IMPROVING DETECTION AND ANNOTATION OF MALWARE DOWNLOADS AND INFECTIONS THROUGH DEEP PACKET INSPECTION

A Thesis
Presented to
The Academic Faculty

by

Terry L. Nelms

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
May 2016

IMPROVING DETECTION AND ANNOTATION OF MALWARE DOWNLOADS AND INFECTIONS THROUGH DEEP PACKET INSPECTION

Approved by:

Professor Mustaque Ahamad, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Roberto Perdici, Co-Advisor
School of Computer Science
Georgia Institute of Technology

Professor Manos Antonakakis
School of ECE
Georgia Institute of Technology

Professor Wenke Lee
School of Computer Science
Georgia Institute of Technology

Dr. Josyula R. Rao
Director, Security Research
IBM Research

Date Approved: 4 December 2015

To Nicole, Britton and Keaton

for the distractions that make life worth living.

ACKNOWLEDGEMENTS

Research is impossible to do alone. The breakthroughs and solutions only come after many discussions and debates with others. I was fortunate enough to work with a group of smart and talented people; this thesis is only possible because of their help.

First, I want to thank my advisor Mustaque for having faith in me from the beginning. I was not a traditional PhD student because I was working full-time and had a family, but he took a chance on me anyway. His guidance and feedback throughout the PhD process was invaluable. My co-advisor Roberto challenged me to become a better researcher and in the process taught me how to do it. The quality of the work in the thesis would have been lower without his help. I and the readers of this thesis thank you.

Manos, collaborator and PhD committee member, helped identify weaknesses in my research and provided insight on how to improve it. Thank you for your constructive advice and all the time you spent reviewing and discussing research in this thesis. Also, many thanks to the other PhD committee members Wenke and J.R. for their valuable feedback and guidance throughout this process.

Having access to real world networks and data is an important part of research. It was critical for performing the studies and validating the systems described in this thesis. Damballa, a research oriented company, provided this access. Many people at Damballa helped with my research, but I specifically want to thank Tyler Graph, Eric Davidson, Marshall Vandegrift, Drew Hobson, Joseph Ward, Jeremy Demar and Stephen Newman.

I first became interested in research while at IBM. I want to thank Hubertus Franke and Steve Hunter of IBM Research for providing encouragement and advice

on pursuing a PhD. It was because of discussions with them that I made the decision to apply to the PhD program.

Lastly, I want to thank my wife Nicole and twin boys Britton and Keaton. I spent many weekends and late nights working throughout the PhD process. They were always supportive and understanding. Without their backing this thesis would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Malware continues to be one of the primary tools employed by attackers. It is used in attacks ranging from click fraud to nation state espionage. Malware infects hosts over the network through drive-by downloads and social engineering. These infected hosts communicate with remote command and control (C&C) servers to perform tasks and exfiltrate data. Malware's reliance on the network provides an opportunity for the detection and annotation of malicious communication.

This thesis contains four primary contributions: First, we design and implement a novel incident investigation system, named WebWitness. It automatically traces back and labels the sequence of events (e.g., visited web pages) preceding malware downloads to highlight how users reach attack pages on the web; providing a better understanding of current attack trends and aiding in the development of more effective defenses. Second, we conduct the first systematic study of modern web based social engineering malware download attacks. From this study we develop a taxonomy for classifying social engineering downloads and use it to measure attack properties. From these measurements we show that is it possible to detect the majority of social engineering downloads using features from the download path. Third, we design and implement ExecScent, a novel system for mining new malware C&C domains from live networks. ExecScent automatically learns C&C traffic models that can adapt to the deployment network's traffic. This adaptive approach allows us to greatly reduce the false positives while maintaining a high number of true positives. Lastly, we develop a new packet scheduling algorithm for deep packet inspection that maximizes throughput by optimizing for cache affinity. By scheduling for cache affinity, we are able to deploy our systems on multi-gigabit networks.

CHAPTER I

INTRODUCTION

Malware continues to be a significant threat to Internet security despite all the resources allocated to combat it [4, 5]. It is a critical component in many of the most costly attacks on organizations such as information stealing and extortion. For instance, Cryptolocker ransomware made an estimated 30 million dollars in less than 100 days for attackers [3]. A number of other attacks in recent years have leveraged malware to have multimillion dollar paydays such as ZeroAccess [16] and Carberp [9]. In addition to monetary gains, malware is often employed in nation state cyber espionage and in targeted attacks that are motived by economic, political or military interests [10, 15]. With such high stakes, defending hosts and networks against malware continues to be a top priority.

The majority of malware infections begin with a remote malicious executable download. Two common infection vectors are drive-by downloads and social engineering [97]. Drive-by downloads exploit an unpatched vulnerability in the browser or plug-in. The exploit gains control of the application by pointing its execution at code (i.e., shellcode) controlled by the attacker. The shellcode then downloads and executes the malware. Often, a drive-by download attack will go unnoticed by the victim and they will be unaware of the compromise. Intrusion detection and prevention systems (IDS/IPS) are often deployed to identify and stop software exploitation, but often fail due to obfuscated exploit code and permutations in the exploitation process.

Social engineering is also a common infection vector. However, the target of the attack is the user, not software. The attacker uses persuasion [43] and deception [127]

techniques to convince the the user to download and install malware. One benefit for the attacker in using social engineering is that his attack can be successful even on systems that have a limited attack surface and no known exploitable vulnerabilities. Furthermore, social engineering attacks are less likely to be identified by modern detection systems since their focus is on exploitation [14, 101].

Blacklisting is a popular technique that is used to detect and block malicious domains that are associated with exploitation, social engineering and malware downloads [6]. The primary issue with blacklists is that they are always out-of-date. For example, in the case of drive-by downloads, the domains that are typically blacklisted are the ones associated with the exploit and malware executable. However, these domains are typically only in use for a day or less. So by the time they are blacklisted, the attackers are already using new domains to host their exploit kit and malware [97].

Executable downloads are often examined at the network and host level using detection systems that leverage antivirus (AV) and sandboxing. AV, like domain blacklists, tend to lag behind the threat and not detect new malware on the day it is downloaded. This is not due to the lack of code reuse because it is common practice for malware [69, 75]. In fact, it is not uncommon for the source code of successful malware to be sold or leaked on underground forums, and to be reused by other attackers [55]. However, even though the code is old, it is easy for attackers to create new (polymorphic) malware releases using packers and crypters to mutate the executable for the purpose of bypassing AV [1]. Thus, little work is required of the attacker to create new malware that appears to have never been observed.

Sandboxes provide a controlled environment to run executables. The outputs of a sandbox are system and network traces. This information is used by analysts and automated systems to identify malware. However, malware often uses anti-sandbox technique such as virtual machine (VM) detection and timing measurements to detect

the sandbox and proceed with a benign code path that has no suspicious behavior [50].

Once downloaded the malware is executed. The vast majority of modern malware, especially botnets, have a network component. After execution the malware communicates over the network to a command and control (C&C) server for the purpose of monetizing (e.g., information stealing) the infection. The most popular protocols used by today's malware are DNS and HTTP [115, 133]. DNS is used because it provides a level of indirection between the malware client and the physical location of the C&C server. This provides flexibility for the attacker allowing them to easily move their hosting to new networks without the need to update the malware clients.

HTTP is a popular C&C protocol for two reasons. First, it is allowed out of most networks. Many enterprises employ strict egress filtering only allowing DNS and HTTP out of their networks. In fact, when there is strict egress filtering, typically HTTP is only allowed out through an HTTP proxy and DNS through the local recursive. All other outbound traffic is blocked at the firewall. Therefore, for malware to communicate from these networks they must use HTTP or DNS. In addition, for most networks, HTTP is by far the most common protocol used allowing C&C over HTTP blends into the background traffic making it harder to identify as suspicious.

The most common techniques used to detect C&C communication are domain blacklist and intrusion detection (IDS) signatures [14, 101]. C&C blacklists, like malware download blacklists, are quickly out-of-date [30]. Also, identifying new C&C domains normally requires domain extraction from sandbox execution or reverse engineering the executable. Even if the malware runs in a sandbox, many will visit benign domains in addition to the C&C domain making automatic blacklisting more difficult. IDS signatures tend to be viable longer than blacklists, but are typically manually created by analysts. In addition, strings in the protocol signature may be commonly found in benign traffic on some deployment networks producing a lot of false positives.

