]	27 students	duration of attack
	[45]	7 students	ability to control attack path
	[198]	3 students	number of detection
	[27]	20 users	ability to detect attackers
	[30]	3 web bots	bot detection accuracy
	[21]	40 students	number of detection
	[164]	173 students	detection rate
	[76]	4 red/blue teams	user reaction
	[16]	[none]	practicality
Real-world	[191]	spammer	spammer detection
	[174]	spam bot	spammer detection
	[117]	potential attackers	privacy violation detection
	[28]	snoopers at Defcon'09	snooper detection
	[39]	malicious Tor exit nodes	eavesdropping detection
	[26]	potential attackers	attack reduction
	[136]	potential attackers	privacy violation detection
	[108]	potential attackers	detection of malicious activities
	[138]	potential attackers	detection of malicious behaviors
	[29]	malware	malware detection
	[4]	malware	attack detection
	[121]	phishing site	phisher detection

Table 5.4 – Evaluation of deception effectiveness

generated deception, which is similar to a Turing test where a human is used to judge the behavior of a machine. Generally, previous work managed to evaluate and produce realistic deception, but in specific application domain such as network, files, and database. Equivalent experiments are still missing for other domains, including web applications.

5.5.3 *Evaluation of Deception Effectiveness*

The effectiveness of a given deception technique refers to its ability to achieve the desired functionality. Different methodologies have been used in the literature in order to evaluate the effectiveness of deception. We classify them into two main categories, as illustrated in table 5.4. The first category includes evaluations that were conducted in a controlled environment, typically by involving few participants. The second category includes instead evaluations where deception techniques have been publicly exposed to the Internet and evaluated against real users and attackers.

5.5.3.1 *Evaluation in Confined Environments*

Different experimental setups have been introduced in the literature to evaluate the effectiveness of deception in instrumented and

confined environments. These experiments have also adopted different strategies for evaluation, depending on whether deception applies at the network, system, application, or the data layers.

NETWORK LAYER Cohen et al. [47] measured the effects of network-based deceptive defenses by conducting experiments over simulated attack graphs. Participants to these experiments included students and few security experts (both security professionals and researchers). The participants were divided into two groups, including those who were not aware of existing deception techniques, and those who have been informed about the existence of deception. The obtained results have shown that network-based deception techniques were indeed effective as attackers spent more time trying to go thorough deceptive paths rather than the real attack paths. The authors were also able to drive some conclusions on the cognitive confusing factors related to deception, through analyzing the forms filled by the participants during the experiment.

In [45], Cohen et al. extended the previous experiments by introducing a more generic attack graph model designed to drive attackers towards fake targets. The experiments involved seven students who were asked to attack and try to compromise the system. The results were promising, with students constantly misguided and driven through fake attack paths that were introduced for this purpose.

SYSTEM LAYER Heckman et al. [76] organized a cyber-wargame within an instrumented environment. The game involved two distinct teams. A so-called blue team was tasked to set-up a command and control system and try to protect the system against attacks from a second red team. The blue team experimented multiple deception techniques to mislead the adversaries. The log analysis following the experiments resulted in very promising observations. In particular, the adopted deception techniques had a significant impact on the red team operation, as attackers spent a long time trying to compromise fake targets.

APPLICATION LAYER Araujo et al. [16] offered to mitigate known vulnerabilities by implementing the concept of honey patches. In this scope, the effectiveness of their solution largely hinges to its applicability to a larger number of known existing vulnerabilities. The authors evaluated their approach using an experimental environment that included an Apache HTTP server and a simulated web application. They evaluated the effectiveness of their approach according to the number of real application vulnerabilities that they were able to transform into decoy vulnerability through the concept of honey patch. In particular, the authors collected a total number of 75 vulnerabilities that affect their configuration, and that were reported be-

tween years 2005 and 2013. Overall, they found that 49 out of the 75 analyzed vulnerabilities (almost 65%) were indeed convertible into honey patches.

DATA LAYER As described in Section 5.2, state of the art data layer deception techniques mostly consisted of generating and placing decoy user accounts or content. The evaluation of these techniques in instrumented environments mainly involved human participants who were asked to analyze and tell apart decoy from other real authentic accounts and data. In this scope, Yuill et al. [198] tested their honey file system by deploying it on a honeynet, and then asking a group of students to test and try to compromise the system. During the experiment, a honey file was considered as effective when it contributed to detecting at least one attacker, and that the attacker did not realize the fake nature of the honey file before an alert had been triggered. Overall, the authors found that the most effective honey files were those placed closer to the root directory of the file system.

In [27], Bowen et al. evaluated their deception system by integrating it into a honeynet of several virtual machines. The authors posted multiple message invitations with dedicated accounts to their platform in order to encourage volunteer participants to connect and test their deception system. After one week of observation, the authors were able to collect 20 unique users. Five out of these participants triggered at least one alert associated with a decoy document.

In [21], Ben Salem and Stolfo designed an experiment to evaluate the enticing and conspicuous nature of decoy documents. The authors asked a group of students to access an unlocked computer system, looking for financial documents. Several decoy documents were also placed in the system in order to evaluate the reaction and attitude of users when they discovered such documents. The results of the study show that the use of decoys was very efficient in detecting malicious access to the system. In particular, all attackers were detected in the first ten minutes after they had connected to the target computers.

Finally, Shabtai et al. [164] organized an experiment that involved 173 distinct participants. The experiment consisted of generating 50 loan requests, some including deceptive honey tokens. Participants, acting as individual bank employees, could choose to behave in a honest way by validating the loan request and obtaining a 10% commission. Alternatively, they could have acted in a malicious way by suggesting a private funding program from a competitor, and thus obtaining a 20% illegal commission. The participants were divided into two distinct groups, one informed about existing honey tokens in the loan requests and the other unaware of this fact. The results of the experiment showed that all malicious participants could be detected by planting honey tokens in 20% of the loan requests.

5.5.3.2 *Real-World Evaluations*

In open evaluations conducted on the Internet the administrator does not fully control the users who interact with the deception system. Therefore, these experimental setups usually enable to collect and analyze a wider set of interactions with the system. Nonetheless, they often lack the appropriate ground truth about the number and nature of each attack, as well as information about user incentives and real intentions when connecting to the system.

In this category, Borders et al. [26] evaluated the effectiveness of decoy IP addresses in misleading external attackers. To do so, the authors have set up an experimental testbed including an OpenFire gateway that controlled the traffic from the Internet towards three distinct workstations. Three experiments were conducted over a period of 21 days, involving both a normal and a deceptive OpenFire configurations. The normal configuration included default firewall rules that drop access to unauthorized ports and protocols. The deceptive configuration included also 36 unallocated IP addresses towards which the gateway would accept connections in an effort to mislead the attackers. An intrusion detection system was also configured to notify the administrator in case of successful attacks towards the three workstations. After the experiment, the number of successful attacks when using the deceptive configuration was reduced by almost 65% compared with the number of attacks that where reported when using the normal configuration.

To evaluate the effectiveness of honey files, Nikiforakis et al. [136] uploaded such decoy documents to 100 public file hosting services. These files were designed to call back a dedicated server so that the authors will be informed when someone downloaded the decoy file. Over a period of one month, 80 unique IP addresses accessed the honey files and triggered the notification to the remote server. More interestingly, the decoy files also included decoy credentials that enabled the remote users to connect to a fake web application that was implemented for the purpose of the experiment. The authors were able to observe that miscreants from 43 distinct IP addresses have successfully logged in 93 times with the fake account information leaked in the decoy document. Similarly, Liu et al [117] posted five honey files containing decoy accounts on a public file sharing network. Over a period of one month, 192 distinct users downloaded the honey files, including also 45 users who used the decoy accounts in a deliberate attempt to conduct identity theft attacks. Finally, Lazarov et al. [108] constructed five fake Google spreadsheets with decoy banking information and hidden links. Over a period of 72 days, they found that the decoy document have been accessed 165 times and modified 28 times. Moreover, there were 174 clicks on the hidden links inside the same document.

In a rather different attempt to detect spammers, Webb et al. [191] created 51 honey profiles on MySpace, and used these profiles to monitor and detect scam accounts. The authors were then able to collect 1,570 distinct friend requests over a period of four months. Similarly, Stringhini et al. [174] produced 300 honey profiles on three distinct social networks. During their one year long experiment, the authors were able to detect 173 distinct spammers on Facebook, 8 spammers on MySpace, and 361 spammers on Twitter.

Finally, the evaluation of honey accounts has been mostly performed through open deployment and advertisement on popular and publicly accessible Internet services. For example, during one week, McRae and Vaughn [121] submitted 11 honey accounts which contained decoy URL to phishing sites to track down the phishers while they viewed the honey accounts. However, only two out of 11 worked successfully. Similar experiments with honey accounts have been performed on the wireless network at the Defcon 09 hacking conference [28], on malware executables [4, 29], on the Tor network [39], and also on underground forum and online paste tool [138]. All of them have been able to detect a variety of attackers, which empirically shows the effectiveness of honey account at detecting attackers.

5.5.3.3 *Summary*

Considering the current application domain, deception is found to be effective to delay and detect attackers both in controlled and real-world environment. Future deployment of deception in other domain still requires similar evaluation.

More importantly, only two studies has evaluated the false negative rate [21, 164] (testing honey tokens and honey documents respectively), which is fundamental to compare deception solutions with more traditional intrusion detection approaches. In real-world evaluations, the false negative rate is rarely measurable due to the lack of ground truth information about the number of attackers. In contrast, such evaluation is feasible in a controlled environment. Therefore, we believe future effort in this space should shift their focus on assessing the false negative rate.

Reference	Experiment subject	Metric
[65]	students and faculty members	number of false alerts
[30]	human subjects	number of false alerts
[21]	57 students	number of false alerts
[186]	27 users	interference with normal activities

Table 5.5 – False positive evaluation of deception techniques

5.5.4 *False Alarms Evaluations*

Compared to the previous metrics discussed in this section, the evaluation of the false positives rate when using deception in cyber defense has attracted much less interest among the security research community. In particular, we are unaware of any structured approach and methodology to evaluate the false positive rate when deception is being used for intrusion prevention and mitigation.

False positive measurements have been conducted in only a handful of publications that proposed deceptive techniques for intrusion detection.

For example, Garvilis et al. [65] conducted a real-world experiment where they exposed decoy links on the public web site of their university. The authors manually examined the web server logs to determine whether interactions with the decoy links were benign requests that would have resulted in false alarms. The experiment run over a period of one month, during which the authors collected a total of 45,121 distinct requests – only 19 of which were marked as benign (corresponding to a 0.04% false positive rate). The remaining hits were originated from various bots. Nonetheless, a main limitation of this approach is the lack of ground truth about the origin and real nature of false positives. In fact, the authors were unable to verify that the triggered links were indeed benign, and not the result of malicious users who accidentally interacted with the server.

Following a similar approach, Brewer et al. [30] introduced an experimental testbed that simulated a real web site. Multiple decoy links were also introduced to the web site in order to detect potential attackers. The authors further asked multiple participants to navigate through the web site in order to mimic benign user interactions with the server. All requests towards decoy links during this experiment were then considered false positives since the organizers were not expecting the participants to attack the system. In this case, no user have triggered the decoy links during the experiment, which led to a 0% false positives rate.

Similarly, Ben Salem and Stolfo [21] evaluated the false positive generated while using decoy documents to detect attackers. Authors grouped 52 student into 4 groups installing respectively 10, 20, 30 or 40 decoy documents on their file system. Whenever a benign user opens the decoy document, an alert is generated. The number of alerts detected starting from one hour after the students have placed the decoy were respectively 2, 6, 9 and 24. Therefore, the authors concluded that the false positive rate increases with more decoy files. Finally, Voris et al. [186] also measured the false positive rate of decoy documents. In their experiment, 27 normal users were asked to install 40 generated decoy documents inside their file system. The authors then collected more than 318 hours of file access logs across

all participants, finding that legitimate users accidentally touched decoys less than 7 times over a 8 hour workday. Thus, they suggested a simple threshold should be used to differentiate between attackers and legitimate users.

5.5.5 *Summary*

This section has reviewed different aspects of how to conduct experiments to rigorously evaluate deception techniques. Previous research have analyzed some of these aspects, such as the assessment of the quality of newly generated deception elements, better than others. Overall, results seems to suggest that intrusion deception is an effective solution that can complement other defense approaches. However, three points in particular still need more measurement experiments: 1) the assessment of optimal way to manually or automatically place deception elements in a target system, 2) the evaluation of the false negative rate to understand how many attack are performed without triggering any deception element, 3) the evaluation of the false positive rate, in particular in relationship with different placement approaches, 4) the evaluation of how detection degrades over time if deception elements are not constantly replaced with new ones.

5.6 CONCLUSIONS

In this Chapter, we first present a four dimensional classification of existing deception techniques. Note that our focus was not to discuss all deception techniques presented to date, but mainly to identify the different approaches that can be used to reinforce or substitute current intrusion detection and protection solutions. Our work presents an comprehensive analysis of previous studies, addressing a number of key aspects including the theoretical models that had been proposed for deception techniques in computer security as well as the generation, placement, deployment, and monitoring of deception elements. Finally, we examined previous measurements and evaluations of the effectiveness of deception techniques.

During our study, we found that the use of deception and the type of elements that can be used to deceive an attacker are well covered in dozens of publications. However, we identified several shortcomings involving different aspects of deception techniques. In particular, there is not yet a clear methodology to test and measure the placement, the accuracy, the false negative rate, and the false positive rate of such techniques. Therefore, we believe future research should focus designing and conducting real-world experiments to measure the effectiveness of the proposed solutions.

In the next Chapter, we will present some preliminary experiments we performed to answer some of those questions.

6

EVALUATION OF DECEPTION-BASED WEB ATTACKS DETECTION

During our survey of previous work on deception techniques we learned that, despite the fact researchers have proposed several approaches so far, it is still unclear how well these solution work in practice. Evaluation have been purely qualitative, and measure of detection and false positive rates are still missing. In this final Chapter, we try to address these limitations by performing two real-world experiments on the effectiveness of using deception techniques to detect attacks against web applications.

It has been estimated that there are over one billion websites on the World Wide Web today [81], and this number is steadily increasing over time. In 2015 alone, the global business-to-consumer e-commerce turnover grew by about 20 percent, reaching a value of 2.2 billion dollars [57]. Even governments are becoming increasingly dependent on web services to reduce their budget [181].

Unfortunately, this popularity regularly attracts a large number of attackers and according to Symantec [176] three quarters of the websites they scanned in 2015 contained unpatched vulnerabilities.

A large number of techniques have been proposed to secure web applications on the server side. Li et al. [112] classified these techniques into three categories: 1) secure construction of new web applications, 2) security analysis and testing of legacy applications and 3) runtime protection of legacy applications. The first category of techniques usually requires the design of new languages or frameworks, whereas the challenge of security analysis and testing stems from finding the right balance between correctness and completeness. Runtime protection typically provides a scalable solution to secure legacy applications at the cost of certain performance overhead.

These traditional measures, although being essential in any modern security arsenal, cannot provide a comprehensive solution against Internet threats. Due to these limitations, complementary solutions have been recently investigated to help anticipating threats and possibly warn users against attacks in their very early stages. As we already summarized in the previous Chapter, deception techniques have attracted a lot of interest among the security research community [8, 13].

In Chapter 5 we pointed out as one of the main gap in the current understanding of deception techniques is the lack of precise measurements of the detection accuracy, the false positive rate, and to de-

cide whether these techniques alone are sufficient to detect attackers. Taking these issues into account, in this Chapter we present some preliminary experiments we conducted to evaluate these three important aspects.

To perform our tests we first implemented a web deception framework that allows to easily introduce different forms of deceptive elements to any web application. Our solution is deployed as a transparent reverse proxy which intercepts outgoing HTTP connections, injecting deceptive elements based on pattern matching rules and further removing them from the incoming HTTP connections before forwarding the request to its destination. When deceptive element is triggered, it allows to generate an alert or redirect the current HTTP session to another endpoint.

Using our framework, we first focused on ways to measure the false alarms that are generated by a deception-based protection. For this purpose we conducted a second experiment in which we used our framework to protect a production content management system used internally by Orange for a period of seven months – from December 2016 to June 2017. To be able to compare results, the framework was configured to re-use the same techniques we adopted in the red team experiment. During our experiment, the CMS system was used by 258 authenticated users and no false alert were reported by the system. These tests seems to support the hypothesis that deception-based techniques are good candidates for the purpose of attack detection in the web domain.

We then performed a one-day red team experiment hold by Orange Labs where 150 participants were asked to find vulnerabilities within a custom developed e-commerce application. We introduced examples of all known web deception techniques we could find in the literature, and added two new ones we developed specifically for this task. We found that in our experiment setup deception was able to detect 64% of the 25 participants who have successfully exploited at least one vulnerability while missing the remaining 36% of them. Among the 25 participants that have found at least one vulnerability, 14 of them have triggered a trap even *before* finding the vulnerability, which may indicate that deception may be used as an early warning system to detect attackers before they may identify true vulnerabilities.

6.1 METHODOLOGY

Our aim is to design experiments to evaluate the efficiency of deception-based techniques to detect web attacks. To achieve this goal, we first survey existing work to collect a catalogue of deception techniques that had been previously proposed for web attack detection. We then implemented a framework that allowed us to quickly and transpar-

ently insert these elements in an existing web application. At the end of this Section we discuss the strategy that we use to deploy deception techniques, which finally allows us to perform the experiments described in Section 6.2.