Improving detection and annotation (e.g., cause of the download) of malware begins by examining how the user came to download malware. There is a large body of work on detecting drive-by downloads [44,46,83,92,113,132] and a few efforts have examined social engineering attacks and techniques [31,111,125]. Yet, little attention has been dedicated to investigating and categorizing the web browsing paths followed by users before they reach the web pages from which the attack starts to unfold.

Focusing on what happened prior to a malicious download provides a better understanding of current attack trends and shows how users become victims. Web browsing paths can be used to study the tactics and techniques of attackers. For instance, there is a pressing need for a comprehensive study of social engineering malware downloads to aid in the development of training programs that educate end users. Web browsing paths provide a means for performing such a study. In addition, insights gained from them can be leveraged to devise more effective defenses for both drive-by downloads and social engineering attacks.

After the malware is downloaded and successfully executed, the host is infected. Detecting infected hosts by observing their network activity is challenging because domain names and IP addresses used for C&C change frequently to stay ahead of blacklists. Also, there is nothing unique about the protocols used for C&C. However, the fundamental structure of the communication (i.e., language) between the malware and the C&C server remains constant for much longer periods of time [96]. Once there are active clients the protocol structure (e.g., data types, encodings, lengths and ordering) is much harder for the attacker to modify. In addition, code reuse is common so different botnets that use the same malware family will often have similar protocol structures. But, identifying network communication with protocol structure similar to that of malware is not enough for detection. It must also be unique on the deployment network (i.e., not commonly found in benign communication) for there to be confidence in the match.

Analyzing malware downloads and C&C protocol communication at the network level requires deep packet inspection (DPI). DPI provides on-the-fly TCP flow reconstruction allowing for the analysis of high level protocols such as HTTP [56]. The analysis performed by DPI is processor intensive and performing DPI on large complex networks at multi-gigabit speeds is challenging [95]. Yet, it is these networks that are often the target of malware infections. Multithreading DPI provides additional CPU resources, but requires packet scheduling to divide the work. Packet schedulers can be optimized using different techniques such as load balancing or cache affinity. These tradeoffs must be understood before an optimal packet scheduling algorithm can be selected.

1.1 Thesis Statement

The goal of this thesis is to study real malware downloads and infections at the network level using DPI on live traffic in order to design and build systems that aid in their detection and annotation. Based on our analysis of network communication generated by malicious applications we propose the following thesis statement.

Analyzing and modeling the network behavior of malware using DPI improves our understanding of malware downloads and infections. Insights gained from this process can be leveraged to improve detection, annotation and network defenses.

1.2 Thesis Overview

Modern malware utilizes the network for both the initial compromise and post infection management. The initial compromise often occurs through a web browser and is typically due to social engineering or the exploitation of a software vulnerability. The web browser remains a popular infection vector because it is widely used to communicate with other hosts, it provides the resources required for social engineering and vulnerabilities are plentiful. Unlike prior research that focuses on the domains

and websites that deliver the attack, we examine how and why a user came to download malware. To better understand how users get infected and improve defenses, we investigate the following research problems:

- Identify the sequence of web pages visited by a user that led to a malicious download with only network visibility; i.e., reconstruct the download path from observed HTTP transactions.
- Determine the reason for a malicious download by using features that can be extracted from the download path.
- Leverage the download paths to better understand current attack trends and develop more effective defenses.

Once a host is compromised, malware communicates with its command and control (C&C) server for infection management. Hosts are often compromised for months or years before they are discovered. Thus, effective network defenses against the initial infection are not enough because most of the deployment networks will already have infected hosts that need to be identified and hosts can become infected via other mediums such as USB. Furthermore, trends like bring your own device result in compromised hosts from unprotected networks joining in the future. To detect compromised hosts at the network level, we explore the following research problems:

- Learn the structure of malware communication from packet captures of malware executed in a sandbox or from an infected host.
- Detect infected hosts and new C&C domains by comparing the observed protocol structure to learned malicious communication.
- Adapt to the deployment network's communication profile to provide a better tradeoff between true and false positives.

Detection, annotation and the defense of malware at the network level requires significant processing resources. In order to deploy systems that address the above research problems while scaling to large enterprise networks, it is necessary to multi-thread deep packet inspection (DPI). This requires packets to be scheduled across DPI threads. We examine the following research problems to optimize packet scheduling for DPI performance:

- Determine if maximizing workload balance or cache affinity provides the best performance.
- Identify the packet scheduling performance tradeoffs in regards to throughput and latency.

Figure 1 shows a high level view of the systems developed and how they work together to address the above research problems. Packets from a live network are scheduled based on cache affinity for deep packet inspection. Reconstructed HTTP transactions are then processed by both WebWitness and ExecScent. The focus of WebWitness is on the initial infection. When it identifies an executable download the web path is extracted and the cause is labeled as drive-by, social engineering or update. WebWitness requires an oracle to determine if a download is malicious. For malicious drive-by downloads the web paths are furthered processed to identify the important web pages on the path to be blocked. WebSentry extends the capabilities of WebWitness by detecting, without the need of an oracle, social engineering downloads. Using WebSentry we further our analysis of malicious downloads as well as provide a defense against an understudied infection method. Lastly, ExecScent detects already infected hosts by comparing the protocol structure of their network communication to that of known malware.

1.3 Contributions

This dissertation makes the following contributions:

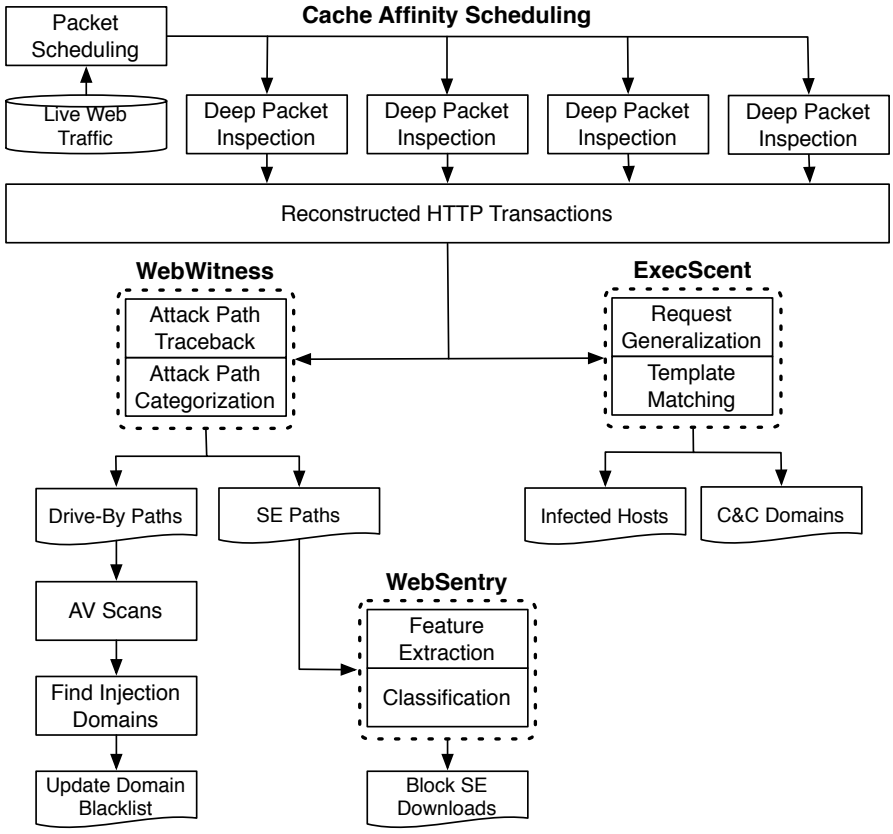

Figure 1: Overview of the systems developed to address the research problems examined in this thesis. The systems include WebWitness, WebSentry, ExecScent and a cache affinity packet scheduler.

- *Perform the first study of the web paths followed by real network users who eventually fall victim to different types of malware downloads, including social engineering and drive-by downloads:* Through this investigation, we provide quantitative information on attack scenarios that have been previously explained only anecdotally or through limited case studies.

- *Design and implement WebWitness, a system that enables the continuous collection and study of web paths leading to malware download attacks:* WebWitness can automatically trace back and categorize in-the-wild malware downloads. We show that this information can be leveraged to design more effective defenses against future malware download attacks.