6.1.1 *Deceptive Elements*

Most deception techniques for the purpose of web attack detection actually find their root in the concept of honeytoken, which is *"a digital or information system resource whose value lies in the unauthorized use of that resource"* [171]. Early examples of this technique mainly consisted of using honey hyperlinks to detect phishers [121], flash crowd attack [65] and web bots [30]. Most recently, several works promoted the use of deception for web applications [97, 147, 185]. Moreover, the OWASP AppSensor project [141] that aims at detecting and responding to attacks from within the application, provides a detailed descriptions of honey traps that may be used as detection points inside a web application. More precisely, it classifies these detection points into three categories: 1) alteration to honey trap data, which mainly refers to honeytoken in HTTP protocol such as fake hidden form fields, additional URL parameters and cookies, 2) honey trap resource requested, for instance, fake page and directory listed in the application's `robot.txt`file and 3) honey trap data collected and used by the attacker, where a typical example is the use of honey accounts information only visible in source HTML code.

In this dissertation, we design our system using aforementioned deception techniques. Furthermore, we extend them by adding two additional deceptive elements. The first is a fake protected area (e.g., and administration console) that require HTTP authentication. This is similar to a normal fake page, but it contains no content and the fact that it prompts the client for an authentication can catch both humans and automated tools that try to brute-force the password to gain access to the protected resource.

The second additional technique consists of set of fake vulnerabilities that, when triggered, returns realistic error messages based on the pre-defined attack types. The focus of this particular type of deceptive element is to maintain the attackers busy trying to exploit some bug that does not exist.

For our experiment we supported two types of fake vulnerabilities: local file inclusion and SQL injection. For instance, if an attacker alters the honey trap by attempting a SQL injection as illustrated below, the fake vulnerability returns a classic error that exposes a valid-looking vulnerability.

```
Injection: ' or 1=1 --

Response: Invalid query: You have an error in your SQL syntax;
```

```
check the manual that corresponds to your MySQL server version
for the right syntax to use near '' at line 1
```

6.1.2 *Deception Framework*

We now describe the design requirements and the implementation details of our deception framework, which allows a service provider to transparently add different deceptive elements without the need of modifying the target application. While the framework is not the main contribution of our work, it is necessary to quickly test different approaches and conduct experiments on their actual effectiveness on a real-world deployment.

6.1.2.1 *Design Requirements*

To build a deception framework that enables the application of deception techniques to most web application, our system achieves the following design requirements:

— *Language/Framework Independent*
In order to keep our system applicable to a large number of different web applications, our application should not be tied to any programming language nor any particular framework.

— *No Access to Source Code*
One of the main goals of our system is to provide an additional layer of protection without touching or modifying the target web applications.

— *Non-Interference.*
The system needs to support the insertion of any type of deceptive elements, including additional URL or form parameters, fake pages, or honey accounts. However, it is important that these elements do not interfere with the normal behavior of the target application.

To satisfy these requirements, our system was designed to work at the HTTP protocol level, modifying requests and responses on the fly by acting as a reverse proxy in front of the target application. To avoid any potential interference with the original application, the system make sure that any direct sign of deceptive elements or side-effect of their presence is transparently removed from the incoming traffic and does not reach the target application.

6.1.2.2 *Implementation*

Following the above design requirements, we decide to deploy our system in-front of the target web application in the form of a transparent reverse proxy, as illustrated in Figure 6.1. Our framework implements common mechanisms to modify the HTTP protocol and

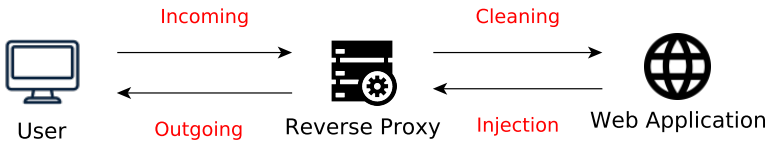


Figure 6.1 – Deception Framework

HTML content such as adding additional cookies, hidden input form fields, additional HTTP GET and POST parameters, fake protected areas, and fake vulnerabilities, which enables the injection of all deceptive elements described in 6.1.1.

The deception framework is configured using simple regular expression rules that specify which request or response needs to be intercepted and modified, the type of deceptive element that need to be inserted, and the fake data associated to that element. For each rule, the reverse proxy is responsible for adding the specified deception technique to outgoing responses, and removing them from incoming requests, before relaying them to the target web application.

The reverse proxy is implemented based on an open source HTTP hacking tool, Hoxy [151]. It enables the interception of HTTP request and responses, observing and altering all aspects of the requests and responses. Coupled with a HTML parsing library, it allows us to modify the HTML content at runtime.

Adding Deceptive Elements While Hoxy intercepts the HTTP response, it exposes the HTTP header and also the body in the form of JSON, string, jQuery or raw buffer. It is quite straightforward to add deceptive elements which are located in the HTTP header. For instance, to add a fake cookie, we simply add a `set-cookie` field with the desired data. Similarly, the fake protected area is implemented by adding a `WWW-Authenticate` field in the HTTP header. To implement a deceptive HTTP GET parameter, we modify the HTTP response status code to 302, which then redirects the original request toward the new URL that has been appended with the fake parameter.

There are also a few types of deceptive elements that require modifications of HTTP body such as the fake hidden input field and fake data in JSON response. Hoxy supports by default the edit of HTML as a DOM object similar to jQuery. We implement hidden input field by search the `form` field in the DOM object and further add the hidden field inside it. It is also possible to convert the HTTP body to JSON object to which we can easily adding fake data.

Finally, in order to implement the fake vulnerability, we detect any modification on the deceptive elements and then apply regular expressions to determine whether the modification belongs to a known attack pattern. For this purpose we reuse the type of attacks defined in the Glastopf Web Honyepot [131]. For our experiments, we sup-

port two of the most popular types of attacks, which are local file inclusion and SQL injection. If the request matches one of the regular expressions associated to those attacks, our system forges and returns a realistic error message to deceive the attackers and make them believe that the system is indeed vulnerable.

Cleaning Deceptive Elements Our framework keeps records of the injected deceptive elements and their locations. For each incoming HTTP request whose URL is known to contain deceptive elements, our system cleans these elements before handling the request to the target web application. In this way, our framework does not interfere with the target application as it receives only original requests.

Reporting and Redirection When the proxy identifies that a user has interacted with one of the deceptive elements, our framework logs the incoming request including the source IP address, the deceptive element, and any user-supplied data that may contain attack information. The framework can also be configured to to redirect incoming requests whose URL matches a regular expression to another application by modifying the back-end server of the reverse proxy. This functionality, for example, allows defenders to redirect attackers from the target application to a similar but protected version of it so as to contain them and better observe their behaviors.

6.1.3 *Deployment Strategy*

The proper placement and integration of various deceptive elements into a target web application is still an open research question. For our purpose, we manually identified the position in which each technique was deployed by following the OWASP testing guide [140], which presents the best practices for web penetration testing. OWASP testing methodology is based on a black-box approach where testers have little information about the target application. In this situation, the testers play exactly the role of an external attacker. Moreover, this guide provides concrete examples of the techniques to use to discover vulnerabilities, which makes it a good starting point to select the placement of deceptive elements right where testers/attackers would most likely look for existing vulnerabilities.

As a result, we combined existing deception techniques with the methodology proposed by OWASP testing guide to devise deception at enticing locations under the perspective of an attacker. For example, at the information gathering stage, we place honey trap resources in `robot.txt` files, as the OWASP testing guide suggests to review them for information leakage. We then designed specific honey accounts disguised as a configuration file of the web application, which relates to the second step of the configuration and deployment management testing. With regarding to the identity, authentication and

session management testing, we deploy at both the login page and password reset page multiple deception techniques such as honey accounts only visible in source HTML code, hidden honey form field and additional session-related cookies. Lastly, we placed additional `HTTP GET` and `POST` parameters both in the web pages and in the client JavaScript code that are subject to the input validation testing. The name of such parameters were carefully selected according to the content of the web application. On top of these parameters, we further deployed the two classes of fake vulnerabilities discussed above.

6.2 EXPERIMENT DESIGN

In this section, we present two experiments designed respectively to measure the number of generated false alarms due to the deployment of deception techniques in a production environment, and evaluate their ability to detect realistic web attacks.

This is a challenging task, as deception should be evaluated against human attackers that manually try to exploit an unknown application. In fact, automated script that target known vulnerabilities would not trigger any detection element, as they are programmed to generate exactly the request required to exploit the target, without visiting or interacting with anything else. For this reason, honeypots are not suitable to perform this test, and collecting data about hundreds of real attackers is a gigantic effort. We solved this problem by splitting the experiment in two parts. First, we deployed our solution on a real application, and monitored its users to detect any possible false alarm raised by the our deception proxy. However, since this deployment cannot provide the data required to test the detection rate, we also performed a second experiment, this time deploying similar deceptive techniques in an application designed for a penetration testing experiment.

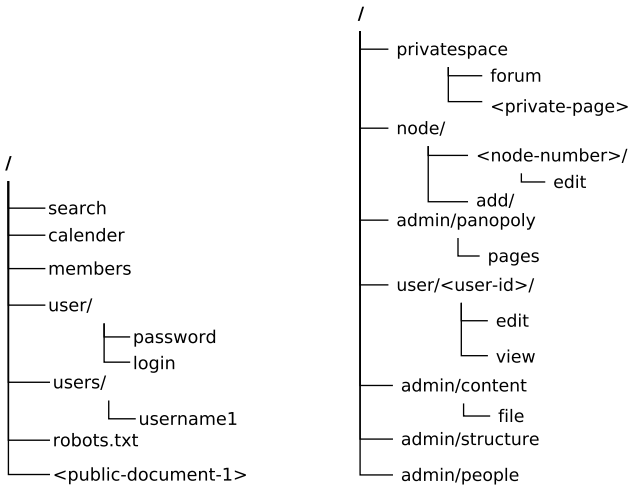
6.2.1 *Use of Deception in a Real Content Management System*

In the first experiment, we implement deception inside an Internet-facing Content Management System (CMS) used in a production environment within Orange, in order to evaluate the false positive rate caused by such technique.

6.2.1.1 *CMS Application*

The web application is based on Open Atrium [1], is an extensible collaboration framework. It provides out-of-the-box functionalities such as dashboard, document sharing, forum, agenda, and user access control. The software was customized to allow research project

1. https://www.drupal.org/project/openatrium



(a) Public space (b) Private space

Figure 6.2 – CMS application tree structure

members to manage a public accessible project websites (open to the public) and a private collaboration space (which requires a user account). Figure 6.2a and 6.2b illustrate respectively the structure of the public and private space. The public space consists of public documents, a search page, and a login page. Once authenticated, a user may view and edit private pages. Administrator can further add/remove users and manage user access.

6.2.1.2 *Deception Placement*

This experiment aims at evaluating the false positive rate of deception techniques in presence of legitimate users. While the two CMSes are different in structure and scope, we placed the deception elements to resemble as close as possible the deployment adopted in the CTF exercise. Table 6.1 and 6.2 present respectively the deception techniques that have been introduced in the public and in the private space.

6.2.2 *Use of Deception in a Capture-The-Flag Competition*

In the second experiment, we integrate deception techniques in a Capture The Flag (CTF) exercise, in which participants are presented with a specific environment where vulnerabilities are purposely planted. By successfully exploiting a vulnerability, the participant finds a flag which allows him to score points. The participant with the highest final score wins the exercise.

While CTF participants are not necessarily a perfect model of real attackers on the Internet, the red team exercise is designed to mimic

Table 6.1 – Deception in public CMS space

URL path	Deception technique	Quantity	FV	Auth
/robots.txt	Honey trap resourse	3		✓
/search/*	Honey POST parameter	1	✓	
	Additional cookie	1	✓	
/usr/login	Honey GET parameter	1	✓	
	Honey account	1		
/usr/password	Hidden input field	1	✓	

Table 6.2 – Deception in private CMS space

URL path	Deception technique	Quantity	FV	Auth
/node/add/*	Honey GET parameter	1	✓	
/node/0/*	Honey trap resource	1		✓
/node/*/edit	Honey GET parameter	1	✓	
/user/*/edit	Honey GET parameter	1	✓	
/user/*/view	Honey GET parameter	1	✓	

*FV: fake vulnerability enabled

*Auth: basic HTTP authentication required

what users would do to discover vulnerabilities in an unknown piece of software. Moreover, using a CTF competition to evaluate the accuracy of deception techniques at detecting attackers has the advantage of providing access to hundreds of "attackers", something that would be very difficult by simply collecting data using a real application.

6.2.2.1 *CTF Environment*

The CTF exercise we used for our test was organized by Orange Labs and aimed at simulating a situation when participants audit the security of an e-commerce application in a black-box approach. The e-commerce application has been purposely developed for the experiment, following the workflow illustrated in Figure 6.3. Each visitor is first presented with a user agreement page that already presents the first challenge. This page contains a obfuscated JavaScript code that requires the user to read at least for one hour the user condition. By successfully hijacking the client side check, the user may visualize the product list and the comments of each product. Furthermore, users can create an account and use it to log in and checkout their orders, and finally perform online payment. In addition, there is also a form that allows authenticated user to post public messages.

Many classic vulnerabilities such as cross site scripting, local file inclusion, SQL inject, and remote code execution have been planted

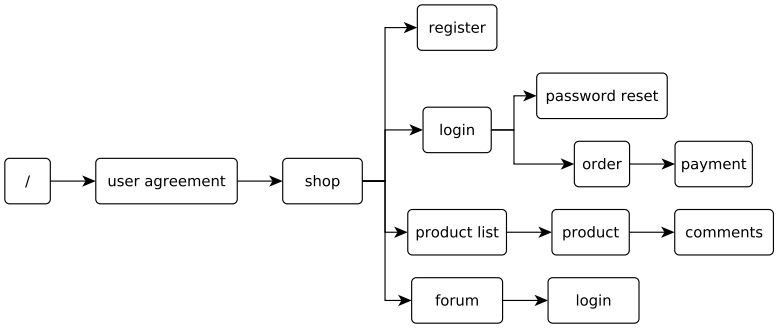


Figure 6.3 – CTF Application Workflow

Table 6.3 – Deception in CTF exercise

Page	Deception technique	Quantity	FV	Auth
User agreement	Honey POST parameter	1	✓	
	Additional cookie	1	✓	
Shop login	Honey POST parameter	1		✓
	Honey account	1		
Forum login	Honey account	1		
Account API	Honey GET parameter	1	✓	
Product comment	Honey POST parameter	1	✓	
/robots.txt	Honey trap resource	3		✓
/web.config	Honey trap resource	1		✓
/web.config	Honey account	1		

*FV: fake vulnerability enabled

*Auth: basic HTTP authentication required

at different locations including the user profile, product comment page, order page and in particular the online payment page. In total, 15 flags have been inserted inside the CTF application.

6.2.2.2 *Deception Placement*

Our main goal is to evaluate the ability of deception techniques to detect web attacks in their early stage. Following the placement strategy described in Section 6.1, we manually inserted a number of deception tokens, as summarized by Table 6.3.

We started by modifying `robots.txt` and `web.config` (a classic configuration file for Windows web server) to insert four honey trap resources and one honey account. We then added additional `HTTP POST` parameter and a fake cookie in the user agreement page to see how deception affects the advancement of participants. We also add fake `HTTP GET` and `POST` parameters and honey accounts in most of the

login-related pages and APIs. Lastly, we placed another fake `HTTP POST` parameter in the product comment page. Note that we did not place any deception on pages such as orders and payment where many vulnerabilities are planted. The main reason is that we wanted to use our deception element to detecting attackers still at the beginning of their interaction with the target application. The second reason is that we were asked by the organizer to minimize the interference between deception techniques and real CTF vulnerabilities, not to discourage the participants.

6.3 RESULTS

In this section we present and discuss the results of the two experiments.

6.3.1 *CMS Experiment*

The experiment on the real CMS application has been performed for a period of seven months from December 2016 to June 2017. In this time frame the application was used to host 13 active projects, each of which had its own dedicated sub-domain name. In total, we observed 258 users that have successfully authenticated and interacted with the application.

Public Space – Four alerts have been triggered from the honey trap resources placed inside the `robots.txt` file. The four accesses originated from the same IP address that, according to the web server access log, was actually performing a scan to test the system for known vulnerabilities. Similar abuse reports have been found from `AbuseIPDB`, which confirms that we have actually detected a malicious IP address.

Private Space – No alerts have been triggered from the deception placed in the private space during our experiment. Therefore, based our experiment setup, deception techniques that have been deployed in the CMS application have generated zero false positive. This confirm that since these techniques are implemented at the protocol or source HTML code level, they remain mostly invisible to normal users.

6.3.2 *CTF Experiment*

The CTF competition has been conducted in a local network for a period of 8 hours in September 2016. In total, the event counted 150 participants (each clearly identifiable by a different IP address), which included a mix of information security students and security professionals.

Overall, only 25 (16.7%) of them successfully discovered at least one flag, while 84 (56%) have triggered at least one of the 12 deception traps – e.g., by trying to tamper with a fake cookie or form parameter, by trying to use a honey credential to log in, or by visiting some of the honey URLs. These results seems to suggest that the deceptive elements were easier to trigger than the real vulnerabilities, which is exactly what a good deployment strategy should achieve. More interesting, 64% of the participants that discovered a real vulnerability also triggered a deception trap, showing that an attacker who spend enough time and resources to discover a real bug is likely to raise an alert due to his interaction with deceptive elements.