- *Conduct the first systematic study of modern web based social engineering (SE) malware download attacks:* Our analysis of hundreds of SE malware attack instances reveals that most attacks are enabled by malicious online advertisement served through a handful of "low tier" ad networks. Also, we find that the most common types of SE malware attacks include fake updates for Adobe Flash and Java. However, fake antivirus (Fake AV), which has been a popular and effective infection vector in the recent past, represent less than 1% of all SE malware downloads observed in the wild.

- *Create a categorization system for labeling social engineering malware download attacks:* The categorization system expresses how attackers typically gain the user's attention and the most common types of deception and persuasion tactics used to trick victims into downloading malicious applications. It enables us to abstract the specific tactics used by attackers and focus our end user training and defenses at the deception and persuasion techniques currently used in the wild.

- *Develop WebSentry, a network defense system that aims to detect web-based SE*

malware attacks in real time: WebSentry leverages the download paths provided by WebWitness to extract features from the download path. We show that our system is able to accurately detect SE malware download attempts with 91% true positives and only 0.5% false positives.

- *Design and implement ExecScent, a novel system for identifying infected hosts by analyzing their network communication on live networks:* ExecScent automatically learns C&C traffic models that can adapt to the deployment network's traffic. This adaptive approach allows us to greatly reduce the false positives while maintaining a high number of true positives. To the best of our knowledge, ExecScent is the first system to use this type of adaptive C&C traffic models.

- *Develop a packet scheduling algorithm that maximizes DPI throughput by optimizing for cache affinity:* We design and implement two packet scheduling algorithms. One designed to maximize workload balance and the other cache affinity. We compare the performance of both algorithms to direct hash, a common packet scheduling algorithm used by industry. The algorithms are evaluated on three different types of network loads. Our evaluation results show that our cache affinity packet scheduling algorithm provides the highest DPI throughput for all workloads, besting the other two algorithms by almost 40%.

1.4 Research Impact

ExecScent is currently deployed as a feature in Damballa's Failsafe product line as the Request Profiler. Also, Damballa runs ExecScent daily on the network commutation of malware samples executed in a sandbox where it provides the largest feed of new C&C domains to their threat research staff. WebWitness is on Damballa's Failsafe product roadmap where it will be known as Traceback. Damballa is also considering a standalone offering that could be deployed at web proxies using ICAP. Patents have

been filed for both ExecScent and WebWitness. As for WebSentry, we are in the process of filing a patent and Damballa plans to include it in an update to Traceback. Our packet scheduling research greatly influenced the design of IBM's multithreaded version of the Protocol Analysis Module (PAM), which was the first DPI engine to reach 20 gigabit per second performance in a software only architecture.

1.5 Thesis Outline

In Chapter 2, we describe related work and explain how the contributions in this thesis relate to this work and advance our understanding of malware infections.

In Chapter 3, we discuss WebWitness, a system that provides context to malware downloads. Section 3.2 reports the results of a large study of in-the-wild malware downloads captured on a live academic network. This study describes the challenges of download path traceback as well as potential features for determining the cause of a download. Next, Section 3.3 we present the WebWitness system. The details of both the Attack Path Traceback (ATC) and Malware Download Defense (MDD) Modules are provided. In Section 3.4 we evaluate WebWitness ATC and MDD modules. We also demonstrate the overall benefits of our new defense approach against drive-by downloads, by measuring the effectiveness of blacklisting the injection domains discovered by WebWitness. In Section 3.5 we cover the limitations of the system. Lastly in Section 3.6 we discuss related work.

Chapter 4 presents our systematic study of modern web based social engineering (SE) malware download attacks. In Section 4.2 we discuss how we collected and identified SE downloads in live network traffic. Section 4.3 details our SE download attack categorization and provides two real classification examples from our study data. Next, Section 4.4 presents our SE study measurements. Our analysis of hundreds of SE malware attack instances reveals that most such attacks are enabled by malicious online advertisement served through a handful of low tier ad networks. In

Section 4.5 we describe WebSentry, a system that defends against the most common types of social engineering malware download attacks. In Section 4.6, we evaluate the performance of WebSentry. In Section 4.7 we cover the limitations of the system. Finally in Section 4.8 we discuss related work.

In Chapter 5, we describe ExecScent, a system that automatically learns C&C traffic models that can adapt to the deployment networks traffic. Section 5.2 provides an overview of our system before we discuss the intuitions that motivated us to build adaptive control protocol templates in Section 5.3. Furthermore, we discuss the advantages of considering the entire content of C&C HTTP requests, rather than limiting ourselves to the URL strings. Then we provide the system details of ExecScent in Section 5.4. In Section 5.5 we present the experimental detection results in different live networks and quantify the advantage of modeling entire HTTP requests, rather than only considering URLs. In Section 5.6 we cover the system limitations. Ending with related work in Section 5.7.

In Chapter 6, we examine algorithms for DPI packet scheduling with the goal of maximizing throughput and minimizing latency. Section 6.2 discusses related work. Section 6.3 describes the properties of the ideal packet scheduler then discuss the algorithms we designed, implemented, and evaluated. In Section 6.4, we explain our testing methodology and present the results. Our goal is to understand the impact each scheduling algorithm has on throughput and latency. To that end we measure raw and scaled throughput as well as average and maximum packet latency for each packet scheduling algorithm.

Chapter 7 concludes the thesis. In Section 7.1 we summarize the contributions and then in Section 7.2 we discuss limitations. Section 7.3 presents future work. Lastly, Section 7.4 concludes with closing remarks.

CHAPTER II

RELATED WORK

This chapter discusses related work that applies to the research explored in this thesis such as detection detection of malware using network communication. Also, each chapter also contains a related work section that covers research specific to it.

2.1 Network Based Malware Detection

There have been a number of studies to address the problem of detecting botnet traffic at the network level, e.g., [59, 60, 128]. BotSniffer [60] and BotMiner [59] are anomaly-based botnet detection systems that look for similar network behavior across hosts. The idea is that hosts infected with the same bot malware have common C&C communication patterns. Furthermore, BotMiner [59] leverages the fact that bots respond to commands in a coordinated way, producing similar malicious network activities. This type of systems require multiple infected hosts on the same monitored network for detection. In addition, being anomaly-based, they are not capable of attributing the infections to a specific malware family, and tend to suffer from relatively high false positive rates.

Wurzinger et al. [128] propose to isolate C&C traffic from mixed malicious and legitimate traffic generated by executing malware samples in a controlled environment. They propose to first identify malicious network activities (e.g., scanning, spamming, etc.), and then analyze the network traffic going back in time until a network flow is found that is likely to represent the command sent to the malware that caused the previously identified malicious activities to be initiated. However, finding commands in malware network traces is not always possible. In fact, most datasets of malware network traces are obtained by running thousands of malware samples, with only a

few minutes of execution time allocated to each sample. Therefore, the chances of witnessing a valid command being sent to a sample within such a small amount of time is intuitively small. On the other hand, malware samples typically attempt to contact the C&C server as soon as they run, even though no command to perform malicious activities may be issued at first contact. For this reason, our works does not focus on identifying malicious network activities performed by the malware, and the related commands. Rather, we leverage any type of (HTTP-based) communication with a C&C server to learn malware communication that can be later used to identify new C&C communications and related C&C domains, even when malicious activities are not directly observable.

Recently researchers have proposed executable reputation systems [65, 110, 126] due to the limitations of signature AV [103]. Instead of using content features from the executable, they focus on properties of the network hosting the malware and related distribution infrastructure. These systems can be very effective at identifying malicious downloads. However, they do not provide context such as how and why the user came to download a malicious executable. Providing download context in addition to detection is a major component of our work. These systems are complementary and can aid in the detection of malicious executables for annotation and study.

2.2 Domain Based Malware Detection

Recently, a number of approaches for identifying malicious domains by monitoring DNS traffic have been proposed [28–30, 34]. Both [28] and [34] are domain reputation systems. The goal of these systems in to assign a low reputation score to a malicious domain, which includes C&C, infector, update and monetization domains. Since no distinction is made between the types of malicious domains they cannot be used to detect infected hosts (e.g., a host may visit an infector domain but not become

infected).