6.3.2.1 *Manual vs Automated Actions*

A first interesting question we want to understand is whether our traps were triggered by manual interactions or by automated tools and scanners. To answer this question we inspected the `user agent` strings in the access log of the web server. In the context of a CTF exercise, we believe the user agent is a reliable source of information since there is no incentive for the participants to hide their actions or disguise their activities.

We found that five participants triggered the four honey trap resources (placed in the `robots.txt` and `/web.config` files) using popular scanner tools (Nikto and DirBuster). For the remaining honey traps, the trigger connection originated from popular browsers. We thus consider that those participants have discovered manually the traps. Interestingly, we also found evidence of users who run the `sqlmap` tool to try to exploit the fake SQL injection vulnerability, and others that used the `hydra` tool to brute force the HTTP authentication of a fake URL. This supports the hypothesis that these two new form of deception we introduced in our framework are successful in slowing down attackers by tricking them into focusing their effort on fake parts of the application.

6.3.2.2 *Impacts of Deception Placement*

In order to measure the impacts of deception placed at different locations, we evaluate the number of participants that triggered each trap. Furthermore, we analyze the average number of manual attempts that have been performed on top of the deception element, which reflects the extent to which the participants are enticed by the given deception.

As illustrated in Table 6.4, for each location, we present the number of traps implemented, the number of distinct IP addresses detected and the average number of manual attempts that modified the traps on that location. The user agreement page is the most effective location to detect the participants using deception. This page is actu-

Table 6.4 – Number of distinct IP addresses detected

Page	Traps	Detected Participants	Average N. of Attempts
Configuration files	5	14	8
User agreement	2	56	28
Account API	1	22	6
Product comment	1	16	446
Shop login	2	22	11
Forum login	1	7	2

*Configuration files: /robots.txt and /web.config

ally situated just after the starting point of the application workflow, which seems to suggest that deception placed at the first pages of the workflow is quite effective to detect attackers since among the 84 detected IP addresses, 56 of them have been detected at the user agreement page.

On the other hand, deception at product comment page has received the largest average number of manual attempts per deception element, which indicates that this location is particularly interesting for participants. One possible explanation is that the product comment page accepts user-provided comments, so the participants spent more time playing with its parameters looking for a possible injection vulnerability. This kind of insight is quite useful and should be integrated while devising and deploying deceptions for a web application.

6.3.2.3 *Effectiveness*

In order to shed some light on whether deception techniques are effective in detecting attackers, we analyze the potential relationship between the triggered honey traps and the flags that were found by each participant. For each participant who found at least one flag, we collected the time at which each flag was discovered and each trap (if any) was triggered. Figure 6.4 presents the cumulative rate of honey traps triggered and flags found by the 25 participants. The horizontal axis describes the time during which the CTF has been played. The vertical axis presents the cumulative rate. Thus the red and blue curves illustrate respectively the cumulative rate of triggered honey traps and discovered flags.

Almost 35% of honey traps have been triggered even before any flag could be found. This seems to indicate that honey traps may be used as an early warning system to detect attackers before they may identify true vulnerabilities and weaknesses in the target application. Further investigation shows that 16 of the 25 participants who have

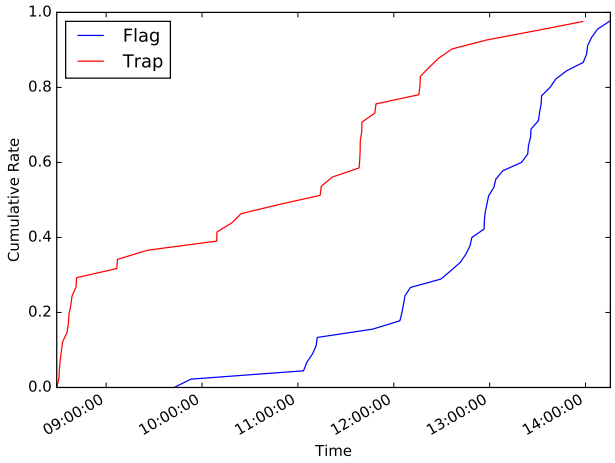


Figure 6.4 – Cumulative distribution of detection and flag

discovered at least one flag have triggered at least one honey trap. Moreover, 14 out of 16 have triggered a trap *before* finding a flag.

By the time 80% of the discovered traps had been triggered, only 30% of flags had been found. This seems to indicate that as long as the attacker advances further in his attack plan (by getting deeper in his attack path and finding new flags), he will be more exposed to additional traps, and quite effectively (80% of traps activated). We can deduce from those figures that the expected efficiency rapidly increases when the attacker advances further in his attack scenario, which may enhance the detection rates. This means that to be more effective, honey traps need to be composed and intertwined within the workflow of an application, and not only as a single layer or on the front page.

6.4 DISCUSSION

In this chapter, we presented the design and implementation of a deception framework for web applications. We used our framework to conduct two experiments to evaluate respectively the effectiveness of deception techniques to detect web attacks and the false positive rate of the same techniques when deployed in a production environment. Even though the two experiments have been conducted in different conditions, we have implemented and deployed similar deception techniques in both of them.

Our test on a real application raised no false alarms over a period of seven months. This seems to confirm one of the main advantage of deception-based defenses: the ability to provide detection with zero false alarms.

The detection accuracy was obtained through a red teaming exercise, which we consider the most appropriate method available to perform such measurement in a controlled environment. While the results depends largely on the profile of the participants, our tests included a considerable number (150) of users with different skills and knowledge of web security. A drawback of using a CTF scenario instead of real attackers is that CTF participants are more aggressive and have no incentives in being stealthy and in reducing their footprint on the target system. A real attacker may be more cautious, thus resulting on a different (probably lower) rate of deception elements triggered and vulnerability discovered. The results of this second experiment are positive but still far from perfect – with 36% of the attackers that were successfully able to find and exploit a vulnerability without interacting with any deceptive elements. We believe that this can be a consequence of the placement strategy we adopted, and maybe a more aggressive deployment would provide a higher detection rate. However, too many deceptive elements can tip of the attackers about the presence of this type of defense, therefore actually reducing the overall efficiency of the technique. More tests are required to better understand this phenomenon and the intricacies of an "optimal" solution to deploy deception on a web application.

7

CONCLUSIONS AND FUTURE WORK

This dissertation discussed several directions a service provider may follow, in addition to merely secure its infrastructure, to provide better security for its customers. A number of techniques have been proposed, from the vantage point of a service provider, to enhance its ability to measure and monitor security threats – including possible abuses, compromised instances, and external attacks.

The thesis first proposed a systematic study to confirm that cloud service provider has been abused by attackers to host part of their malicious infrastructure and better understand the extent of this phenomenon. By conducting a large-scale analysis of about one million malware samples, our results showed that miscreants sustained long-lived operations through the use of public cloud services. Moreover, we observed an increasing trend in the number of malicious and dedicated cloud-based domains from 2008 to 2014. The existing security mechanisms employed by the services providers were actually insufficient to correctly measure and detect this kind of abuses.

The second part of the dissertation explored the valuable information a service provider has access to, that can be used to analyze online attacks – with a focus on the phishing phenomenon that was previously studied by external researchers in a more limited scope. In particular, we presented the design and implementation of PhishEye, a system specifically designed to analyze phishing kits in ethical way. Using this system, we performed a five-month experiment that enabled for the first time the understanding and measurement of the entire life-cycle of this type of attack. We were able to distinguish the victims from the attackers from other third party visitors. Our results showed that most of the victims activity takes place in the first period after the phishing kit is installed, and before the security community discovers its existence.

Finally, we discussed the deception techniques that a service provider may employ in order to add an additional layer of security for its customer applications. We investigated existing deception techniques, and proposed a comprehensive classification that allowed us to identify open research directions – including the design and modeling of deception techniques, their deployment, and the evaluation of such techniques. Furthermore, we presented the design and evaluation of a deception-based approach to protect customer web applications. The results from a red teaming experiment showed that our approach was able to detect 64% of the participants who successfully exploited at least one vulnerability. In a separate long term experiment in a pro-

duction environment, we implemented similar deception techniques in a system that was routinely used by 258 different users. During a period of seven months, zero false alert were raised by the system.

The security from the perspective of a service provider have attracted little attention among security researchers before. The work in this dissertation first revealed the phenomenon that attackers used public cloud services to host their malicious infrastructure and that existing security mechanisms were unable to detect timely this type of abuse. Service providers are always reluctant to conduct research on this direction due to the privacy concern of working with customer data. It would be interesting to work together with the public cloud service providers to fully understand the precise role of such abused services, which would further enable researchers to build prompt detection of this kind of threat. In our study, we only looked at network communication data, but it would be interesting for a service provider to combine the network data with the system information such as CPU usage and performance counters to measure and detect abusing customers.

Recently, a few works have emerged that proposed to measure and understand the role of service providers in different cases of abuses, mostly involved public cloud repositories. Another interesting extension to our work would be to study the role of service provider in other types of attacks – such as distributed denial of services or the distribution of malware. This kind of study would eventually raise awareness to the importance of this phenomenon, and help other service providers to detect related attacks.

The second work in this dissertation showcased the ability of a service provider to monitor phishing attacks for their entire life-cycle, which was previous unknown to external researchers. Our study illustrated the way that an application service provider can implement the security monitoring in an ethical way. While in our work we resorted to a honeypot implementation, it would be interesting for service providers to implement our system in real world production environments.

Furthermore, other studies could be conducted to understand other types of attacks that were otherwise difficult to analyze for external researchers, including the use of exploit kits in the wild. This would allow researchers to gain some insight of the different actors, their behaviors, and the entire life-cycle of an exploit kit.

The last part of the dissertation applied deception techniques to enable services provider to achieve an additional layer of security for its customers web applications. We conducted two experiments where we obtained a reasonable detection rate with no false positives. The results seems to support that service providers are capable to provide better security for their customer applications but more experiments are necessary to study how the deployment of deception

elements affects the attack detection rate. It would also be interesting to apply similar techniques on other services such as FTP, email, and databases. Moreover, there is not yet a consensus among researchers about the way to deploy and evaluate deception techniques when they are used as a defense mechanism. More experiments are actually needed to evaluate such techniques in various application environments.

In conclusion, this dissertation investigated different directions in which a service provider can measure malicious infrastructure, monitor the phishing attacks installed on compromised services, and finally protect customers applications using deception techniques. However, we only look at the tip of the iceberg and we did not fully explore all the valuable information a service provider has access to, which could help investigating other attacks or propose other protection mechanisms. As a results, we believe service providers will play an increasingly important role to provide secure services to both customers and other Internet users.

a

APPENDIX

A.1 RÉSUMÉ

De nos jours, de nombreux services sur Internet sont proie à des cyberattaques qui menacent les fournisseurs de ces services et leurs utilisateurs. De nombreux acteurs peuvent alors rentrer en jeu afin de mieux contrer les nouvelles menaces. Dans cette thèse, nous explorons le rôle que peuvent jouer les hébergeurs de services (fournisseurs Cloud) pour la sécurisation des services sur Internet ; un rôle qui est très marginal dans les modèles de sécurité existants.

Nous explorons plusieurs directions qu'un hébergeur de services peut suivre pour renforcer la sécurité. Plus précisément, nous exploitons les informations dont disposent ces hébergeurs pour mesurer et surveiller les différentes menaces, y compris les abus de logiciels malveillants, les sites d'hameçonnage, et les attaques Web.

L'ampleur de l'utilisation des services cloud par les logiciels malveillants est peu appréciée. Nous présentons une étude systématique et à grande échelle qui montre que les mécanismes de sécurité adoptés par les hébergeurs de services sont insuffisants pour mesurer et détecter ce type d'abus.

Dans la deuxième partie, nous examinons la capacité d'un hébergeur à surveiller les attaques qui sont autrement difficiles à étudier pour des chercheurs tiers. En particulier, nous avons conçu et mis en place PhishEye, un système éthique pour analyser en temps réel les outils d'hameçonnage, ce qui nous a permis pour la première fois de comprendre l'ensemble du cycle de vie des attaques de ce type.

Enfin, nous explorons des techniques alternatives, en particulier les techniques de diversion/leurre, qu'un hébergeur de services pourrait déployer afin d'améliorer la sécurité des services qu'il héberge. Nous menons une étude exhaustive des techniques existantes et nous évaluons leur efficacité lorsqu'elles sont utilisées pour protéger les applications Web.

A.2 INTRODUCTION

L'Internet a rapidement évolué à partir d'un petit réseau régional qui connectait quelques institutions académiques et militaires jusqu'à un très grand réseau mondial offrant une variété de services – tels que les sites web, la messagerie électronique, la téléphonie sur IP et le partage des fichiers. En 2017 [124] il a été estimé que plus de 49% de la population mondiale utilisent Internet pour divers aspects de

la vie, y compris les réseaux sociaux, le commerce électronique et la télécommunication.

Les premiers fournisseurs de services Internet (FAI) sont apparus au début des années 1990, et au départ ils ne pouvaient offrir qu'un accès limité sur Internet et nombre réduit de services. Aujourd'hui, cette offre s'est beaucoup élargie en incluant une multitude de services, par exemple, des services de paiement, du cloud computing et des solutions de stockage en ligne pour les clients allant des grandes entreprises aux utilisateurs individuels. Pour faire face à ces offres diversifiées, dans cette thèse le terme *fournisseurs de services* désigne non seulement les FAI, mais aussi les hébergeurs d'application, de sites Web et les hébergeurs de services de cloud public.

Malheureusement, l'expansion rapide d'Internet et ses nombreux utilisateurs ont également fait appel à des utilisateurs malveillants. Les cybercriminels ciblent régulièrement des services lucratifs (par exemple, les services bancaires en ligne), les sites Web d'organisation cilbe, l'ordinateur personnel et smartphone, la plupart du temps à la recherche d'un gain financier. Pour atteindre cet objectif, les cybercriminels recourent souvent à une combinaison de vulnérabilités `0-day`, logiciels malveillants sophistiqués (malware), attaques par hameçonnage (phishing) et exploits Web. Cela a entraîné une lutte continue avec la communauté de sécurité engagée à développer de nouvelles solutions pour détecter les attaques et protéger les services légitimes. Dans ce contexte, les hébergeurs de services jouent un rôle très important dans la sécurité des services et de leurs utilisateurs, un rôle souvent négligé ou sous-estimé dans les modèles de sécurité existants.

A.2.1 *Modèle de sécurité actuel*

La garantie de sécurité offerte par chaque fournisseur de services peut grandement varier selon le type de service offert, la taille de l'hébergeur et aussi les règlements du pays où réside l'hébergeur. Cette diversité rend presque impossible de comparer la sécurité des différents hébergeurs de services. Cependant, même si parfois le modèle de sécurité n'est pas clairement défini, les hébergeurs de services adhèrent généralement à un *modèle de responsabilité partagée*, proposé par Amazon Web Services (AWS) [10] en 2014. La Figure a.1 illustre une version simplifiée où nous distinguons quatre acteurs principaux : le *client* qui installe un service légitime en utilisant l'infrastructure d'*hébergeur de services*, *l'utilisateur final* de ce service et *l'attaquant* qui peut cibler à la fois l'infrastructure de l'hébergeur de services et le service légitime.

Le modèle de responsabilité partagée définit les mesures de sécurité que l'hébergeur de service et le client peuvent convenir et mettre en œuvre respectivement. L'hébergeur est responsable de la gestion

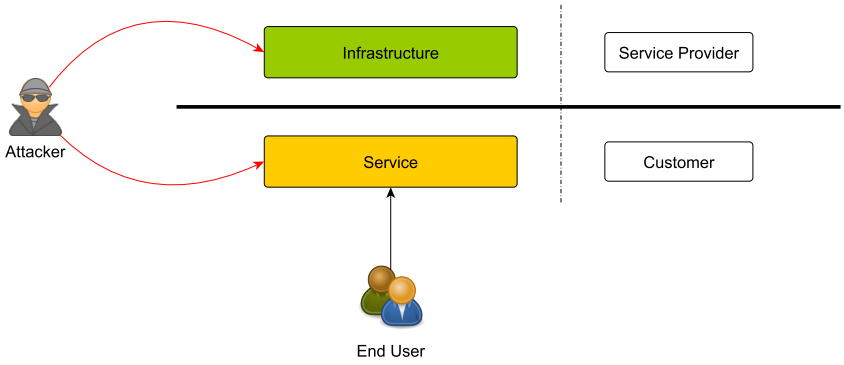


FIGURE A.1 – Modèle de responsabilité partagée

de la sécurité de l'infrastructure avec l'ensemble des logiciels qui prennent en charge le service client. Par exemple, l'hébergeur de services doit protéger l'infrastructure contre les attaques par déni de service distribuées, et corriger les failles logicielles connues et les vulnérabilités. D'autre part, le client doit prendre les mesures nécessaires de sécurité visant à protéger les utilisateurs de l'application, qui sont considérés jusqu'ici comme hors de portée pour l'hébergeur de services.

Bien que simple à comprendre, ce modèle présente également certaines limites. Un exemple simple est le cas où un cybercriminel abuse un service vulnérable pour effectuer des activités malveillantes qui menacent la sécurité d'autres utilisateurs sur Internet. Dans ce scénario, le modèle actuel de responsabilité partagée attribue la responsabilité au client exécutant ce service. Pourtant, l'hébergeur est sans doute aussi responsable parce que son infrastructure est utilisée de manière abusive pour effectuer des activités malveillantes. Ce phénomène a été confirmé par notre étude dans le Chapitre 3. Le modèle de responsabilité partagée délègue la responsabilité de sécurité de l'hébergeur de services au client, ce qui est regrettable parce que l'hébergeur dispose d'un point de vue avantageux pour proposer des mesures de sécurité et pour mieux surveiller les attaques et enfin protéger leur clients.

A.2.2 *Objectif de la thèse*

Actuellement, sous le modèle de responsabilité partagée, un hébergeur de services a peu d'incitation à fournir de manière active une meilleure sécurité pour ses clients. Ceci est confirmé par une étude récente sur les hébergeurs de service web [36], dans lequel les auteurs ont constaté que la plupart des hébergeurs ont échoué à détecter les

signes les plus évidents que les applications de leur clients ont été compromises.

Dans cette thèse, nous voulons explorer les mesures de sécurité qu'un hébergeur de services pourrait proposer, au lieu de se contenter uniquement de la sécurisation de son infrastructure, en profitant de sa position privilégiée entre les utilisateurs finaux et les services, afin de fournir une meilleure sécurité.

Par exemple, un hébergeur de services a accès à des informations précieuses qui sont indisponibles à ses clients, y compris une vue globale sur l'ensemble de l'infrastructure, le trafic réseau, et également un contrôle total de la couche logicielle en dessous de l'application client. Pour montrer que cette information est actuellement sous-utilisée pour développer des mécanismes de sécurité et étudier les nouveaux phénomènes de sécurité , nous avons identifié trois objectifs de recherche distincts :

O1. Les hébergeurs de services concentrent leurs efforts sur la protection de leur propre infrastructure contre les menaces externes. Cependant, On sait peu de choses sur le fait si l'hébergeur de services peut être *abusé* pour des fins malveillantes – c'est-à-dire, dans le cas particulier où le rôle de l'attaquant et du client est joué par le même acteur. Par exemple, le client peut utiliser un service pour attaquer d'autres machines sur Internet, voir même héberger des applications malveillantes ou illicites sur l'infrastructure de l'hébergeur de services. Bien qu'il existe des preuves anecdotiques que les hébergeurs de services, en particulier les hébergeurs de services cloud, sont en fait abusés par les logiciels malveillants, peu d'attention a été accordée à ce phénomène et une étude plus rigoureuse est nécessaire pour mesurer ce phénomène émergent.

O2. Au cours de la dernière décennie, la communauté scientifique a mis un effort considérable pour étudier différentes attaques, telles que les exploits Web ou kits de hameçonnage. Malheureusement, la plupart des études précédentes étaient seulement en mesure de réaliser des expérimentations en recueillant des données publiquement disponibles (par exemple, soit en analysant des infections déjà signalées ou en explorant le Web à la recherche de signes de contenu malveillant). Ceci a considérablement limité la portée de ces études, parce que le comportement d'un service juste après qu'il ne soit compromis, et avant qu'il ne soit découvert par la communauté de la sécurité, est resté largement inconnu. Cependant, les journaux d'infrastructure de l'hébergeur et l'analyse approfondie des connexions entrantes peuvent fournir une meilleure vue sur certains types d'attaques, par rapport à ce qui a été étudiées par des chercheurs externes. En particulier, dans cet objectif, nous nous concentrons sur l'analyse de la durée de vie de kits de hameçonnage (qui sont très

souvent installés pour monétiser des applications web compromises), pour identifier les différences entre les vue de l'hébergeur et celles d'un chercheur tiers.

O3. Pour notre objectif final, nous considérons les hébergeurs de services qui veulent ajouter une couche de sécurité supplémentaire à leurs applications clients, en particulier dans le cas des applications Web. Des solutions de sécurité traditionnelles adaptées à cette exigence, comme la surveillance du réseau et le système de détection d'intrusion, génèrent beaucoup de fausses alarmes ou sont incapables de faire face aux attaques avancées et inconnues auparavant. Par conséquent, nous voulons étudier si les *techniques de diversion/leurre ("deception" en anglais)* seraient un meilleur choix dans ce contexte, et quels sont les défis scientifiques qui empêchent ces solutions d'être déployées à grande échelle.

Le but de cette thèse est de faire progresser l'état de l'art le long de ces trois objectifs différents, en effectuant des mesures et en concevant des systèmes d'analyse et de protection du point de vue d'un hébergeur de services.

A.2.3 *Aperçu de la thèse*

Motivé par les trois objectifs présentés ci-dessus, cette thèse présente un certain nombre de techniques qui tirent parti du point de vue d'un hébergeur de service pour *mesurer les abus* (Objectif **O1.**), *surveiller la compromission* (Objective **O2.**) et *améliorer la protection* (Objective **O3.**) des applications clients (comme illustré dans la Figure a.2).

Pour le premier objectif, nous avons concentré nos efforts sur les hébergeurs de services cloud – car ils sont l'un des principaux hébergeurs de services et leur nature les rend plus enclins aux abus possibles. Les services cloud offrent des avantages supplémentaires tels qu'une plus grande résilience, une protection de l'hyperviseur contre les attaques réseau, la reprise après sinistre à faible coût, le contrôle de sécurité à la demande, détection en temps réel de la falsification du système et reconstitution rapide de services [175]. Beaucoup de ces avantages rendent le cloud particulièrement attrayant pour héberger des services malveillants et les attaquants comptent souvent sur les hébergeurs de cloud de la même manière et pour les mêmes raisons que les clients légitimes [169]. Dans le Chapitre 3, nous analysons le rôle aujourd'hui des services cloud dans un logiciel malveillant, présentant une approche systématique pour mesurer la sécurité de l'hébergeur de services cloud contre les abus de logiciels malveillants. Les résultats de notre étude ont montré que les criminels ont entretenu des activités malveillantes d'une durée prolongée sur le cloud,

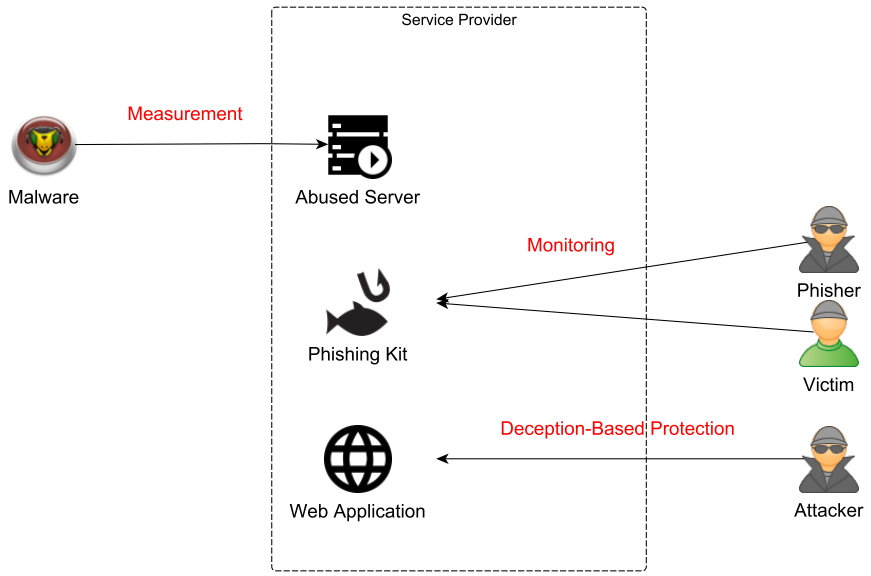


FIGURE a.2 – Aperçu de la thèse

ce qui révèle le besoin d'un nouveau mécanisme de surveillance pour les hébergeurs de services comme conjecturé dans l'objectif **O2.**.

Dans le deuxième objectif, nous examinons la capacité d'un hébergeur à surveiller les attaques qui sont autrement difficiles à comprendre pleinement. Dans ce contexte, nous nous concentrons sur les attaques par hameçonnage qui compromettent les sites Web vulnérables hébergés sur l'infrastructure d'un hébergeur de services pour installer des sites d'hameçonnage. Les attaques par hameçonnage restent une menace majeure pour les fournisseurs de services Internet (FAI), hébergeurs de service cloud, ainsi que les fournisseurs d'email. En effet, le nombre de sites de hameçonnage uniques a atteint un record absolu au deuxième trimestre 2016 [2]. Cependant, des études antérieures menées par des chercheurs utilisant des données externes (c'est-à-dire sans accès aux journaux d'application ou le trafic réseau) ont échoué à surveiller l'ensemble du cycle de vie des attaques par hameçonnage. Le travail dans cette thèse propose une nouvelle approche pour un hébergeur de services d'application pour surveiller les attaques par hameçonnage afin d'avoir un aperçu sur les différents acteurs et leurs comportements, comme décrit dans le Chapitre 4.

Dans notre objectif final, nous explorons des techniques alternatives qu'un hébergeur de services peut adopter pour protéger ses applications clients. Du point de vu d'un hébergeur, la protection au moment d'exécution est la seule solution viable, en raison de son évolutivité et le fait qu'elle ne nécessite pas de toucher ou de modifier les applications client. Les schémas traditionnels de protection d'exécution reposent soit sur un pare-feu applicatif basé sur la si-

gnature ou sur la détection d'anomalie [112]. Les solutions à base de signature sont efficaces mais incapables de détecter des attaques inconnues, alors qu'il est difficile pour les approches basées sur les anomalies d'équilibrer la précision de détection et le taux de faux positif – en particulier dans le domaine du Web. En raison de ces limitations, cette thèse étudie l'utilisation des techniques de diversion/leurre qui peuvent être intégrées inconsciemment au-dessus de l'application client par l'hébergeur de service, promettant un taux de faux positifs extrêmement faible combiné avec un taux de détection élevé contre des attaques connues et inconnues.

Bien que la diversion/leurre ne soit pas un nouveau concept, lors de notre effort pour résumer les travaux existants sur ce sujet, nous avons constaté qu'il manque encore une compréhension globale de ces techniques et de leur application dans la sécurité informatique. En particulier, il n'y a pas encore de consensus parmi les chercheurs sur quels sont les principaux objectifs et les défis techniques à résoudre quand les techniques de diversion/leurre sont adoptées en tant que mécanisme de défense. En particulier, la modélisation, le déploiement et l'évaluation de ces techniques étaient peu ou mal abordées dans la littérature. Pour élucider ce sujet, le Chapitre 5 présente une étude complète sur l'utilisation de techniques de diversion/leurre dans la sécurité informatique.

Enfin, si les schémas de diversion/leurre sont bien étudiés dans certains domaines, ce n'est pas le cas pour les applications web. Les travaux antérieurs dans ce domaine [92] ont seulement étudié l'utilisation de la diversion/leurre pour retarder les attaques. Dans cette thèse, nous explorons l'utilisation de leurre pour détecter de manière active les attaques Web à leurs débuts. C'est un domaine qui attire récemment beaucoup d'attention dans l'industrie, mais il est encore largement inexploré du point de vue de la recherche. Au meilleur de notre connaissance, aucun travail antérieur n'a effectué des expérimentations pour évaluer la précision et le taux de faux positif tout en utilisant la diversion/leurre pour améliorer la détection des attaques Web. Nous présentons dans le Chapitre 6 la conception et la mise en œuvre de deux expérimentations préliminaires sur l'application de la diversion/leurre pour détecter les attaques web.

A.2.4 *Synthèse du manuscrit*

Cette thèse propose un certain nombre de techniques pour mesurer et surveiller le sécurité du point de vue d'un hébergeur de services. Le Chapitre a.3 analyse les travaux précédents liés à ce sujet.

Dans le Chapitre 3 nous proposons une approche systématique pour mesurer le rôle des services cloud dans les logiciels malveillants (Chapitre 3). En particulier, nous étudions la façon comment les cybercriminels abusent les services cloud public pour héberger une partie

de leur infrastructure malveillante. Nous avons effectué une analyse à grande échelle de tous les échantillons de logiciels malveillants soumis au système d'analyse de logiciels malveillants Anubis entre 2008 et 2014. Nos résultats révèlent que les cybercriminels obtiennent des opérations d'une longue durée au travers l'utilisation des ressources de cloud public, en tant que composante redondante ou majeure de l'infrastructure de logiciels malveillants. Nous observons également que le nombre de domaines dédiés hébergés dans le cloud ont augmenté de près de 4 fois entre 2010 et 2013. Pour comprendre les raisons de cette tendance, nous présentons également une analyse à l'aide d'enregistrements DNS publics.

Le Chapitre 4 présente une nouvelle approche tirant parti du point de vue d'un hébergeur de services pour surveiller les attaques et les compromis contre les applications Web, en mettant l'accent sur les kits d'hameçonnage (Chapitre 4). Nous proposons une nouvelle technique de bac à sable en direct pour observer le comportement de kits de hameçonnage tout en protégeant la vie privée des victimes possibles. En utilisant cette technique, nous avons effectué une analyse réelle et complète des attaques par hameçonnage, leurs mécanismes et le comportement des cybercriminels, des victimes et de la communauté de la sécurité impliqués dans le processus – sur une base des données collectées sur une période de cinq mois. Notre infrastructure nous a permis de tracer pour la première fois l'image globale d'une attaque par hameçonnage, depuis un attaquant ait installé et testé les pages de hameçonnage sur un serveur compromis, jusqu'à la dernière interaction avec de vraies victimes et avec des chercheurs en sécurité.

Dans la dernière partie de la thèse, nous étudions comment un hébergeur peut surveiller et protéger ses clients contre des attaques Web, en utilisant une combinaison de techniques de diversion/leurre – qui fournissent une alternative intéressante par rapport aux mécanismes de sécurité traditionnels.

Nous commençons dans le Chapitre 5 en présentant un aperçu de l'utilisation actuelle des techniques de diversion/leurre dans la sécurité informatique, introduisant une classification complète des solutions existantes. De plus, nous analysons plusieurs directions de recherche ouvertes, y compris la conception de stratégies aidant les défenseurs à concevoir et à intégrer la diversion/leurre dans une architecture cible, l'étude des moyens automatisés de déployer la diversion/leurre dans des systèmes complexes et, le plus important, la conception de nouvelles techniques et expérimentations pour évaluer l'efficacité de ces techniques. Enfin, nous discutons les limites des solutions existantes et fournissons des indications pour d'autres recherche sur ce sujet.

Dans le Chapitre 6, nous présentons la conception préliminaire et évaluation d'une approche de détection d'attaques Web basée sur la diversion/leurre, qui vise à à détecter les attaques à leurs débuts.

Au cours d'une journée d'évaluation où les participants recherchent des vulnérabilités dans un site Web, notre système a pu détecter 64% des participants qui ont réussi à exploiter au moins une vulnérabilité. Nous avons également mené une longue expérimentation dans un environnement de production sur une période de sept mois pour évaluer le taux de fausses alarmes. Pendant la période de notre test, le service a été utilisé par 258 utilisateurs différents, générant zéro fausse alerte.

Enfin, dans le Chapitre 7 nous tirons les conclusions et discutons des travaux futurs dans ce domaine.

A.3 ÉTAT DE L'ART

Le travail de cette thèse couvre le domaine de la mesure et de la surveillance de la sécurité, et leur application pour détecter et protéger un hébergeur de services contre une variété de menaces – y compris l'abus de logiciels malveillants, les sites Web compromis hébergeant des kits d'hameçonnage et des attaques Web.

Dans ce Chapitre, nous résumons les études antérieures liées aux techniques que nous présentons dans les parties suivantes de cette thèse. Plus précisément, ceci couvre trois domaines différents. Dans la section A.3.1 nous introduisons des travaux connexes sur la mesure de l'utilisation malveillante des services cloud. Dans la section A.3.2 nous discutons des travaux existants qui étudient et surveillent les attaques par hameçonnage. Enfin, dans la section A.3.4 nous passons en revue les études précédentes et les classifications des techniques de diversion en sécurité informatique. De plus, nous examinons les travaux connexes qui ont utilisé les techniques de la diversion/leurre afin de protéger une application Web.

A.3.1 *Utilisation néfaste des services cloud*

La sécurité des services cloud a été largement étudiée dans la littérature. Comme discuté dans [184], de nombreuses études ont été dédiées à la sécurisation du moniteur de machines virtuelles afin de fournir la sécurité pour les logiciels client et les données contre les exploits et les attaques par canaux auxiliaires. D'autres travaux ont proposé d'utiliser l'introspection de machine virtuelle pour identifier le fonctionnement du système d'exploitation [56] ou pour détecter la présence d'outil de dissimulation d'activité (rootkit) dans le système d'exploitation invité [86, 152]. Enfin, de nombreux documents ont proposé des solutions pour construire un domaine de réseau virtuel sécurisé [34], du stockage sécurisé [66], et le démarrage sécurisé [64] pour les machines virtuelles.

Dans [3], Aceto et al. ont interrogé les plates-formes existantes et les services de surveillance pour le cloud. Les travaux antérieurs ont

couvert divers aspects de la surveillance cloud, y compris la performance, l'accord sur le niveau de service, la qualité des services, la gestion de la capacité et des ressources et la sécurité. Cependant, l'abus des services cloud restent l'une des neuf principales menaces pour le cloud computing [118]. En effet, la menace d'abus et d'utilisation malveillante est davantage un problème pour les hébergeurs de services de cloud que pour les consommateurs de cloud. Malheureusement, du point de vue d'un hébergeur de services, la mesure et le contrôle de la sécurité sont encore insuffisants pour détecter et protéger contres les abus, compromis et attaques.