Pleiades [30] detects hosts infected with malware that use a domain generation algorithm (DGA). To identify infections it collects groups of NX-domains generated by a host. Features are extracted from these domains and classified using supervised learning with models created from known DGA malware. Even though it is effective at detecting DGA based botnets, they comprise a small subset of the threat landscape. Thus, it is not a generic solution for detecting infected hosts or new C&C domains.

Kopis [29] is a domain based system that focuses on detecting the malicious domains infected hosts query. It is designed to be deployed at the upper DNS hierarchy so it can observe all hosts that lookup a domain. It calculates features such as the diversity of the network locations of the hosts that query the domain name, their "popularity" and the reputation of the IP address of the resolved domain. It utilizes supervised learning to model malicious versus benign domain lookup behavior. However, these features require the ability to monitor DNS traffic at the upper DNS hierarchy, which is difficult to obtain.

2.3 Host Based Malware Detection

Host based antivirus (AV) is commonly run on end user systems to detect malware infections. AV typically uses signature based detection to identify known malware. In recent years malware has become more sophisticated through the use of packers and crypters. Attackers can leverage popular toolkits to quickly create new variants of their code that bypass AV detection [63, 99]. Since the new malware variant is needed for signature generation, AV tends to lag behind the threat.

Sandboxes provide a controlled, instrumented host based system to execute malware [54]. They typically provide traces of the system calls and network communications performed by an executable. Traces from a sandbox can be used to identify malicious executables [67]. Since the sandbox controls execution it has access to much

more information and context than a system that only has network level visibility. However, malware will often employ anti-sandboxing techniques that detect virtual machine execution and its instrumentation as well as use timing measurements and delays to circumvent analysis [50].

Other researchers have focused on building host based systems to defend against drive-by downloads [46, 83]. Zozzle [46] is a JavaScript malware detector that can be deployed in the browser. It is a supervised learning system that uses features based on the structure and context of the code. Since it is browser based it is able to analyze the code just prior to execution after it has been deobfuscated by the browser. Blade [83] prevents host infections by limiting the execution of binaries downloaded in the browser. When a binary is downloaded Blade places it in a security controlled location that prevents execution. Only when consent is given by the user is the binary allowed to execute.

Both Zoozle and Blade are limited to only detecting drive-by downloads; thus, provide no defense against infections due to social engineering. In addition, they require all hosts on the network to have the software installed and running. On modern networks, imposing this requirement may not be possible due to the variety of devices and the limited control an organization may have on many of them (e.g., bring your own device). Also, as with all host based systems, if malware is successfully executed on the the device, defenses can be disabled leaving the device completely vulnerable to more attacks and infections.

2.4 Malware Detection Challenges

Despite all the research, reliably detecting malware at the vantage point of the network, domain or host remains a difficult problem. For one thing, there is no single behavior that can be categorized as malicious. A network protocol or domain name is not malicious in its own right; rather, we label it as malicious because it is employed

by malware. Likewise on the host, the behavior of an executable (system calls, file modifications, etc.) alone is not enough to label it as malicious – the context of the behavior is also required. In addition, the benign is much more prevalent than the malicious causing even a low false positive rate in a detection system to generate many more false detections than true ones. This results in users losing confidence in the system and ignoring all detections.

To detect malware at the network level we create models from known malware behavior. These models label future observations using classifiers and similarity measures. The key to building a good malware model is selecting a behavior that is difficult for the malware to change, but unique enough to separate it. For example, modeling the protocol structure of the C&C communication of a malware family and placing the emphasis of a match on the unique elements in relation to the traffic on the deployment network (see Chapter 5). In the remaining chapters of this thesis we explore malware behavior that persists across variants and discuss how to model it to maximize true positives while keeping false positives manageable.

CHAPTER III

WEBWITNESS: INVESTIGATING, CATEGORIZING, AND MITIGATING MALWARE DOWNLOAD PATHS

3.1 Introduction

Remote malware downloads currently represent the most common infection vector. In particular, the vast majority of malware downloads occur via the browser, typically due to social engineering attacks and drive-by downloads. A large body of work exists on detecting drive-by downloads (e.g., [44, 46, 83, 92, 113, 132]), and a few efforts have been dedicated to studying social engineering attacks [31, 111, 125]. However, very little attention has been dedicated to investigating and categorizing the *web browsing paths* followed by users *before* they reach the web pages from which the attacks start to unfold.

In this chapter, we study the *web paths* followed by *real users* that become victims of different types of malware downloads, including social engineering and drive-by downloads. We have two primary goals: 1) provide context to the attack by automatically identifying and labeling the sequence of web pages visited by the user *prior to the attack*, giving insight into how users reach attack pages on the web; and 2) leverage these annotated in-the-wild malware download paths to better understand current attack trends and to *develop more effective defenses*.

To achieve these goals we propose a novel malware download *incident investigation system*, named WebWitness, that is designed to be deployed passively on enterprise scale networks. As shown in Figure 2, our system consists of two main components: an *attack path traceback and categorization* (ATC) module and a *malware download*

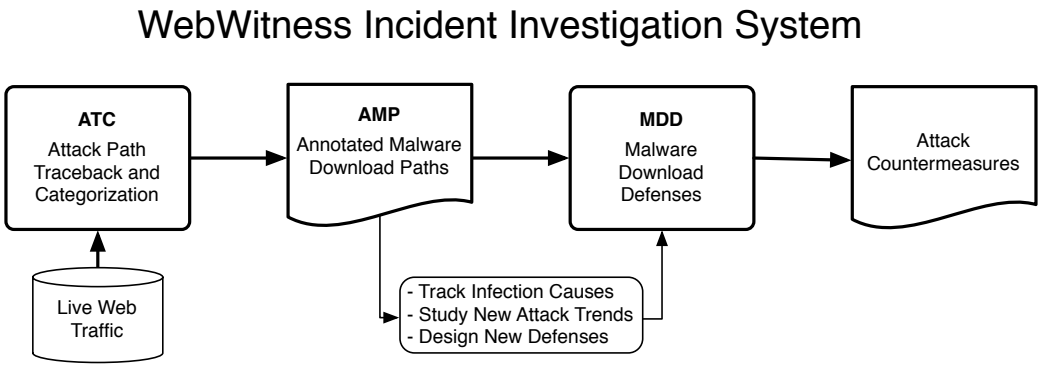

Figure 2: WebWitness – high-level system overview.

defense (MDD) module. Given all (live) network traffic generated by a user's browsing activities within a time window that includes a malware download event, the ATC module is responsible for identifying and linking together all HTTP requests and responses that constitute the web path followed by the user from an "origin" node (e.g., a search engine) to the actual malware download page, while filtering out all other irrelevant traffic. Afterwards, a statistical classifier automatically divides all collected malware download paths into *update*, *social engineering* and *drive-by* attacks. We refer to the output of the ATC module as *annotated malware download paths* (AMP).

The AMPs are continuously updated as new malware downloads are witnessed in the live traffic, and can therefore be used to aid the study of recent attack trends. Furthermore, the AMP data is instrumental in designing and building new defenses that can be plugged into the MDD module (see Figure 2). As an example, by investigating real-world web paths leading to drive-by malware downloads, we found that it is often possible to automatically trace back the domain names typically used in drive-by attacks to inject malicious code into compromised web pages (e.g., via the source of a malicious `script` or `iframe` tag). The injected code is normally used as an

attack trigger, directing the browser towards an actual exploit and finally to a "transparent" malware download and execution. We empirically show that automatically discovering and promptly blocking the domain names serving the injected malicious code is a much more effective defense, compared to the more common approach of blacklisting the URLs that directly serve the drive-by browser exploits themselves or the actual malware executables (see Section 3.4.4).

Chapter Summary. In this chapter we explore the following:

- Investigate the *web paths* followed by *real network users* who eventually fall victim to different types of malware downloads, including social engineering and drive-by downloads. Through this investigation, we provide quantitative information on attack scenarios that have been previously explained only anecdotally or through limited case studies.

- To enable a continuous collection and study of web paths leading to malware download attacks, we build a system called WebWitness. Our system can automatically trace back and categorize in-the-wild malware downloads. We show that this information can then be leveraged to design more effective defenses against future malware download attacks.

- We deployed WebWitness on a large academic network for a period of ten months, where we collected and categorized thousands of *live* malicious download paths. Using these web paths, we were able to design a new defense against drive-by downloads that rely on injecting malicious content into (hacked) legitimate web pages. For example, we show that by leveraging the incident investigation information output by WebWitness, on average we can decrease the infection rate for this type of drive-by downloads by almost *six times*, compared to existing URL blacklisting approaches.