Des effets secondaires peuvent exister lorsqu'un attaquant abuse des services cloud. Premièrement, certains abus peuvent conduire à des attaques par canaux auxiliaires qui fuient les informations du client [153] ou qui permettent à un attaquant d'extraire les clés privées du client [201]. Deuxièmement, de tels abus peuvent être utilisé comme un facteur d'amplification pour déclencher des attaques distribuées, par exemple en envoyant un grand nombre de messages de spam [31] ou en rejoignant des efforts distribués pour briser la clé de chiffrement [14].

De plus, ce genre d'abus permet aux attaquants de recourir aux services de cloud pour héberger une partie de leur infrastructure de logiciels malveillants [23, 60]. Le terme *logiciel malveillant* est généralement utilisé dans la communauté de la sécurité pour désigner un logiciel indésirable tels que les virus, les chevaux de Troie, les rançongiciels, les vers et les réseaux de zombies – qui présentent un comportement malveillant pour répondre à l'intention des attaquants [58]. Les attaquants commencent généralement par infecter le système de la victime soit par un sabotage technique (par exemple, en exploitant des services réseaux vulnérables ou en effectuant des attaques de "drive-by download" ciblant les navigateurs Web) ou via des méthodes d'ingénierie sociale.

Une fois que la machine cible a été infectée, les logiciels malveillants modernes sont généralement équipés de mécanismes de communication et de télécommande qui fournissent aux attaquants un contrôle total sur l'hôte infecté. Considérons l'exemple d'un bot, qui est un logiciel malveillant sous le contrôle d'une entité malveillante, également appelée "bot-master". Un bot permet au bot-master de livrer des commandes et contrôler à distance le système de la victime. En outre, les bots peuvent être capables d'exfiltrer les données confidentielles de la victime telles que le numéro de carte de crédit et les informations d'identification bancaire à un serveur distant sous le contrôle de l'attaquant. Ce type de fonctionnalité nécessite une infrastructure spécifique, y compris un certain nombre de serveurs malveillants pour recueillir le téléchargement des informations et orchestrer la communication de *commande et contrôle* (C&C).

Pour détecter les serveurs C&C malveillants, les travaux antérieurs se sont concentrés sur l'analyse du réseau local, où ils ont analysé la corrélation spatio-temporelle des activités préprogrammées liées au C&C, telles que la communication coordonnée et la propagation [68]. Plus récemment, les chercheurs se sont tournés vers les réseaux de fournisseur d'accès Internet (FAI) à grande échelle [24]. Cependant, on en sait peu sur l'infrastructure des logiciels malveillants, et en particulier sur le rôle des services cloud pour héberger ces logiciels malveillants.

Les abus antérieurs des hébergeurs de service cloud ont attiré beaucoup d'attention au cours des dernières années [23, 60, 77, 182]. Par exemple, les services de cloud sont répertoriés par Solutionary parmi les composants majeurs de cybercriminalité moderne, et les attaquants *"semblent utiliser ces services de la même manière et pour les mêmes raisons que les clients légitimes"* [169]. Malgré cette popularité, nous avons connaissance seulement de quelques études de recherche qui ont réussi à évaluer l'ampleur réelle de ce phénomène.

Hamza et al. [71] a présenté un aperçu des techniques possibles pour abuser les services de cloud dans la cybercriminalité moderne. Les auteurs ont fourni d'intéressantes idées sur la façon dont les cyber-attaques ont été perpétrées à partir de plates-formes cloud, y compris des exemples tels que les attaques de saut d'hôte et l'abus de privilèges. Cependant, cette étude se concentre uniquement sur les signaux d'attaque forts, et ne considère pas d'autres signaux faibles qui déterminent les façons possibles dont les services de cloud sont utilisés dans le cadre des infrastructures de commande et de contrôle des attaquants.

Dans [132], Nappa et al. ont analysé les attaques de "drive-by download" et les serveurs exploités qui ont été gérés par les mêmes organisations. Ils ont trouvé que 60% des hébergeurs de services qui hébergeaient les serveurs d'exploit étaient en effet des hébergeurs de services cloud. Plus intéressant, ils ont évalué les procédures de rapport d'abus mis en œuvre par les hébergeurs de services de cloud public. Les auteurs ont découvert que sur 19 rapports d'abus qu'ils ont soumis, seulement 7 ont été étudiés par les hébergeurs de service cloud. De plus, les auteurs ont calculé qu'il faut en moyenne 4,3 jours pour un hébergeur de cloud de supprimer un serveur d'exploit après qu'il a été signalé. Il est important de noter que les auteurs de cette étude se concentrent uniquement sur les attaques "drive-by-download" impliquant des services cloud. Bien que les serveurs "drive-by-download" constituent un composant majeur de l'infrastructure du logiciel malveillant moderne, dans cette thèse, nous allons au-delà de ce cas d'utilisation unique pour fournir une évaluation plus complète sur la façon dont les services de cloud sont intégrés dans l'infrastructure de logiciels malveillants dans la cybercriminalité moderne. Nous avons aussi essayé de comprendre si les services

cloud constituent les éléments principaux de l'infrastructure du logiciel malveillant, ou si ils ne sont utilisés que comme des composants redondants ou de basculement.

Canali et al. [36] ont proposé une approche active à évaluer les mécanismes de sécurité mis en place par les hébergeurs de sites Web. Ils ont installé des services Web vulnérables sur 22 hébergeurs distincts, et déclenché plusieurs attaques pour tirer parti des capacités de réaction de ces hébergeurs. Pour tester les mécanismes de sécurité implémentés par les hébergeurs de service cloud, dans le Chapitre 3 nous adoptons une approche moins intrusive où nous observons seulement les interactions de logiciels malveillants avec le service cloud. Dans notre étude, nous nous concentrons uniquement sur le cloud d'Amazon EC2. Bien que ce choix puisse limiter la mesure de nos observations, en même temps il peut éliminer autant que possible l'impact des hébergeurs nocif ou ceux qui ne garantissent pas les exigences minimales de sécurité pour leurs clients. Nous considérons que la concentration seulement sur le plus grand hébergeur de cloud en termes de part de marché peut également éclairer les limites de mécanismes de sécurité actuels et de responsabilité mis en œuvre par les hébergeurs de cloud d'aujourd'hui.

Wang et al. [188] ont proposé un système qui mesure le taux de désabonnement des hébergeurs de services cloud public (par exemple, EC2 et Azure) afin d'évaluer l'efficacité des listes noires IP contre les activités malveillantes basées sur le cloud. Les auteurs ont activement sondé les adresses IP appartenant au cloud EC2 et Azures, et ont proposé un mécanisme de regroupement qui regroupe ensemble des adresses IP implémentant les mêmes services. Ils ont également observé tous les services Web hébergés par les hébergeurs de cloud, couvrant à la fois les activités malveillantes et bénignes. Les résultats de leurs expériences n'ont trouvé que une faible quantité d'activités malveillantes (principalement l'hameçonnage et l'hébergement de logiciels malveillants) en comparant les données de leur système avec les listes noires publiques. Dans le Chapitre 3, nous proposons une approche complémentaire qui observe uniquement les interactions de logiciels malveillants avec les service cloud afin de tirer parti de la véritable étendue de l'activité malveillante hébergée par les hébergeurs de cloud public.

Plus récemment, Liao et al. [114] ont analysé les dépôts du cloud public qui ont été abusés par des cybercriminels dans le but de mener des activités illicites. Les auteurs ont également caractérisé l'efficacité des liens de spam qui a abusé les services d'hébergement cloud [115]. Tajalizadehkhoob et al. [177] ont analysé les données de communication C&C sur une période de sept ans et ont constaté que les attaquants ont peu de préférence pour les hébergeurs où les domaines C&C ont relativement une longue durée de vie.

Enfin, dans un article récent publié en 2017, Lever et al. [111] ont étudié les activités réseaux d'environ 26M échantillons de logiciels malveillants recueillis de 2011 à 2015. Ils ont pu confirmer une tendance similaire et obtenir des résultats en ligne avec ce que nous présentons dans Chapitre 3, mais à plus grande échelle qui couvrait plusieurs hébergeurs de service cloud. Ils ont également constaté que les programmes potentiellement indésirables (PUP) étaient celles parmi les différentes familles de logiciels malveillants qui ont utilisé le plus l'infrastructure de cloud et les réseaux de diffusion de contenu (CDN).

Pour conclure, le travail de cette thèse présente la première mesure systématique de la sécurité des services de cloud public en ce qui concerne les abus de logiciels malveillants. Des études récentes ont porté sur les abus des services cloud depuis différents points de vue.

A.3.2 *Comprendre les attaques par hameçonnage*

La littérature scientifique comprend un grand nombre de documents liés aux attaques par hameçonnage. En particulier, les chercheurs ont proposé de nombreuses techniques à étudier, bloquer et supprimer les sites d'hameçonnage. Cependant, il n'y a pas encore de système de surveillance complet qui permet l'observation de l'ensemble du cycle de vie d'un kit d'hameçonnage. Nous classons les études existantes dans les trois catégories suivantes : anatomie de l'attaque par hameçonnage, techniques d'anti-hameçonnage, et l'évaluation des techniques d'anti-hameçonnage.

A.3.2.1 *Anatomie de l'attaque par hameçonnage*

Le travail le plus étroitement lié à cette thèse a été effectué par Waston et al., qui a décrit deux incidents d'attaques par hameçonnage [190] qui ont été découverts par le projet Honeynet [172]. Les auteurs ont décrit comment les attaquants se comportent et les techniques utilisées pour mettre en place les sites d'hameçonnage. L'un des deux kits d'hameçonnage a reçu 256 requêtes HTTP entrantes, mais apparemment aucune donnée personnelle n'a été soumise par les visiteurs. Pourtant, les auteurs ont dû fermer le pot de miel (honeypot) parce qu'ils n'avaient pas un système pour empêcher les données de l'utilisateur d'être volées par les attaquants. Notre travail présenté dans le Chapitre 4 adopte une approche similaire basée sur le pot de miel, mais se concentre sur un système éthique pour étudier comment les attaques par hameçonnage dans le monde réel sont structurées. McGrath et al. [120] ont analysé la façon comment les hameçonneurs ont effectué leurs attaques, les caractéristiques des liens d'hameçonnage, les domaines et leur infrastructure d'hébergement. Les auteurs ont estimé également la durée de vie des

noms de domaine utilisés pour l'hameçonnage en utilisant des enregistrements DNS collectés périodiquement. Moore et al. [126] ont présenté la preuve que les cybercriminel utilisent le moteur de recherche («Google Hacking») pour compromettre et re-compromettre des machines, qui sont en outre utilisées pour héberger des sites d'hameçonnage. Dans un autre travail, Moore et al. [129] ont étudié les corrélations temporelles entre le spam et les sites d'hameçonnage afin de comprendre le comportement des attaquants, et de évaluer l'efficacité du retrait du site d'hameçonnage. Sheng et al. [165] ont mené une analyse démographique pour évaluer la susceptibilité des victimes face aux attaques d'hameçonnage et discuté l'efficacité du matériel éducatif.

Quelques études ont porté sur l'estimation du taux de réussite des emails d'hameçonnage. Jagatic et al. ont d'abord rapporté le taux de réussite des e-mails d'hameçonnage simulés [82], tandis que Jakobsson et al. ont proposé des expérimentations d'hameçonnage éthiques sur un site web d'enchère en ligne populaire [85], afin de mesurer le taux de réussite des e-mails d'hameçonnage simulés. Le taux de succès rapporté est respectivement environ 15% et 11% (comparé à 9% que nous avons trouvé dans notre étude). Cependant, ces travaux ont mesuré le taux de réussite basé uniquement sur les attaques par hameçonnage *simulées*.

Plusieurs études ont mesuré l'impact de l'hameçonnage sur les victimes potentiels. Moore et al. ont mesuré empiriquement la durée de vie des sites d'hameçonnage et le nombre de réponses de victimes [125]. Les auteurs ont récupéré des rapports confirmés sur PhishTank, puis se sont appuyés sur les enregistrements générés par Webalizer, un outil gratuit d'analyse du journal du serveur Web, en cas où il était déjà installé sur les sites Web compromis. Cependant, cet outil enregistre simplement le nombre de visites pour une page Web donnée au lieu du nombre unique de visites, ce qui rend les résultats d'estimation très approximatives. Les auteurs ont également pu estimer le nombre de victimes de 20 sites d'hameçonnage, en supposant que seuls les utilisateurs victimes du hameçonnage seraient redirigés vers une page de confirmation après qu'ils aient fourni leurs informations d'identification. Trusteer a mesuré l'efficacité de attaques par hameçonnage basées sur les statistiques recueillies par un plugin de navigateur sur une période de trois mois [180]. Les auteurs ont constaté qu'en 2009, 45% des victimes observés qui ont été connectées à un site d'hameçonnage ont révélé leur informations d'identification personnelles. Cependant, leur étude ne fournit qu'une vue partielle de l'attaque par hameçonnage.

Enfin, Cova et al. ont analysé dans [49] les kits d'hameçonnage librement disponibles et quelques sites d'hameçonnage en ligne. Ils ont analysé les organisations ciblées, les techniques utilisées pour ex-

filtrer les données et les méthodes d'obfuscation mises en place dans les kits d'hameçonnage.

A.3.2.2 *Techniques d'anti-hameçonnage*

Les contre-mesures d'hameçonnage ont suscité beaucoup d'intérêt dans communauté scientifique. Les contre-mesures proposées peuvent être regroupées en trois catégories : la détection de pages d'hameçonnage, le blocage et la formation des utilisateurs.

La plupart des techniques de détection d'hameçonnage identifient les pages d'hameçonnage en créant un classifieur utilisant différentes heuristiques basées sur les caractéristiques d'URL [62, 110] ou sur le contenu de la page web [144, 195, 203]. Certains études visent à détecter et bloquer les attaques par hameçonnage en différentes étapes. Par exemple, Fette et al. [61] ont utilisé une machine d'apprentissage pour identifier et bloquer les emails d'hameçonnage. Plusieurs études ont proposé un plugin navigateur pour protéger les utilisateurs contre le hameçonnage [40, 54, 88]. Une autre approche populaire de bloquer les attaques par hameçonnage consiste à compiler et à distribuer des listes noires, telles que Google Safe Browsing et PhishTank. De nombreuses études ont porté sur l'éducation contre les attaques par hameçonnage et comment former les utilisateurs à identifier l'hameçonnage [102–104, 139]. Enfin, une seule étude a porté sur l'identification des adresses électroniques des attaquants en utilisant des sites d'hameçonnage vérifiés par PhishTank et des métadonnées fournies par les fournisseurs d'email [127]. Cependant, cette méthode introduit une latence significative par rapport à notre approche puisque les listes noires publiques peuvent ne pas être suffisamment efficace pour détecter rapidement les kits d'hameçonnage en direct.

A.3.3 *Évaluation de techniques d'anti-hameçonnage*

En 2006, un certain nombre d'études ont conclu que les solutions d'anti-hameçonnage [202], les indicateurs de sécurité [55], et Les barres d'outils de navigateur [196] étaient insuffisantes pour détecter les sites d'hameçonnage et protéger les utilisateurs.

En 2007, Ludl et al. [119] et Sheng et al. [166] se sont spécifiquement concentrés sur l'efficacité de listes noires pour empêcher les attaques par hameçonnage, mais ils ont obtenu des résultats contradictoires. Dans la première étude, les auteurs ont collecté des liens de site d'hameçonnage en ligne depuis PhishTank, et les a testés contre Google Safe Browsing et aussi le filtre Phish de Microsoft Internet Explorer. Cette étude a conclu que l'approche basée sur la liste noire est efficace pour protéger les utilisateurs, en particulier Google qui a correctement détecté près de 90% des liens de site d'hameçonnage [119]. Dans la seconde étude, Sheng et al. ont évalué cinq listes noires (y compris celles testées par Ludl et al.) avec les URLs d'hameçonnage datant de

moins de 30 minutes, collectées auprès de dépôt de données email de l'Université de Alabama Phishing Team. Cette étude a trouvé que les listes noires étaient inefficaces car la plupart d'entre elles détectaient moins de 20% de nouvelles pages d'hameçonnage [166].

Egelman et al. ont mené une étude empirique pour évaluer l'efficacité de l'avertissement d'hameçonnage du navigateur Web et fourni quelques conseils sur la façon d'améliorer les indicateurs de sécurité [59]. En 2014, Gupta et al. [69] ont évalué l'efficacité des pages de sensibilisation contre les attaques par hameçonnage [102] pour déterminer si elles ont aidé les utilisateurs à identifier les tentatives d'hameçonnage.

En résumé, la plupart des études existantes ont évalué l'efficacité de techniques d'anti-hameçonnage seulement *après* que la page d'hameçonnage ne soit rapportée publiquement ou en privé. Notre étude évalue plutôt l'efficacité de l'approche par liste noire dès le début des attaques par hameçonnage. Pour atteindre cet objectif, le travail présenté dans cette thèse propose un mécanisme de surveillance qui profite du point de vue d'un hébergeur de services, et qui permet l'observation de tout le cycle de vie de l'attaque par hameçonnage dans la nature.

A.3.4 *Techniques de diversion/leurre dans la sécurité informatique*

Dans cette thèse, nous étudions les techniques de diversion/leurre existantes et proposons une nouvelle classification selon quatre dimensions. Nous recourons également aux techniques de diversion/-leurre, qui nous permettent d'obtenir une meilleure protection pour les applications Web. Dans la suite, nous présentons d'abord les études et classifications existantes autour de ce sujet, et laissons une présentation plus complète des techniques liées à la diversion/leurre dans l'étude présentée au Chapitre 5. Ensuite, nous résumons les travaux antérieurs qui ont utilisé des techniques de diversion/leurre pour surveiller, détecter et atténuer les attaques Web.

A.3.4.1 *Études précédentes*

Dans [170], Spitzner a discuté l'utilisation de système de pot de miel et des techniques de l'honeytoken comme un moyen de protection contre la menace interne. Dans [157], Rowe a discuté de différentes possibilités d'intégrer la diversion/leurre dans les systèmes de pots de miel – tels que les informations de leurre, les retards, les faux messages d'erreur – et les a comparés à d'autres possibilités d'utiliser la diversion/leurre dans les systèmes informatiques réels. Voris et al. [187] ont discuté des cas d'utilisation divers où les leurres peuvent être importants pour la sécurité informatique. Juels et Tech [91] ont discuté l'utilisation des objets de miel (un terme générique utilisé

pour désigner plusieurs types de diversion/leurre) pour améliorer la sécurité des systèmes d'information. Jogdand et Padiya [87] ont également analysé les solutions IDS et la façon dont ils peuvent mettre en œuvre des honeytokens, qui sont en effet un type spécifique de la diversion/leurre. Dans [46], Cohen et al. ont aperçu plusieurs problèmes lors de l'application de la diversion/leurre dans la sécurité informatique, et introduit leur propre cadre pour la diversion/leurre. Les auteurs ont également discuté les principaux défis à relever pour mettre en œuvre dans la pratique les technique de diversion/leurre en utilisant ce cadre pour défendre le système d'information. Cohen a également étudié dans [44] l'utilisation historique et récente (jusqu'en 2005) des pots de miel et des leurres pour la protection de l'information, ainsi que la théorie derrière les techniques de diversion/leurre et leurs limites.

Notre travail est différent des études précédentes parce qu'il examine les contributions récentes, en se concentrant sur les défis techniques et évaluations de la diversion/leurre, au lieu d'étudier l'histoire de ce concept et comment il a trouvé son chemin dans la sécurité informatique.

A.3.4.2 *Classifications précédentes*

Les schémas de classification précoce de la diversion/leurre suivaient un système militaire traditionnel de classification de la tromperie. Par exemple, Cohen [42] a examiné les techniques de diversion/leurre basées sur la nature de ces techniques, y compris *"la dissimulation, le camouflage, des informations plantées et fausses, les ruses, les affichages, les démonstrations, les feintes, les mensonges et la perspicacité"*. L'examen historique a révélé que la diversion/leurre était loin d'être pleinement exploité en sécurité informatique. Le même type de taxonomie a également été utilisé dans des travaux ultérieurs [159, 160].

Rowe et Rothstein [159] ont développé une théorie de la diversion/-leurre qui est inspirée de la théorie linguistique computationnelle des cas sémantiques. Les rôles de cas sémantiques peuvent contenir le participant (agent, bénéficiaire), l'espace (direction, emplacement), le temps, la causalité (cause, but), la qualité (contenu, valeur), etc. Les auteurs ont expliqué une opération de diversion/leurre comme une action qui modifie les valeurs du rôle de cas associé. Cohen [44] a proposé un modèle de diversion/leurre informatique qui regroupe ces techniques par le niveau hiérarchique de l'ordinateur auquel ils sont appliqués, du plus bas niveau matériel au niveau le plus élevé d'application. Les informations et signaux entre différents niveaux peut être induits ou inhibés afin d'implémenter la diversion/leurre. De même, Almeshekah et Spafford [7] ont classées les techniques de diversion/leurre basé sur l'état de système et les composants où la diversion/leurre peut être déployée. Les catégories de premier niveau

incluent la fonctionnalité du système (décisions système, logiciels et services, données internes) et l'état du système (activités, configurations et performance).

Gartner [149] a proposé une pile de diversion/leurre à quatre couches, y compris les couches de réseau, d'hôte, d'application et de données. De plus, il a analysé les techniques de diversion/leurre mises en œuvre par les produits commerciaux actuels. Gartner [149] et Almeshekah et Spafford [8] ont tous examiné les possibilités de déployer des techniques de diversion/leurre au cours de différentes phases d'une cyberattaque, sans aucune classification systématique.

La plupart des classifications précédentes ne prenaient en compte qu'une seule dimension, telle que le composant ou la granularité de chaque technique [7], ou la couche où la diversion/leurre est appliquée [149]. Ces classifications mono dimensionnelles ont échoué à capturer d'autres aspects de la diversion/leurre qui sont d'égale importance, telles que les menaces couvertes par chaque technique, et la façon dont le mécanisme de déception peut être intégré à l'intérieur d'un système cible. Pour prendre en compte tous ces différents aspects, cette thèse introduit un nouveau système de classification complet basé sur quatre dimensions orthogonales.

A.3.4.3 *Protection d'application Web basée sur la diversion/leurre*

La littérature déborde d'approches pour protéger les applications web [112] contre les attaques. Néanmoins, les techniques de la détection d'attaques Web actuelles ne parviennent pas à détecter de manière fiable et active les attaques à leurs stade précoce. En raison de ces limitations, des solutions complémentaires telles que les techniques de diversion/leurre ont été récemment étudiées par la communauté scientifique [8, 13]. Dans cette thèse, nous nous concentrons principalement sur les solutions qui adoptent une approche de protection des applications Web basée sur la diversion/leurre. Nous regroupons les approches existantes en deux catégories, l'une utilisée pour améliorer la détection des attaques et l'autre utilisée dans le but d'atténuation des attaques.

Détection d'attaque Brewer et al. [30] ont proposé une application web qui intègre des liens contenant des leurres. Ces liens sont invisibles pour les utilisateurs normaux, mais devraient être déclenchés par les robots d'exploration et les robots Web qui se connectent à l'application. De même, Gavrilis et al. [65] ont présenté une méthode qui détecte les attaques par déni de service sur les services Web en utilisant les faux liens cachés dans la page Web. De la même façon, McRae et Vaughn [121] ont soumis des faux comptes qui contenait l'URL avec du leurre aux sites d'hameçonnage pour suivre les attaquants quand ils regardaient ces faux comptes.

Une autre approche pour tromper les attaques Web consiste à utiliser de fausses informations déguisées en erreurs de configuration du serveur Web. Seulement les utilisateurs malveillants sont censés manipuler ou exploiter ces erreurs, qui les exposent à la détection par le système. Dans ce cadre, Virvilis et al. [185] ont introduit de faux fichiers de configuration, tels que `robots.txt`, y compris les fausses entrées, les liens cachés et des commentaires HTML qui indiquent les faux comptes, afin de détecter les attaquants. D'autres études ont proposé de faux formulaires en tant que leurre [97] et de faux paramètres d'URL [147] qui affichent de fausses erreurs de configuration dans le but d'égarer les attaquants et de protéger le système cible.

Dans [6], Almeshekah a proposé un serveur de diversion/leurre centralisé ce qui permet l'implémentation de la diversion pour protéger les serveurs web cibles. Dans chaque serveur Web, le système proposé analyse les requêtes entrantes et envoie leurs méta données vers le serveur centralisé où une décision est prise quant à l'attitude du système et s'il devrait répondre avec la diversion/leurre. L'auteur a ensuite analysé les impacts sur la performance introduits par le système.

Atténuation d'attaque Julian [92] a proposé de modifier le temps de réponse d'un moteur de recherche Web en injectant des retards aléatoires en réponse aux requêtes malveillantes afin de confondre un attaquant. Anagnostakis et al. [12] ont introduit *"le pot de miel d'ombre (shadow honeypots)"* qui étend les pots de miel traditionnels avec la détection de dépassement de tampon basé sur les anomalies pour protéger l'application Web. Le pot de miel d'ombre est une copie de l'application cible, avec le même contexte et l'état d'application. Il est utilisé pour analyser le trafic anormal et améliorer la précision de la détection basée sur les anomalies.

Finalement, Araujo et al. [16] ont présenté une technique pour transformer les correctif traditionnels en *"honey-patches"*, conçu pour supprimer la vulnérabilité tout en leurrant les attaquants en leur faisant croire que leurs attaques ont réussi. Sur la détection d'une attaque ciblant la vulnérabilité connue du serveur Web, le système redirige l'attaquant à un serveur non corrigé mais isolé du serveur d'origine. Les auteurs ont étendu leur système de "honey-patches" avec un compilateur instrumenté qui nettoie automatiquement les informations d'identification dans le serveur non corrigé [15].

Le travail dans cette thèse diffère des travaux ci-dessus parce que nous nous concentrons sur la détection d'attaque au lieu de l'atténuation d'attaque. Malheureusement, notre étude souligne que la plupart des travaux précédents dans la détection de l'attaque basée sur la diversion/leurre n'ont pas fourni d'expérimentation ou d'évaluations sur les techniques proposées. Par conséquent, dans le Chapitre 6 nous présentons deux expérimentations que nous avons conduites pour

évaluer l'efficacité et le taux de faux positifs de protection d'application web basée sur la diversion/leurre.

A.4 CONCLUSIONS ET TRAVAUX FUTURS

Cette thèse a discuté de plusieurs directions qu'un hébergeur de services peut suivre, en plus de simplement sécuriser son infrastructure, pour fournir une meilleure sécurité pour ses clients. Un certain nombre de techniques ont été proposées, depuis le point de vue d'un hébergeur de services, pour améliorer sa capacité à mesurer et surveiller les menaces contre la sécurité – y compris les abus possibles, les serveurs compromis et les attaques externes.

La thèse a d'abord proposé une étude systématique pour confirmer que l'hébergeur de service cloud a été abusé par des attaquants pour héberger une partie de leur infrastructure malveillante, et mieux comprendre l'ampleur de ce phénomène. En effectuant une analyse à grande échelle d'environ un million d'échantillons de logiciels malveillants, nos résultats ont montré que les cybercriminels soutenaient des opérations de longue durée à travers l'utilisation de services cloud public. De plus, nous avons observé une tendance à la hausse de nombre de domaines cloud malveillants et dédiés de 2008 à 2014. Les mécanismes de sécurité existants employés par les hébergeurs de services étaient insuffisants pour mesurer et détecter correctement ce type d'abus.

La deuxième partie de la thèse a exploré les informations précieuses auxquelles l'hébergeur de services a accès, qui peuvent être utilisées pour analyser les attaques en temps réel – en mettant l'accent sur les attaques par hameçonnage précédemment étudiées par des chercheurs externes dans une portée plus limitée. En particulier, nous avons présenté la conception et la mise en œuvre de PhishEye, un système spécialement conçu pour analyser les kits d'hameçonnage de manière éthique. En utilisant ce système, nous avons effectué une expérimentation durant cinq mois qui nous a permis pour la première fois de comprendre et de mesurer l'ensemble du cycle de vie de ce type d'attaque. Nous étions capable de distinguer les victimes des attaquants et des autres visiteurs tiers. Nos résultats ont montré que la majeure partie des activités de victimes se déroulent à la première période après l'installation du kit d'hameçonnage, et avant la communauté de la sécurité découvre son existence.

Enfin, nous avons discuté des techniques de diversion/leurre qu'un hébergeur de services peut employer afin d'ajouter une couche de sécurité supplémentaire pour ses applications client. Nous avons étudié les techniques de diversion/leurre existantes et proposé une classification complète qui nous a permis d'identifier les directions de recherche ouvertes – y compris la conception et la modélisation des techniques de diversion/leurre, leur déploiement et l'évaluation de

ces techniques. De plus, nous avons présenté la conception et l'évaluation d'une approche basée sur la diversion/leurre pour protéger les applications Web des clients. Les résultats d'une expérimentation offensive ont montré que notre approche était capable de détecter 64% des participants qui ont réussi à exploiter au moins une vulnérabilité. Dans une autre expérimentation séparée dans un environnement de production, nous avons mis en œuvre des techniques de diversion/-leurres similaires dans un système qui était couramment utilisé par 258 utilisateurs différents. Pendant une période de sept mois, zéro fausse alerte était élevée par le système.

La sécurité du point de vue d'un hébergeur de services a attiré peu d'attention parmi les chercheurs en sécurité auparavant. Le travail dans cette thèse a d'abord révélé le phénomène que les cybercriminels ont utilisé les services de cloud public pour héberger leur infrastructure malveillante et que les mécanismes de sécurité existants ont été incapables de détecter en temps opportun ce type d'abus. Les hébergeurs de services sont toujours réticents à mener des recherches sur cette direction en raison de la préoccupation des clients sur la protection de la vie privée et de leurs données. Il serait intéressant de travailler avec les hébergeurs de services cloud public pour comprendre pleinement le rôle précis de ces services abusifs, ce qui permettrait aux chercheurs de construire rapidement un moyen de détection de ce type de menace. Dans notre étude, nous avons seulement regardé les données de la communication réseau, mais il serait intéressant pour un hébergeur de services de combiner les données réseau avec les informations système telles que l'utilisation du processeur et les compteurs de performance pour mesurer et détecter les clients malveillants.

Récemment, quelques travaux ont émergé qui proposaient de mesurer et de comprendre le rôle des hébergeurs de services dans les différents cas d'abus, principalement sur le stockage cloud. Une autre extension intéressante de notre travail consisterait à étudier le rôle de l'hébergeur de services dans d'autres types d'attaques – telles que les services par déni de service distribué ou la distribution de logiciels malveillants. Ce genre d'étude finirait par sensibiliser la communauté scientifique à l'importance de ce phénomène et aider les autres hébergeurs à détecter les attaques connexes.

Le deuxième travail dans cette thèse a mis en évidence la capacité d'un hébergeur de services à surveiller des attaques par hameçonnage pour l'ensemble de leur cycle de vie, ce qui était précédemment inconnu aux chercheurs externes. Notre étude a illustré la possibilité qu'un hébergeur de services d'application peut surveiller la sécurité des ses client d'une manière éthique. Alors que dans notre travail, nous avons eu recours à une implémentation d'un pot de miel, il serait intéressant que les hébergeurs de services implémentent notre système dans les environnements de production du monde réel.

En outre, d'autres études pourraient être menées pour comprendre d'autres types d'attaque qui étaient autrement difficiles à analyser pour des chercheurs externes, y compris l'utilisation de kits d'exploitation dans la nature. Cela permettrait aux chercheurs d'obtenir un aperçu des différents acteurs, de leurs comportements et de l'ensemble du cycle de vie d'un kit d'exploit.

La dernière partie de la thèse a appliqué des techniques de diversion/leurre pour permettre aux hébergeurs de services de proposer une couche supplémentaire de sécurité pour les applications Web de leurs clients. Nous avons mené deux expérimentations où nous avons obtenu un taux de détection raisonnable sans faux positifs. Les résultats semblent soutenir notre hypothèse que les hébergeurs de services sont capables de fournir une meilleure sécurité pour leurs applications client, mais plus d'expérimentations sont nécessaires pour étudier comment le déploiement des éléments de diversion/leurre affecte le taux de détection d'attaque. Ce serait aussi intéressant d'appliquer des techniques similaires sur d'autres services tels que FTP, email, et bases de données. De plus, il n'y a pas encore de consensus parmi les chercheurs sur la façon de déployer et d'évaluer les techniques de diversion/leurre quand elles sont utilisées comme un mécanisme de défense. Plus d'expérimentations sont réellement nécessaires pour évaluer ces techniques dans divers environnements d'application.

En conclusion, cette thèse a étudié de différentes directions dans lesquelles un hébergeur de services peut mesurer l'infrastructure malveillante, surveiller les attaques par hameçonnage installées sur des services compromis, et enfin protéger les applications clients utilisant des techniques de diversion/leurre. Cependant, nous regardons seulement la pointe de l'iceberg et nous n'avons pas pleinement exploré tous les informations auxquelles un hébergeur de services a accès, ce qui pourrait nous aider à enquêter sur d'autres attaques ou proposer d'autres mécanismes de protection. Par conséquent, nous croyons que les hébergeurs de services joueront un rôle de plus en plus important pour fournir des services sécurisés aux clients et aux autres utilisateurs d'Internet.

BIBLIOGRAPHY

[1] Greg Aaron and Ronnie Manning. *APWG Global Phishing Report 2014*. http://apwg.org/download/document/245/APWG_Global_Phishing_Report_2H_2014.pdf. 2014.

[2] Greg Aaron and Ronnie Manning. *APWG Phishing Activity Trends Report 2015*. https://docs.apwg.org/reports/apwg_trends_report_q1-q3_2015.pdf. 2015.

[3] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. « Cloud monitoring: A survey. » In: *Computer Networks* 57.9 (2013), pp. 2093–2115.

[4] Mitsuaki Akiyama, Takeshi Yagi, Kazufumi Aoki, Takeo Hariu, and Youki Kadobayashi. « Active credential leakage for observing web-based attack cycle. » In: *International Workshop on Recent Advances in Intrusion Detection*. 2013.

[5] Boniface Kayode Alese, FM Dahunsi, RA Akingbola, OS Adewale, and TJ Ogundele. « Improving deception in honeynet: Through data manipulation. » In: *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for*. IEEE. 2014, pp. 198–204.

[6] Mohammed H Almeshekah. « Using deception to enhance security: A Taxonomy, Model, and Novel Uses. » PhD thesis. 2015.