3.2 In-The-Wild Malware Download Study

Goals: In this section we report the results of a large study of *in-the-wild* malware downloads captured on a live academic network. Through this study, we aim to create a labeled dataset of download paths that can be used to design (including feature engineering), train, and evaluate the ATC and MDD modules of WebWitness shown in Figure 2. A detailed discussion of ATC and MDD is reported in Section 3.3.

3.2.1 Collecting Executable File Downloads

To collect executable file downloads we use deep packet inspection to perform on-the-fly TCP flow reconstruction, keeping a buffer of all recent HTTP transactions (i.e., request-response pairs) observed on a live network. For each transaction, we check the content of the response to determine if it contains an executable file. If so, we retrieve all buffered HTTP transactions related to the client that initiated the download. Namely, we store all HTTP traffic a client generated preceding (and including) an executable file download; this allows us to study what *web path* users follow *before* falling victim to malware downloads. All data is saved in accordance with the policies set forth by our Institutional Review Board and are protected under a nondisclosure agreement.

3.2.2 Identifying Malicious Executables

Since many legitimate applications are installed or updated via HTTP (e.g., Windows Update), we immediately exclude all executable downloads from a manually-compiled whitelist of domain names consisting of approximately 120 effective second level domains (e2LDs) of popular benign sites (e.g., `microsoft.com`, `google.com`, etc.). For the remaining downloads, we scan them with more than 40 antivirus (AV) engines, using `virustotal.com`. In addition, we rescan them periodically because many "fresh" malware files are not immediately detected by AV scanners, allowing us to also take into account some "zero-day" downloads. We label a file as malicious if at least one of

the top five AV vendors (w.r.t. market share) and a minimum of two other AVs detect it as malicious. The remaining downloads are considered benign until the rescan. In addition, we discard binary samples that are assigned labels that are too generic or based purely on AV detection heuristics.

3.2.3 Overview of Study Data

To gather our study data we deployed our collection agent (Section 3.2.1) on a large academic network serving tens of thousands of users for a period of 6 months. Notice that the system was deployed for a total of 10 months, with the study conducted in the first 6 months and the evaluation in the 4 months that followed (see Section 3.4 details on the evaluation). During these 6 months, we collected a total of $174,376$ executable downloads from domains that were not on our whitelist. Using the malicious executable identification process defined in Section 3.2.2, we labeled $5,536$ downloads as malicious.

However, many of these malicious downloads were related to *adware*. As we are primarily interested in studying *malware* downloads, because they are potentially the most damaging ones, we devised a number of "best effort" heuristics to separate *adware* from *malware*. For example, given a malicious file, if the majority of AV labels contain the term "adware", or related empirically derived keywords that identify specific unwanted applications (e.g., "not-a-virus", "installer", "PUP", etc.), we label the file as *adware*. The malicious executables not labeled as *adware* by our heuristics were manually reviewed to determine if they were truly *malware*. This resulted in $1,064$ *malware* downloads, with a total of 533 unique samples.

For these 533 unique malware downloads, we performed extensive manual analysis of their download paths, including reverse engineering web pages, heavy javascript deobfuscation, complex plugin content analysis, etc. This time-consuming analysis produced a set of labeled paths, with 164 drive-by, 41 social engineering and 328

update/drop malware download events.

Study Data Limitations: Our collection agent was deployed on an existing production network monitoring sensor. This sensor had limited hardware resources; in addition, our data collection system had to run alongside production software whose functionality could not be disrupted. We therefore collected downloads only during off-peak hours, due to traffic volumes that would oversubscribe the sensor and result in dropped packets during other periods of the day. Thus, the malicious downloads in our study represent only a sample of the ones that occurred during the six month monitoring period. In addition, our system monitors the network in a *purely passive* way; therefore, any malicious downloads preemptively blocked by existing defenses (e.g., URL blacklists such as Google Safe Browsing) were not observed. Yet, based on our extensive manual analysis, we believe the 533 malware downloads to be sufficiently diverse and representative of the overall set of malware downloads that occurred during our study period.

3.2.4 Download Path Traceback Challenges

One of the goals of our system is to automatically trace back the sequence of steps (i.e., HTTP transactions) that lead victims to be infected via a malware download. One may think that reconstructing the *web path to infection* is fairly easy, because we could rely on the `Referer` and `Location` header fields to link subsequent HTTP transactions together (see RFC2616). For example, a simple strategy would be to start from the download transaction and "walk back" the sequence of transactions by following the `Referer` header found in the HTTP requests.

Unfortunately, in practice download path traceback is much more difficult than it may seem at first. Depending on the particular version of the browser, JavaScript engine, and plugin software running on the client, the `Referer` and/or `Location` headers may be suppressed (e.g., see [70]), resulting in the inability to correctly reconstruct

the entire sequence of download path transactions in a given network trace.

Deriving and Measuring Surrogate Features: As part of our study, we reviewed hundreds of malicious download traces. In most cases we cannot rely completely on the `Referer` and `Location` headers, and we therefore derive surrogate "referrer indicator" features and heuristics, which can be used to perform a more complete download path traceback. Next, we define each of the features we observed, and then provide a measure of how prevalent they are for malware download paths. While in this section we simply *measure their prevalence*, we later use these features to automate path traceback (Section 3.3).

First, let us more precisely define what we mean with *download path traceback*. Let T_d indicate an HTTP transaction carrying an executable file download initiated by client C. Given the recording of all web traffic generated by C during a time window preceding (and including) T_d, we would like to reconstruct the sequence of transactions $(T_1, T_2, \ldots, T_d)$ that led to the download, while filtering out all unrelated traffic. This sequence of transactions may be the consequence of both explicit user interactions (e.g., a click on a link) and actions taken by the browser during rendering (e.g., following a page redirection). Notice that the traffic trace we are given may contain a large number of transactions that are completely unrelated to the download path, simply because the user may have multiple browser tabs open and multiple web-based applications active in parallel. Thus, potentially producing a large amount of overlapping unrelated traffic.

Let T_1 and T_2 be two HTTP transactions. We found that the features/heuristics listed below can be used to determine whether T_1 is a *likely source* of T_2, therefore allowing us to "link" them with different levels of confidence. Table 1 summarizes the prevalence of each feature in both drive-by and social engineering downloads (we discuss how we can distinguish drive-by from social engineering later in Section 3.2.5). A detailed discussion of how WebWitness uses these features for automated download

path traceback is given in Section 3.3.

(1) **Location**: According to RFC2616, if transaction T_2's URL matches T_1's `Location` header, it indicates that T_2 was reached as a consequence of a server redirection from T_1.

(2) **Referrer**: Similarly, if T_1's URL matches T_2's `Referer` header, this indicates that the request for T_2 originated (either directly or through a redirection chain) from T_1, for example as a consequence of page rendering, a click on a hyperlink, etc.

(3) **Domain-in-URL**: We observed that advertisement URLs often embed the URL of the page that displayed the ad. So, if T_1's domain name is "embedded" in T_2's URL, it is likely that T_1 was the "source" of the request, even though the `Referer` is not present. This is especially true if there is only a small time gap between the transactions.

(4) **URL-in-Content**: If T_1's response content includes T_2's URL (e.g., within an HTML or non-obfuscated JavaScript code), this indicates there is (potentially) a "source of" relationship that links T_1 to T_2.

(5) **Same-Domain**: By investigating numerous drive-by malware downloads, we found that in many cases the exploit code and the malware executable file itself are served from the same domain. This approach is likely chosen by the attackers because if the exploit is successfully served, it means that the related malicious domain is currently reachable and serving the malware file from the same domain helps guarantee a successful infection (a similar observation was made in [65]). Therefore, if T_1 and T_2 share the same domain name and are temporally close, this likely indicates that T_1 is the "source of" T_2.

(6) **Commonly Exploitable Content (CEC)**: In our observations, most drive-by downloads use "commonly exploitable" content (e.g., .jar, .swf, or .pdf files that carry an exploit) to compromise their victims. The exploit downloads the

malicious executable; thus, if T_1 contains commonly exploitable content (CEC) and T_2 is an executable download that occurred within a small time delta after T_1, this indicates that T_1 may be the "source of" T_2.