[7] Mohammed H Almeshekah and Eugene H Spafford. « Planning and integrating deception into computer security defenses. » In: *ACM Workshop on New Security Paradigms Workshop (NSPW)*. 2014.

[8] Mohammed Almeshekah and Eugene Spafford. « The case of using negative (deceiving) information in data protection. » In: *International Conference on Cyber Warfare and Security*. 2014.

[9] Lance Alt, Robert Beverly, and Alberto Dainotti. « Uncovering network tarpits with degreaser. » In: *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM. 2014, pp. 156–165.

[10] Amazon Web Services. *Shared Responsibility Model*. https://aws.amazon.com/compliance/shared-responsibility-model/. 2017.

[11] EC Amazon. *Amazon elastic compute cloud (Amazon EC2)*. http://aws.amazon.com/ec2/. 2010.

[12] Kostas G Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Konstantinos Xinidis, Evangelos P Markatos, and Angelos D Keromytis. « Detecting Targeted Attacks Using Shadow Honeypots. » In: *Usenix Security*. 2005.

[13] Chuvakin Anton. *"Deception as Detection" or Give Deception a Chance?* http://blogs.gartner.com/anton-chuvakin/2016/01/08/deception-as-detection-or-give-deception-a-chance. 2016.

[14] Jacob Appelbaum, Dino Dai Zovi, and Karsten Nohl. *Crippling Crypto: The Debian OpenSSL Debacle*. https://trailofbits.files.wordpress.com/2008/07/hope-08-openssl.pdf. 2008.

[15] Frederico Araujo and Kevin W Hamlen. « Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception. » In: *USENIX Security*. 2015.

[16] Frederico Araujo, Kevin W Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. « From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. » In: *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 2014.

[17] Rana Aamir Raza Ashfaq, Xi-Zhao Wang, Joshua Zhexue Huang, Haider Abbas, and Yu-Lin He. « Fuzziness based semi-supervised learning approach for intrusion detection system. » In: *Information Sciences* (2017).

[18] Windows Azure. *Microsoft's cloud platform*. http://azure.microsoft.com/. 2013.

[19] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. « Scalable, Behavior-Based Malware Clustering. » In: *NDSS*. Vol. 9. 2009, pp. 8–11.

[20] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. « TTAnalyze: A tool for analyzing malware. » In: *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*. 2006.

[21] Malek Ben Salem and Salvatore J. Stolfo. « Decoy document deployment for effective masquerade attack detection. » In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2011.

[22] Maya Bercovitch, Meir Renford, Lior Hasson, Asaf Shabtai, Lior Rokach, and Yuval Elovici. « HoneyGen: An automated honeytokens generator. » In: *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics, ISI 2011* (2011).

[23] Dmitry Bestuzhev. *Financial data stealing Malware now on Amazon Web Services Cloud*. http://www.securelist.com/en/blog/208188099/Financial_data_stealing_Malware_now_on_Amazon_Web_Services_Cloud. [accessed 15-May-2014]. 2011.

[24] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. « Disclosure: detecting botnet command and control servers through large-scale netflow analysis. » In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012, pp. 129–138.

[25] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. « Kamouflage: Loss-Resistant Password Management. » In: *European Symposium on Research in Computer Security*. 2010.

[26] Kevin Borders, Laura Falk, and Atul Prakash. « OpenFire: Using deception to reduce network attacks. » In: *Security and Privacy in Communications Networks and the Workshops*. 2007.

[27] Brian M. Bowen, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo. « Baiting inside attackers using decoy documents. » In: *International Conference on Security and Privacy in Communication Systems* (2009).

[28] Brian M Bowen, Vasileios P Kemerlis, Pratap Prabhu, Angelos D Keromytis, and Salvatore J Stolfo. « Automating the Injection of Believable Decoys to Detect Snooping. » In: *Proceedings of the Third ACM Conference on Wireless Network Security* (2010).

[29] Brian M Bowen, Pratap Prabhu, Vasileios P Kemerlis, Stelios Sidiroglou, Angelos D Keromytis, and Salvatore J Stolfo. « Botswindler: Tamper resistant injection of believable decoys in vm-based hosts for crimeware detection. » In: *International Workshop on Recent Advances in Intrusion Detection*. 2010.

[30] Douglas Brewer, Kang Li, Laksmish Ramaswamy, and Calton Pu. « A link obfuscation service to detect webbots. » In: *International Conference on Services Computing (SCC)* (2010).

[31] Carl Brooks. *Amazon EC2 email blocked by antispam group Spamhaus*. http://searchaws.techtarget.com/news/1371369/Amazon-EC2-email-blocked-by-antispam-group-Spamhaus. 2009.

[32] Elie Bursztein, Borbala Benko, Daniel Margolis, Tadek Pietraszek, Andy Archer, Allan Aquino, Andreas Pitsillidis, and Stefan Savage. « Handcrafted fraud and extortion: Manual account hijacking in the wild. » In: *Internet Measurement Conference (IMC)*. 2014.

[33] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. « Measuring Pay-per-Install: The Commoditization of Malware Distribution. » In: *20th USENIX conference on Security*. 2011.

[34] Serdar Cabuk, Chris I Dalton, HariGovind Ramasamy, and Matthias Schunter. « Towards automated provisioning of secure virtualized networks. » In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, pp. 235–245.

[35] Davide Canali and Davide Balzarotti. « Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web. » In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2013.

[36] Davide Canali, Davide Balzarotti, and Aurélien Francillon. « The role of web hosting providers in detecting compromised websites. » In: *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2013, pp. 177–188.

[37] Thomas E Carroll and Daniel Grosu. « A game theoretic investigation of deception in network security. » In: *Security and Communication Networks* (2011).

[38] A Čenys, D Rainys, L Radvilavičius, and N Goranin. « Implementation of Honeytoken Module In DBMS Oracle 9ir2 Enterprise Edition for Internal Malicious Activity Detection. » In: *IEEE Computer Society's TC on Security and Privacy*. 2005.

[39] Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. « Detecting traffic snooping in tor using decoys. » In: *Workshop on Recent Advances in Intrusion Detection* (2011).

[40] Neil Chou, Robert Ledesma, Yuka Teraguchi, and John C Mitchell. « Client-Side Defense Against Web-Based Identity Theft. » In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2004.

[41] Richard Clayton, Tyler Moore, and Nicolas Christin. « Concentrating Correctly on Cybercrime Concentration. » In: *Workshop on the Economics of Information Security*. 2015.

[42] Fred Cohen. « A note on the role of deception in information protection. » In: *Computers & Security* (1998).

[43] Fred Cohen et al. « The deception toolkit. » In: *Risks Digest* 19 (1998).

[44] Fred Cohen. *The Use of Deception Techniques: Honeypots and Decoys*. 2006.

[45] Fred Cohen and Deanna Koike. « Leading Attackers Through Attack Graphs with Deceptions The Attack Graph. » In: *Computers & Security* (2002).

[46] Fred Cohen, Dave Lambert, Charles Preston, Nina Berry, Corbin Stewart, and Eric Thomas. « A framework for deception. » In: *National Security Issues in Science, Law, and Technology* (2001).

[47] Fred Cohen, Irwin Marin, Jeanne Sappington, Corbin Stewart, and Eric Thomas. *Red teaming experiments with deception technologies*. 2001.

[48] Reuven Cohen. « The Cloud Hits the Mainstream: More than Half of U.S. Businesses Now Use Cloud Computing. » In: *Forbes magazine*. 2013.

[49] Marco Cova, Christopher Kruegel, and Giovanni Vigna. « There Is No Free Phish: An Analysis of "Free" and Live Phishing Kits. » In: *Workshop on Offensive Technologies (WOOT)*. 2008.

[50] Stephen Crane, Per Larsen, Stefan Brunthaler, and Michael Franz. « Booby trapping software. » In: *Proceedings of the 2013 workshop on New security paradigms workshop*. 2013.

[51] Emiliano De Cristofaro, Arik Friedman, Guillaume Jourjon, Mohamed Ali Kaafar, and M Zubair Shafiq. « Paying for likes?: Understanding facebook like fraud using honeypots. » In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. 2014.

[52] Cristiano De Faveri and Ana Moreira. « Designing Adaptive Deception Strategies. » In: *Software Quality, Reliability and Security Companion (QRS-C), 2016 IEEE International Conference on*. IEEE. 2016, pp. 77–84.

[53] Cristiano De Faveri, Ana Moreira, and Vasco Amaral. « Goal-driven deception tactics design. » In: *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE. 2016, pp. 264–275.

[54] Rachna Dhamija and J Doug Tygar. « The battle against phishing: Dynamic security skins. » In: *Symposium on Usable Privacy and Security*. 2005.

[55] Rachna Dhamija, J Doug Tygar, and Marti Hearst. « Why phishing works. » In: *SIGCHI conference on Human Factors in computing systems*. 2006.

[56] Brendan Dolan-Gavitt, Tim Leek, Michael Zhivich, Jonathon Giffin, and Wenke Lee. « Virtuoso: Narrowing the semantic gap in virtual machine introspection. » In: *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE. 2011, pp. 297–312.

[57] Ecommerce Foundation. *Global B2C E-commerce Report 2016*. https://www.ecommercewiki.org/wikis/www.ecommercewiki.org/images/5/56/Global_B2C_Ecommerce_Report_2016.pdf. 2016.

[58] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. « A survey on automated dynamic malware-analysis techniques and tools. » In: *ACM Computing Surveys (CSUR)* 44.2 (2012), p. 6.

[59] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. « You've been warned: an empirical study of the effectiveness of web browser phishing warnings. » In: *SIGCHI Conference on Human Factors in Computing Systems*. 2008.

[60] Methusela Cebrian Ferrer. *Zeus in the cloud*. http://community.ca.com/blogs/securityadvisor/archive/2009/12/09/zeus-in-the-cloud.aspx. 2009.

[61] Ian Fette, Norman Sadeh, and Anthony Tomasic. « Learning to detect phishing emails. » In: *World Wide Web (WWW) Conference*. 2007.

[62] Sujata Garera, Niels Provos, Monica Chew, and Aviel D Rubin. « A framework for detection and measurement of phishing attacks. » In: *Workshop on Recurring malcode*. 2007.

[63] Simson Garfinkel. « Anti-forensics: Techniques, detection and countermeasures. » In: 2007.

[64] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. « Terra: A virtual machine-based platform for trusted computing. » In: *ACM SIGOPS Operating Systems Review*. Vol. 37. 5. ACM. 2003, pp. 193–206.

[65] Dimitris Gavrilis, Ioannis Chatzis, and Evangelos Dermatas. « Flash crowd detection using decoy hyperlinks. » In: *International Conference on Networking, Sensing and Control (ICNSC)* (2007).

[66] Carl Gebhardt and Allan Tomlinson. « Secure virtual disk images for grid computing. » In: *Trusted Infrastructure Technologies Conference, 2008. APTC'08. Third Asia-Pacific*. IEEE. 2008, pp. 19–29.

[67] Han C Goh. *Intrusion deception in defense of computer systems*. Tech. rep. 2007.

[68] Guofei Gu, Junjie Zhang, and Wenke Lee. « BotSniffer: Detecting botnet command and control channels in network traffic. » In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2008.

[69] Swastik Gupta and Ponnurangam Kumaraguru. « Emerging phishing trends and effectiveness of the anti-phishing landing page. » In: *Electronic Crime Research (eCrime)*. 2014.

[70] Christopher N Gutierrez, Saurabh Bagchi, H Mohammed, and Jeff Avery. « Modeling Deception In Information Security As A Hypergame–A Primer. » In: *Proceedings of the 16th Annual Information Security Symposium*. CERIAS-Purdue University. 2015.

[71] Yasir Ahmed Hamza and Marwan Dahar Omar. « Cloud Computing Security: Abuse and Nefarious Use of Cloud Computing. » In: *International Journal of Computational Engineering Research*. Ed. by 03. 2013.

[72] Xiao Han, Nizar Kheir, and Davide Balzarotti. « The role of cloud services in malicious software: Trends and insights. » In: *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2015.

[73] Sharif Hassan and Ratan Guha. « Modelling of the State of Systems with Defensive Deception. » In: *Computational Science and Computational Intelligence (CSCI), 2016 International Conference on*. IEEE. 2016, pp. 1031–1036.

[74] Keqiang He, Alexis Fisher, Liang Wang, Aaron Gember, Aditya Akella, and Thomas Ristenpart. « Next stop, the cloud: understanding modern web service deployment in EC2 and Azure. » In: *ACM Internet Measurement Conference (IMC)*. 2013.

[75] Kristin E Heckman, Frank J Stech, Ben S Schmoker, and Roshan K Thomas. « Denial and Deception in Cyber Defense. » In: *Computer* 48.4 (2015), pp. 36–44.

[76] Kristin E Heckman, Michael J Walsh, Frank J Stech, Todd A O'boyle, Stephen R DiCato, and Audra F Herber. « Active cyber defense with denial and deception: A cyber-wargame experiment. » In: *computers & security* 37 (2013), pp. 72–77.

[77] Kelly Jackson Higgins. *Dropbox, WordPress Used As Cloud Cover In New APT Attacks*. http://www.darkreading.com/attacks-breaches/dropbox-wordpress-used-as-cloud-cover-in-new-apt-attacks/d/d-id/1140098?. [accessed 15-May-2014]. 2013.

[78] Jason Hong. « The state of phishing attacks. » In: *Communications of the ACM* (2012).

[79] Kenneth Houkjær, Kristian Torp, and Rico Wind. « Simple and realistic data generation. » In: *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment. 2006, pp. 1243–1246.

[80] AVTest Institute. *Malware statistics & Trends report*. http://www.av-test.org/en/statistics/malware/. 2014.

[81] Internet Live Stats. *Total number of Websites*. http://www.internetlivestats.com/total-number-of-websites/. 2017.

[82] Tom N Jagatic, Nathaniel A Johnson, Markus Jakobsson, and Filippo Menczer. « Social phishing. » In: *Communications of the ACM* (2007).

[83] Sushil Jajodia, VS Subrahmanian, Vipin Swarup, and Cliff Wang. *Cyber Deception: Building the Scientific Foundation*. 2016.

[84] Markus Jakobsson and Steven Myers. *Phishing and countermeasures: understanding the increasing problem of electronic identity theft*. Wiley-Interscience, 2006.

[85] Markus Jakobsson and Jacob Ratkiewicz. « Designing ethical phishing experiments: a study of (ROT13) rOnl query features. » In: *World Wide Web (WWW) Conference*. 2006.

[86] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. « Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. » In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, pp. 128–138.

[87] Priyanka Jogdand and Puja Padiya. « Survey of different IDS using honeytoken based techniques to mitigate cyber threats. » In: *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on*. IEEE. 2016, pp. 802–807.

[88] Yogesh Joshi, Samir Saklikar, Debabrata Das, and Subir Saha. « PhishGuard: a browser plug-in for protection from phishing. » In: *Internet Multimedia Services Architecture and Applications (IMSAA)*. 2008.

[89] Ari Juels and Thomas Ristenpart. « Honey encryption: Security beyond the brute-force bound. » In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2014.

[90] Ari Juels and Ronald L Rivest. « Honeywords: Making password-cracking detectable. » In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013.

[91] Ari Juels and Cornell Tech. « A Bodyguard of Lies : The Use of Honey Objects in Information Security. » In: *Proceedings of the 19th ACM symposium on Access control models and technologies - SACMAT '14* (2014).

[92] Donald P Julian. « Delaying Type Response for Use By Software Decoys. » PhD thesis. 2002.

[93] Parisa Kaghazgaran and Hassan Takabi. « Toward an Insider Threat Detection Framework Using Honey Permissions. » In: *Journal of Internet Services and Information Security (JISIS)* (2015).

[94] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. « Hulk: Eliciting Malicious Behavior in Browser Extensions. » In: *USENIX Security*. 2014.

[95] Kaspersky. *Top 7 Cyberthreats to Watch Out for in 2015-2016*. http://usa.kaspersky.com/internet-security-center/threats/top-7-cyberthreats. 2015.

[96] Constantine Katsinis and Brijesh Kumar. « A Security Mechanism for Web Servers Based on Deception. » In: *Proceedings of the The 2012 International Conference on Internet Computing - ICOMP'12* (2012).

[97] Constantine Katsinis, Brijesh Kumar, Security Technology, and Rapidsoft Systems. « A Framework for Intrusion Deception on Web Servers. » In: *International Conference on Internet Computing, ICOMP'13*. 2013.

[98] Gene H Kim and Eugene H Spafford. « Experiences with tripwire: Using integrity checkers for intrusion detection. » In: *System Administration, Networking, and Security Conference*. 1994.

[99] Gene H Kim and Eugene H Spafford. « The design and implementation of tripwire: A file system integrity checker. » In: *ACM Conference on Computer and Communications Security*. ACM. 1994.

[100] Ryan KL Ko, Peter Jagadpramana, Miranda Mowbray, Siani Pearson, Markus Kirchberg, Qianhui Liang, and Bu Sung Lee. « TrustCloud: A framework for accountability and trust in cloud computing. » In: *Services (SERVICES), 2011 IEEE World Congress on*. IEEE. 2011, pp. 584–588.

[101] Georgios Kontaxis, Michalis Polychronakis, and Angelos D Keromytis. « Computational Decoys for Cloud Security. » In: *Secure Cloud Computing*. 2014.