(7) **Ad-to-Ad**: In some cases, we observed chains of ad-related transactions where the `Referer` and `Location` header are missing (e.g., due to JavaScript or plugin-driven redirections). Therefore, if T_1 and T_2 are consecutive ad-related requests (e.g., identified by matching their URLs against a large list of known ad-distribution sites) and were issued within a small time delta, this indicates there may be a "source of" relationship.

Table 1: Success rate of traceback method and "Source-of" relationships in malware download paths. The numbers indicate the percentage of analyzed download paths.

Traceback method success rate	Drive-by	Social Eng.
Only Referrer and Location	0%	53%
All surrogate referrer features	96%	95%

Feature	Drive-by	Social Eng.
Location	69%	73%
Referrer	97%	100%
Domain-in-URL	0%	5%
URL-in-Content	17%	17%
Same-Domain	97%	20%
CEC	5%	0%
Ad-to-Ad	6%	10%

As a confirmation to the fact that tracing back malware download paths is challenging, we found that not a single drive-by download in our dataset could be traced back by relying only on the `Referer` and `Location` headers. For example, even if 97% of the drive-by download paths contained at least one pair of requests linked via the `Referer`, all drive-by paths contained at least some subsequence of the path's transactions that could not be "linked" by simply using the `Referer` header.

For social engineering paths, we found that 53% of the downloads could be traced back using only the `Referer` and `Location` headers. When this was not possible, the main cause was the presence of requests made via JavaScript and browser plugins. In some cases, we were not able to fully trace back the download path. The cause for the majority of the untraceable drive-by (4%) and social engineering (5%) downloads,

when using all the features, was missing transactions likely due to our system not observing all related packets.

3.2.5 Drive-by vs. Social Engineering

We label a malware download path as *social engineering* if explicit user interaction (e.g., a mouse click) is required to initiate a malware download. In contrast, we label as *drive-by* those malware downloads that are transparently delivered to the victim via a browser exploit. As mentioned earlier (Section 3.2.3), during our study, we were able to manually review and label 164 drive-by and 41 social engineering malware downloads.

What distinguishes drive-by from social-engineering: In the following we report the characteristics that we observed for different types of paths. In particular, some of these characteristics could be leveraged as statistical features to build a classifier that automatically distinguishes between drive-by and social engineering downloads (see Section 3.3). We also discuss characteristics of malware updates/drops that could be used to filter out download paths that belong neither to the drive-by nor to the social-engineering class. Table 2 summarizes the prevalence of each of the characteristics described below.

Table 2: Download path properties.

Feature	Drive-by	Social eng.
Candidate Exploit Domain Age	0	-
Drive-by URL Similarity	69%	0%
Download Domain Recurrence	0.6%	34%
Download Referrer	0.6%	95%
Download Path Length	6	7
User-Agent Popularity	95%	98%

(1) **Candidate Exploit Domain "Age":** Drive-by download attacks often exploit their victims by delivering exploits via files of popular content types such as `.jar`, `.swf`, or `.pdf` files; we simply refer to these file types as "commonly exploitable" content (CEC). For example, during our study, we found that 94% of the drive-by download paths at some point delivered the exploit via CEC. The domains

serving these exploits tend to be short-lived compared to domains serving benign content of the same type. Therefore, CEC served from a recently registered domain is an indicator of a possible drive-by download path. On the other hand, none of the social engineering download paths we observed during our study had this property. Table 2 reports the median domain name "age", computed as the number of days of activities for the domain of a page serving CEC, measured over a very large passive DNS database. The median age is less than one day for drive-by paths, and is not indicated for social engineering paths, because none of the nodes in the social engineering path served content of the type we consider as CEC (the overall traffic traces included HTTP transactions that carried content such as `.swf` files, but none of those were on the download path).

(2) **Drive-by URL Similarity:** The majority of drive-by downloads (about 70% of our observations) are served by a small number of exploit kits. Therefore, in many cases the exploit delivery URLs included in drive-by download paths share a structural URL similarity to known exploit kit URLs. Table 2 reports the fraction of drive-by download paths that had a similarity to known exploit kit URLs greater than 0.8, measured using the approach proposed in [96].

(3) **Download Domain Recurrence:** Most domains serving drive-by and social engineering malware download are contacted rarely, and often only once by one particular client at the time of the attack. On the other hand, malicious software regularly checks for executable updates. To approximately capture this intuition, we measured the number of queries to the malware download domain. As shown in Table 2, only 0.6% of the malware download domains in our drive-by paths are queried mulitple times within a small time window (two days, in our measurements). The higher percentage of social engineering malware paths with download domain recurrence is due to the fact that a significant fraction of the ones we observed used a free file sharing website for the malware download and

that we count the domain query occurrences in aggregate, rather than per client.

(4) **Download Referrer:** In case of social engineering attacks, the HTTP transaction that delivers the malicious file download tends to carry a `Referer`, usually due to the direct user interaction that characterizes them. On the other hand, drive-by attack malware file delivery happens via a browser exploit. The request initiated from the shell code typically does not have a `Referer` header. Similarly, malware updates/drops initiated by malicious applications are already running on a compromised machine, and usually do not carry any referrer information. Table 2 shows that only 0.6% of all drive-by paths, in contrasts to 95% of social engineering paths, carried a `Referer` in the download node.

(5) **Download Path Length:** Drive-by and social engineering attacks typically generate download paths consisting of several nodes, mainly because a user has to first browse to a site that eventually leads to the actual attack. In addition, the malware distribution infrastructure is often built in such ways that enables malware downloads "as a service", which entails the use of a number of "redirection" steps. In contrast, download paths related to malware updates or drops tend to be very short. Table 2 reports the median number of nodes for drive-by and social engineering paths. In case of malware updates/drops, the median length for the path was only one node.

(6) **User-Agent Popularity:** The download paths for both drive-by and social engineering downloads typically include several nodes that report a popular browser user-agent string, as the victims use their browser to reach the attack. On the other hand, in most cases of a malware drop/update, it is not the browser, but the update software making the requests. In practice, we observed that the majority of malware update download paths did not report a popular user-agent string (only 36% of them did). Table 2 reports the percentage of paths that include a popular user-agent string.

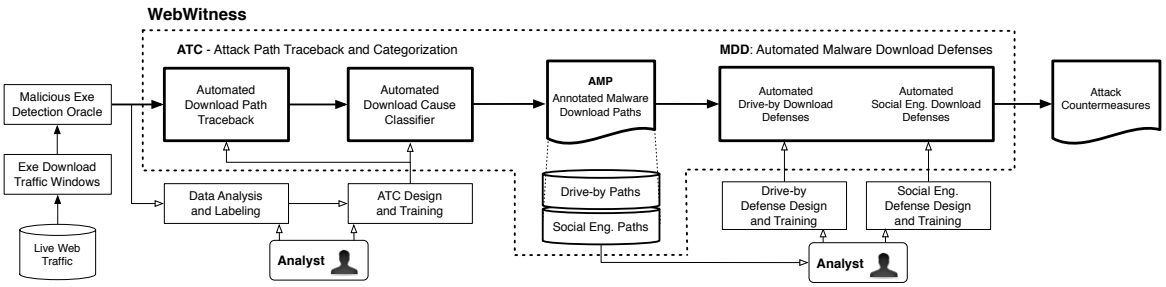

Figure 3: WebWitness system details.

3.3 WebWitness

Inspired by our study of real-world malware download paths, we develop a system called WebWitness that can automate the investigation of new malware download attacks. The primary goal of this system is to provide *context* around malicious executable downloads. To this end, given a traffic trace that includes all web traffic recorded during a time window preceding (and including) a malicious executable file download, WebWitness automatically traces back and categorizes the web paths that led the victim to the malicious download event.

In this section, we describe the components of our system, which are shown in Figure 3.

3.3.1 ATC - Download Path Traceback

Given a malicious file download trace from a given client, WebWitness aims to trace back the *download path* consisting of the sequence of web pages visited by the user that led her to a malware download attack (e.g., via social engineering or to a drive-by exploit). As detailed in Section 3.2.4, the trace may contain many HTTP transactions that are unrelated to the download. Furthermore, it is not always possible to correctly link two related consecutive HTTP transactions by simply leveraging their HTTP `Referer` or `Location` headers.

To mitigate the limitations of referrer-only approaches and more accurately trace

back the download path, we devise an algorithm that leverages the features and heuristics we identified during our initial study of in-the-wild malware downloads presented in Section 3.2.4. In summary, we build a *transactions graph*, where nodes are HTTP transactions within the download trace, and edges connect transaction according to a "probable source of" relationship (explained in detail below). Then, starting from the node (i.e., the HTTP transaction) related to the malware file download, we walk back along the most probable edges until we find a node with no predecessor, which we label as the "origin" of the download path. In the following, we provide more details on our traceback algorithm.

Transactions Graph. Let D be the dataset of HTTP traffic generated by host A before (and including) the download event. We start by considering all HTTP transactions in D, and construct a weighted directed graph $G = (V, E)$. The vertices are A's HTTP transactions and the edges represent the relation "probable source of" for pairs of HTTP transactions. As an example, the edge $e = (v_1 \rightarrow v_2)$ implies that HTTP transaction v_1 likely produced HTTP transaction v_2, either automatically (e.g., via a server-imposed redirection, javascript, etc.) or through explicit user interaction (e.g., via a hyperlink click). Thus, we can consider v_1 as the "source of" v_2. Each edge has a weight that expresses the level of confidence we have on the "link" between two nodes (the weights are ordinal so their absolute values are not important). For example, the higher the weight assigned to $e = (v_1 \rightarrow v_2)$, the stronger the available evidence in support of the conclusion that v_1 is the "source of" v_2 (edge weights are further discussed below). Also, let t_1 and t_2 be the timestamp of v_1 and v_2, respectively. Regardless of any available evidence for a possible edge, the two nodes may be linked only if $t_1 \leq t_2$.

Heuristics and Edge Weights. To build the graph G and draw its edges, we leverage the seven features that we indentified in Section 3.2.4. Specifically, given two nodes (essentially, two URLs) in the directed graph G described earlier, an edge

$e = (v_1 \rightarrow v_2)$ is created if any of the seven features is satisfied. For example, if v_1 and v_2 can be related via the "Domain-in-URL", we draw an edge between the two nodes. We associate a weight to each of the seven features; the "stronger" the feature, the higher its weight. For example, we assign a weight value $w_e = 7$ to the "Location" feature, $w_e = 6$ to the "Referrer" feature, and so on, with the "Ad-to-Ad" receiving a weight $w_e = 1$. The weight values are conveniently assigned simply to express relative importance and precedence among the edges to be considered by our greedy algorithm. If more than one feature happens to link two nodes, the edge will be assigned a weight equal to the maximum weight among the matching features.

Traceback Algorithm. Once G has been built, we use a *greedy algorithm* to construct an approximate "backtrace path". We start from the graph node related to the executable download event, and walk backwards on the graph by always choosing the next edge with the highest weight. Consider the example graph in Figure 4, in which thicker edges have a higher weight. We start from the download node d. At every step, we walk one node backwards following the highest weight edge. We proceed until we reach a node with no predecessor, which we mark as the *origin* of the download path. If a node has more than one predecessor whose edges have the same weight, we follow the edge related to the predecessor node with the smaller time gap to the current node (measured w.r.t. the corresponding HTTP transactions).

Possible False and Missing Edges: Naturally, the heuristics we use for tracing back the download path may in some cases add "false edges" to the graph or miss some edges. However, notice that these challenges are mitigated (though not always completely eliminated) by the following observations:

i) Our algorithm and heuristics aim to solve a much narrower problem than finding the correct "link" between all possible HTTP transactions in a network trace, because we are only concerned with tracing back a sequence of HTTP transactions that terminate into a malicious executable download.

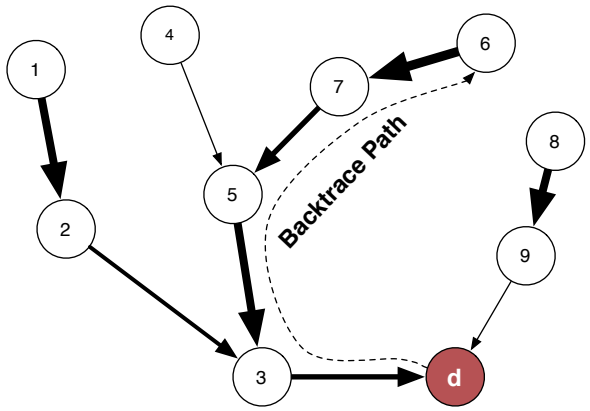

Figure 4: Example of download path traceback.

ii) The "false edge" problem is mitigated by the fact that we always follow the strongest evidence. For example, consider Figure 4. Suppose the edge $(2 \rightarrow 3)$ was drawn due to rule (6), while edge $(5 \rightarrow 3)$ was drawn due to rule (2). In this case, even though edge $(2 \rightarrow 3)$ was mistakenly drawn (i.e., nodes 2 and 3 have no real "source of" relationship), the mistake is irrelevant, because our algorithm will choose $(5 \rightarrow 3)$ as part of the path, which is supported by stronger evidence.

iii) Our algorithm can output not only the sequence of HTTP transactions, but also the nature (and confidence) of every edge. Therefore, a threat analyst (or a downstream post processing system) can take the edge weights into account, before the reconstructed download path is used to make further decisions (e.g., remediation or takedown of certain domains in the download path).

3.3.2 ATC - Download Cause Classification

After we trace back the download path, we aim to label the reconstructed path as either *social engineering* or *drive-by* download. As shown in Figure 3, the output of this classification step allows us to obtain the annotated malware download paths (AMPs), which are then provided as input to the defense module (MDD).

While we are mainly interested in automatically identifying social engineering and

drive-by download paths, we build a three-class classifier that can distinguish between three broad download causes, namely *social engineering*, *drive-by*, and *update/drop*. Essentially the *update/drop* class allows us to more easily identify and exclude malware downloads that are not caused by either social engineering or drive-by attacks.

To automatically classify the "cause" of an executable file download, WebWitness uses a supervised classification approach. First, we describe how we derive the features needed to translate malware download events into feature vectors that can be given as input to a statistical classifier. Then, we discuss how we derive the dataset used to train the classifier. To actually build the classifier, we used the random forest algorithm [37] (see Section 3.4).

Features: To discriminate between the three different classes, we engineered six statistical features that reflect, with a one-to-one mapping, the six characteristics of drive-by and social-engineering malware download paths that we discussed and measured in Section 3.2.5. For example, we measure binary feature (1) "Download Referrer" as true if the HTTP request that initiated the download has a `Referer` header; a numerical feature (2) representing the "age" of domains serving "commonly exploitable" content; etc.

Training dataset: To train the classifier, we use the dataset of in-the-wild malware download paths that we collected and manually labeled during our initial investigation of in-the-wild malware downloads discussed in Section 3.2.5. Our training dataset contained the following number of labeled download paths: 164 instances of *drive-by* download paths, 191 instances of *social engineering* paths, and 328 *update/drop* samples.

3.3.3 MDD - Drive-by Defense

The annotated download paths output by ATC provide a large and up-to-date dataset of real-world malware download incidents, including the web paths followed by the

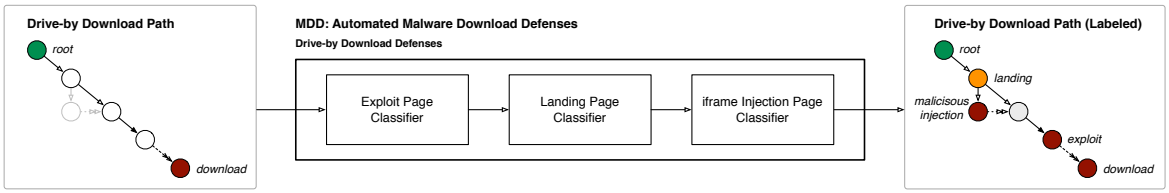

Figure 5: MDD: labeling of drive-by download paths based on malicious content injections.

victims (see Figure 3). This information is very useful for studying new attack trends and developing more effective defenses. As new defenses are developed, they can be plugged into the MDD module, so that as new malware download paths are discovered we can automatically derive appropriate countermeasures.

As an example that demonstrates how WebWitness can enable the development of more effective malware download defenses, we develop a new defense against drive-by download attacks based on code injections. While code injection attacks are not new, current defenses rely mainly on blacklisting the URLs serving the actual drive-by exploit or malware download, rather than blocking the URLs from which malicious code is injected. Our results (Section 3.4) show that by automatically tracing back drive-by download paths and identifying the code injection URLs, we can enable better defenses against future malware attacks.