[102] Ponnurangam Kumaraguru, Lorrie Faith Cranor, and Laura Mather. « Anti-phishing landing page: Turning a 404 into a teachable moment for end users. » In: *Conference on Email and Anti-Spam (CEAS)*. 2009.

[103] Ponnurangam Kumaraguru, Yong Rhee, Steve Sheng, Sharique Hasan, Alessandro Acquisti, Lorrie Faith Cranor, and Jason Hong. « Getting users to pay attention to anti-phishing education: evaluation of retention and transfer. » In: *Anti-phishing working groups annual eCrime researchers summit*. 2007.

[104] Ponnurangam Kumaraguru, Steve Sheng, Alessandro Acquisti, Lorrie Faith Cranor, and Jason Hong. « Teaching Johnny not to fall for phish. » In: *ACM Transactions on Internet Technology (TOIT)* (2010).

[105] Spitzner Lance. *The Value of Honeypots, Part One: Definitions and Values of Honeypots*. https://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots. 2001.

[106] Per Larsen, Andrei Homescu, Stefan Brunthaler, and Michael Franz. « SoK: Automated software diversity. » In: *Security and Privacy (SP), 2014 IEEE Symposium on*. 2014.

[107] Martin Lazarov, Jeremiah Onaolapo, and Gianluca Stringhini. « Honey Sheets: What Happens To Leaked Google Spreadsheets? » In: *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*. 2016.

[108] Martin Lazarov, Jeremiah Onaolapo, and Gianluca Stringhini. « Honey Sheets: What Happens To Leaked Google Spreadsheets ? » In: *Usenix Cset* (2016).

[109] Erwan Le Malécot. « MitiBox: camouflage and deception for network scan mitigation. » In: *Proceedings of the 4th USENIX Workshop on Hot Topics in Security (HotSec)*. 2009.

[110] Anh Le, Athina Markopoulou, and Michalis Faloutsos. « Phishdef: Url names say it all. » In: *Conference on Computer Communications (INFOCOM)*. 2011.

[111] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. « A Lustrum of Malware Network Communication: Evolution and Insights. » In: *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE. 2017, pp. 788–804.

[112] Xiaowei Li and Yuan Xue. « A Survey on Server-side Approaches to Securing Web Applications. » In: *ACM Comput. Surv.* (2014).

[113] Zhou Li, Sumayah Alrwais, Yinglian Xie, Fang Yu, and XiaoFeng Wang. « Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. » In: *Security and Privacy (S&P)*. 2013.

[114] Xiaojing Liao, Sumayah Alrwais, Kan Yuan, Luyi Xing, XiaoFeng Wang, Shuang Hao, and Raheem Beyah. « Lurking Malice in the Cloud: Understanding and Detecting Cloud Repository as a Malicious Service. » In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 1541–1552.

[115] Xiaojing Liao, Chang Liu, Damon McCoy, Elaine Shi, Shuang Hao, and Raheem Beyah. « Characterizing long-tail SEO spam on cloud web hosting services. » In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 321–332.

[116] Tom Liston. « LaBrea:"Sticky" Honeypot and IDS. » In: (2001).

[117] Bingshuang Liu, Zhaoyang Liu, Jianyu Zhang, Tao Wei, and Wei Zou. « How many eyes are spying on your shared folders? » In: *Proceedings of the 2012 ACM workshop on Privacy in the electronic society*. 2012.

[118] Rafal Los, Dave Shackleford, and Bryan Sullivan. « The Notorious Nine Cloud Computing Top Threats in 2013. » In: *Cloud Security Alliance*. 2013.

[119] Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. « On the effectiveness of techniques to detect phishing sites. » In: *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2007.

[120] D Kevin McGrath and Minaxi Gupta. « Behind Phishing: An Examination of Phisher Modi Operandi. » In: *Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. 2008.

[121] Craig M. McRae and Rayford B. Vaughn. « Phighting the phisher: Using Web bugs and honeytokens to investigate the source of phishing attacks. » In: *Proceedings of the Annual Hawaii International Conference on System Sciences* (2007).

[122] Eric Medvet, Engin Kirda, and Christopher Kruegel. « Visual-similarity-based phishing detection. » In: *Security and Privacy in Communication Netowrks Conference*. 2008.

[123] B Michael, M Auguston, N Rowe, and R Riehle. « Software Decoys: Intrusion Detection and Countermeasures. » In: *Proceedings of the IEEE Workshop on Information Assurance*. 2002.

[124] Miniwatts Marketing Group. *Internet Usage and World Population Statistics*. http://www.internetworldstats.com/stats.htm. 2017.

[125] Tyler Moore and Richard Clayton. « Examining the impact of website take-down on phishing. » In: *Anti-phishing working groups annual eCrime researchers summit*. 2007.

[126] Tyler Moore and Richard Clayton. « Evil searching: Compromise and recompromise of internet hosts for phishing. » In: *Financial Cryptography and Data Security*. 2009.

[127] Tyler Moore and Richard Clayton. « Discovering phishing dropboxes using email metadata. » In: *eCrime Researchers Summit (eCrime)*. 2012.

[128] Tyler Moore and Richard Clayton. « Ethical dilemmas in takedown research. » In: *Financial Cryptography and Data Security*. 2012.

[129] Tyler Moore, Richard Clayton, and Henry Stern. « Temporal Correlations between Spam and Phishing Websites. » In: *Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. 2009.

[130] Sherry Murphy, Todd McDonald, and Robert Mills. « An Application of Deception in Cyberspace: Operating System Obfuscation. » In: *International Conference on Information Warfare and Security*. 2010.

[131] MushMush Foundation. *Glastopf*. https://github.com/mushorg/glastopf/blob/master/glastopf/requests.xml. 2017.

[132] Antonio Nappa, M. Zubair Rafique, and Juan Caballero. « Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting. » In: *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. Berlin, Germany, July 2013.

[133] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. « A Survey on Honeypot Software and Data Analysis. » In: *arXiv preprint arXiv:1608.06249* (2016).

[134] Jose Nazario. « PhoneyC: A Virtual Client Honeypot. » In: *LEET* (2009).

[135] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. « Automatically hardening web applications using precise tainting. » In: *Security and Privacy in the Age of Ubiquitous Computing* (2005), pp. 295–307.

[136] Nick Nikiforakis, Marco Balduzzi, Steven Van Acker, Wouter Joosen, and Davide Balzarotti. « Exposing the Lack of Privacy in File Hosting Services. » In: *LEET*. 2011.

[137] ADAM NOSSITER, David E. Sanger, and Nicole Perlroth. *Hackers Came, but the French Were Prepared*. https://www.nytimes.com/2017/05/09/world/europe/hackers-came-but-the-french-were-prepared.html. 2017.

[138] Jeremiah Onaolapo, Enrico Mariconti, and Gianluca Stringhini. « What Happens After You Are Pwnd: Understanding the Use of Leaked Webmail Credentials in the Wild. » In: *ACM SIGCOMM Internet Measurement Conference*. 2016.

[139] Kaan Onarlioglu, U Ozan Yilmaz, Davide Balzarotti, and Engin Kirda. « Insights into user behavior in dealing with internet attacks. » In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2012.

[140] OWASP. *Testing Guide v4*. https://www.owasp.org/images/1/19/OTGv4.pdf. 2014.

[141] OWASP. *AppSensor Project Guide*. https://www.owasp.org/index.php/File:Owasp-appsensor-guide-v2.pdf. 2015.

[142] Keshnee Padayachee. « Aspectising honeytokens to contain the insider threat. » In: *IET Information Security* 9.4 (2014), pp. 240–247.

[143] Augusto Paes de Barros. *RES: Protocol Anomaly Detection IDS - Honeypots*. http://seclists.org/focus-ids/2003/Feb/95. 2003.

[144] Ying Pan and Xuhua Ding. « Anomaly based web phishing page detection. » In: *Annual Computer Security Applications Conference (ACSAC)*. 2006.

[145] Younghee Park and Salvatore J Stolfo. « Software decoys for insider threat. » In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. 2012.

[146] Roberto Perdisci, Wenke Lee, and Nick Feamster. « Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. » In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2010.

[147] AB Robert Petrunić. « Honeytokens as active defense. » In: *38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2015 - Proceedings* (2015).

[148] Christy Pettey and Laurence Goasduff. *Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017*. http://www.gartner.com/newsroom/id/3616417. 2017.

[149] Lawrence Pingree. « Emerging Technology Analysis: Deception Techniques and Technologies Create Security Technology Business Opportunities. » In: *Gartner, Inc* (2015).

[150] Niels Provos et al. « A Virtual Honeypot Framework. » In: *USENIX Security Symposium*. 2004.

[151] Greg Reimer. *Hoxy*. http://greim.github.io/hoxy/. 2015.

[152] Ryan Riley, Xuxian Jiang, and Dongyan Xu. « Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. » In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2008, pp. 1–20.

[153] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. « Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. » In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 199–212.

[154] Neil Rowe. « Planning cost-effective deceptive resource denial in defense to cyber-attacks. » In: *Proceedings of the 2nd International Conference on Information Warfare & Security*. 2007.

[155] Neil C Rowe. « Designing good deceptions in defense of information systems. » In: *Computer Security Applications Conference*. IEEE. 2004.

[156] Neil C Rowe. « Measuring the effectiveness of honeypot counter-counterdeception. » In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS)*. Vol. 6. IEEE. 2006, pp. 129c–129c.

[157] Neil C Rowe. « Deception in Defense of Computer Systems from Cyber Attack. » In: *Cyber Warfare and Cyber Terrorism* (2008).

[158] Neil C Rowe, Binh T Duong, and E Custy. « Fake Honeypots: A Defensive Tactic for Cyberspace. » In: *IEEE Information Assurance Workshop*. 2006.

[159] Neil C Rowe and Hy S Rothstein. « Two taxonomies of deception for attacks on information systems. » In: (2004).

[160] Neil C Rowe and Julian Rrushi. *Introduction to Cyberdeception*. Springer, 2016.

[161] Julian L. Rrushi. « An exploration of defensive deception in industrial communication networks. » In: *International Journal of Critical Infrastructure Protection* (2011).

[162] Julian L Rrushi. « NIC displays to thwart malware attacks mounted from within the OS. » In: *Computers & Security* (2016).

[163] Karen Scarfone and Peter Mell. « Guide to intrusion detection and prevention systems (idps). » In: *NIST special publication* (2007).

[164] Asaf Shabtai, Maya Bercovitch, Lior Rokach, Ya'akov (Kobi) Gal, Yuval Elovici, and Erez Shmueli. « Behavioral Study of Users When Interacting with Active Honeytokens. » In: *ACM Trans. Inf. Syst. Secur.* (2016).

[165] Steve Sheng, Mandy Holbrook, Ponnurangam Kumaraguru, Lorrie Faith Cranor, and Julie Downs. « Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions. » In: *SIGCHI Conference on Human Factors in Computing Systems*. 2010.

[166] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Faith Cranor, Jason Hong, and Chengshan Zhang. « An empirical analysis of phishing blacklists. » In: *Conference on Email and Anti-Spam (CEAS)*. 2009.

[167] Leslie Shing. « An improved tarpit for network deception. » MA thesis. Monterey, California: Naval Postgraduate School, 2016.

[168] Matthew Smart, G Robert Malan, and Farnam Jahanian. « Defeating TCP/IP Stack Fingerprinting. » In: *Usenix Security Symposium*. 2000.

[169] Solutionary. *Security Engineering Research Team (SERT) Quarterly Threat Intelligence Report*. http://www.solutionary.com/_assets/pdf/research/sert-q4-2013-threat-intelligence.pdf. [accessed 15-May-2014]. 2014.

[170] Lance Spitzner. « Honeypots : Catching the Insider Threat. » In: *Annual Computer Security Applications Conference*. 2003.

[171] Lance Spitzner. *Honeytokens: The other honeypot*. http://www.securityfocus.com/infocus/1713. 2003.

[172] Lance Spitzner. « The honeynet project: Trapping the hackers. » In: *IEEE Security & Privacy* (2003).

[173] Cliff Stoll. *The cuckoo's egg: tracking a spy through the maze of computer espionage*. 1989.

[174] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. « Detecting spammers on social networks. » In: *Proceedings of the 26th annual computer security applications conference*. 2010.

[175] Subashini Subashini and Veeraruna Kavitha. « A survey on security issues in service delivery models of cloud computing. » In: *Journal of network and computer applications* 34.1 (2011), pp. 1–11.

[176] Symantec. *Internet Security Threat Report*. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf. 2016.

[177] Samaneh Tajalizadehkhoob, Carlos Gañán, Arman Noroozian, and Michel van Eeten. « The Role of Hosting Providers in Fighting Command and Control Infrastructure of Financial Malware. » In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM. 2017, pp. 575–586.

[178] The Dino. *Best Pay-Per-Install Affiliate Program Reviews*. http://pay-per-install.com. 2007.

[179] Samuel T Trassare. « A technique for presenting a deceptive dynamic network topology. » In: (2013).

[180] Trusteer. *Measuring the Effectiveness of In-the-Wild Phishing Attacks*. https://web.archive.org/web/20120324061250/http://www.trusteer.com/sites/default/files/Phishing-Statistics-Dec-2009-FIN.pdf. 2009.

[181] UK Government Digital Service. *How digital and technology transformation saved £1.7bn last year*. https://gds.blog.gov.uk/2015/10/23/how-digital-and-technology-transformation-saved-1-7bn-last-year/. 2015.

[182] Roman Unuchek. *GCM in malicious attachments*. http://www.securelist.com/en/blog/8113/GCM_in_malicious_attachments. [accessed 15-May-2014]. 2013.

[183] Vincent E Urias, William MS Stout, and Han W Lin. « Gathering threat intelligence through computer network deception. » In: *Technologies for Homeland Security (HST), 2016 IEEE Symposium on*. 2016.

[184] Luis M Vaquero, Luis Rodero-Merino, and Daniel Morán. « Locking the sky: a survey on IaaS cloud security. » In: *Computing* 91.1 (2011), pp. 93–118.

[185] Nikos Virvilis, Bart Vanautgaerden, and Oscar Serrano Serrano. « Changing the game: The art of deceiving sophisticated attackers. » In: *International Conference on Cyber Conflict, CYCON* (2014).

[186] Jonathan Voris, Jill Jermyn, Nathaniel Boggs, and Salvatore Stolfo. « Fox in the Trap : Thwarting Masqueraders via Automated Decoy Document Deployment. » In: *Eurosec* (2015).

[187] Jonathan Voris, Jill Jermyn, Angelos D Keromytis, and Salvatore J Stolfo. « Bait and snitch: Defending computer systems with decoys. » In: *Proceedings of the cyber infrastructure protection conference, Strategic Studies Institute, September*. 2013.

[188] Liang Wang, Antonio Nappa, Juan Caballero, Thomas Ristenpart, and Aditya Akella. « WhoWas: A Platform for Measuring Web Deployments on IaaS Clouds. » In: *ACM Internet Measurement Conference (IMC)*. 2014.

[189] Wei Wang, Jeffrey Bickford, Ilona Murynets, Ramesh Subbaraman, Andrea G. Forte, and Gokul Singaraju. « Detecting Targeted Attacks By Multilayer Deception. » In: *Journal of Cyber Security and Mobility* (2013).

[190] David Watson, Thorsten Holz, and Sven Mueller. *Know your enemy: Phishing*. https://www.honeynet.org/papers/phishing. 2005.

[191] Steve Webb, James Caverlee, and Calton Pu. « Social Honeypots: Making Friends With A Spammer Near You. » In: *CEAS*. 2008.

[192] Jonathan White. « Creating personally identifiable honeytokens. » In: *Innovations and Advances in Computer Sciences and Engineering*. Springer, 2010, pp. 227–232.

[193] Ben Whitham. « Automating the generation of fake documents to detect network intruders. » In: *International Journal of Cyber-Security and Digital Forensics (IJCSDF)* 2.1 (2013), pp. 103–118.

[194] Ben Whitham. « Automating the Generation of Enticing Text Content for High-Interaction Honeyfiles. » In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017.

[195] Colin Whittaker, Brian Ryner, and Marria Nazif. « Large-Scale Automatic Classification of Phishing Pages. » In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2010.

[196] Min Wu, Robert C Miller, and Simson L Garfinkel. « Do security toolbars actually prevent phishing attacks? » In: *SIGCHI conference on Human Factors in computing systems*. 2006.

[197] Guang Xiang and Jason I Hong. « A hybrid phish detection approach by identity discovery and keywords retrieval. » In: *World Wide Web (WWW) conference*. 2009.

[198] J. Yuill, M. Zappe, D. Denning, and F. Feer. « Honeyfiles: deceptive files for intrusion detection. » In: *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* (2004).

[199] James Joseph Yuill. « Defensive Computer-security Deception Operations: Processes, Principles and Techniques. » PhD thesis. 2006.

[200] Jim Yuill, Dorothy E Denning, and Fred Feer. *Using deception to hide things from hackers: Processes, principles, and techniques*. Tech. rep. DTIC Document, 2006.

[201] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. « Cross-VM side channels and their use to extract private keys. » In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 305–316.

[202] Yue Zhang, Serge Egelman, Lorrie Cranor, and Jason Hong. « Phinding phish: Evaluating anti-phishing tools. » In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2007.

[203] Yue Zhang, Jason I Hong, and Lorrie F Cranor. « Cantina: a content-based approach to detecting phishing web sites. » In: *World Wide Web (WWW) Conference*. 2007.