Identifying code injection URLs: Given a *drive-by* download path output by the ATC module, we aim to automatically identify the *landing*, *injection*, and *exploit* nodes within the download path.We tackle this problem using a supervised classification approach. Namely, we train a separate classifier for each of the three types of nodes on a drive-by download path. The final output is a labeled drive-by download path.

Exploit Page Classifier: The exploit classifier takes as input a drive-by download path and labels its nodes as *exploit* or *non-exploit*. We define an exploit node as a page that carries content that exploits a vulnerability on the victim's machine, causing it

to eventually download a malicious executable. The search for exploit nodes proceed "backwards", starting from the node prior to the executable download and ending at the root. It is not uncommon to have more than one exploit node in one path (e.g., some exploit kits try several exploits before success). Thus, multiple nodes could be labeled as *exploit*.

To build the classifier, we use the following features:

(1) *Hops to the download page*. Number of nodes on the download path between the considered node and the final malware download node. *Intuition*: It is typical for the exploit node to only be a few hops away from the actual download. In many cases, the node prior to the download event is an exploit node, because once the exploit succeeds the executable is downloaded immediately.

(2) *"Commonly exploitable" content*. Boolean feature that indicates if a node contains content for Java, Silverlight, Flash or Adobe Reader. *Intuition*: Browser plug-ins are a popular exploitation vector. The exploit is typically delivered though their content.

(3) *Domain age*. The number of days since the first observation of the node's effective second level domain in a large historic passive DNS database. *Intuition*: Exploit domains tend to be short-lived and often only active for one day.

(4) *Same domain*. Boolean feature that is true if the node's domain is equal to the download domain. *Intuition*: It is common for the exploit and download to be served by the same domain, as also noted in [65].

Landing Page Classifier: Once the exploit node(s) is labeled, we attempt to locate the landing page URL. Essentially, the landing page is the web page where the drive-by attack path begins. Often, the landing page itself is a non-malicious page that was previously compromised (or "hacked"). The landing page classifier calculates the probability that a node preceding the exploit node (labeled by the exploit page classifier discussed earlier) is a landing page. Nodes with a probability higher than

a tunable detection threshold (50% in our experiments) are classified as "candidate landing" nodes. If there are multiple candidates, the one with the highest probability is labeled as the landing node.

To label a node as either *landing* or *non-landing*, we engineered the following statistical features:

(1) *Hops to the exploit page.* This feature set consists of the number of non-redirect nodes and unique effective second level domains between the node and the exploit node. *Intuition*: Often, all the nodes between the landing and exploit node are redirects [124]. Also, most drive-by downloads use one to three types of malicious domains (injection, exploit, download). Therefore, in most cases there are zero or one domains (the one being the injection domain) on the download path between the landing and exploit nodes.

(2) *Domain age.* We use two features based on domain age. The first feature is the age of the node's effective second level domain as computed from a passive DNS database. *Intuition*: The domains associated to ("hacked") landing pages tend to be long-lived. Furthermore, "older" landing pages tend to offer more benefits to the attackers, as they often attract more visitors (i.e., potential victims), because it takes time for legitimate pages to become popular. The second feature is the age of the oldest domain between the node and the exploit node. *Intuition*: Nodes on the download path between the landing and exploit nodes tend to be less than a year in age. This is because they are typically malicious and recently registered.

(3) *Same domain.* Boolean feature that is true if the node's domain is equal to the exploit domain. *Intuition*: It is uncommon for an exploit to be served from the same domain as the landing page. They are typically kept separate because installing an exploit kit on a compromised website may increase the likelihood of detection by the legitimate site's webmaster. In addition, it is much easier to manage a centralized exploit kit server than keep all the compromised websites

up-to-date with the latest exploits.

Injection Page Classifier: We define the injection page to be the source of the code inserted into the "hacked" landing page. Typically, the injection and exploit nodes are separate and are served via different domain names. This provides a level of indirection that allows the exploit domain to change without requiring an update to the landing page. The injection node by definition is a successor to the landing page, but depending on the injection technique it may or may not be directly present in the download path traced back by the ATC module. Therefore, the classifier calculates the injection page probability for each direct successor of the landing node in the transactions graph, instead of only considering nodes in the reconstructed download path. The successor of the landing page node with the highest probability is labeled as the injection page node.

To identify the *injection* page, for each successor of the *landing* node we measure the following features:

(1) *On path.* Boolean feature indicating if the node is on the download path. *Intuition*: Being on the download path and a successor of the landing page, makes it a good candidate for the injection node. However, the injection node is not always on the download path due to the structure of some drive-by downloads.

(2) *Advertisement.* Boolean feature that is true if the node is an ad. *Intuition*: By definition, the injection page is not an ad, but code injected into the landing page. It is common for ads that are not related to the malicious download to be served on a landing page. This feature help us exclude those ad nodes.

(3) *Domain age.* The number of days since the first observation of the node's effective second level domain in passive DNS. *Intuition*: Injection pages typically have the sole purpose of injecting malicious code. They are rarely hosted directly on compromised pages, because this would expose the malicious code to cleanup by the legitimate site owners, ending the attacker's ability to exploit visitors.

Consequently, injection pages are hosted on "young" domains that are typically active for the lifetime of a website compromise.

(4) *Successors.* There are two features that are derived from the node's successors. First is the number of direct successors. *Intuition*: Injection nodes tend to have only one direct successor. They typically perform an HTTP redirect or dynamically update the DOM to include the URL of the exploit domain. Benign pages often have more than one direct successor because they load content from many different files or sources. The second feature is boolean and it is true if one of the node's successors is on the download path. It indicates there is a possible "source of" relationship between it and a node on the download path. Even though the node itself may not be on the download path.

(5) *Same domain.* There are two boolean features that compare domain names. The first checks for equality between the node's domain and the landing domain. *Intuition*: It is uncommon for the landing domain to equal the injection domain for reasons similar to those described in the landing page classifier's "same domain" feature described earlier. The second feature compares the node's domain to the exploit domain. *Intuition*: In approximately 70% of the observations in our measurement study (Section 3.2), the exploit and injection domains were different.

3.4 Evaluation

In this section, we evaluate WebWitness' ATC and MDD modules. We also demonstrate the overall benefits of our new defense approach against drive-by downloads, by measuring the effectiveness of blacklisting the injection domains discovered by WebWitness. We show that while blacklisting the injection domains provides a better defense, compared to blacklisting only the exploit and download domains, injection domains appear very rarely in current blacklists, including Google Safe Browsing and

a variety of large public blacklists.

3.4.1 ATC - Download Cause Classification

The download cause classifier uses a supervised learning approach to label each download path as either *social engineering*, *drive-by* or *update/drop* (Section 3.3.2). To evaluate its accuracy, we use WebWitness to traceback and classify all malicious downloads collected from the large academic network (Section 3.2) in the months *following* our initial study and development of the system. Specifically, all download events and samples used during evaluation have *no overlap* with the data we used for the study presented in Section 3.2, to design WebWitness' features and heuristics, or to train our classifiers. Each malicious download observed during the *testing period* was then classified as one of the following: *drive-by*, *social engineering* or *update*. From each of the three predicted classes we randomly sampled 50 downloads for manual verification. We limited the sample size to a total of 150 downloads because of the extensive manual analysis required to determine the ground truth, including reverse engineering web pages, heavy javascript deobfuscation, complex html and plugin content analysis, etc. This time consuming review process allowed us to identify the correct web path and the *true cause* of download, creating our *ground truth* for the evaluation. Table 3 reports the confusion matrix for the cause classifier.

Table 3: Cause Classifier - confusion matrix results

		Predicted Class		
	Class	Drive-by	Social	Update/Drop
Ground Truth	Drive-by	47	1	0
	Social	2	46	3
	Update/Drop	1	3	47

The classifier correctly labeled over 93% of the downloads. Notice that these results represent the overall system performance of the ATC module, because the download paths used in the experiment (i.e., input to the cause classifier) were extracted using our download path traceback algorithm (Section 3.3.1). The two social engineering samples classified as drive-by downloads both had commonly exploitable

content (CEC) on the download path. They were misclassified even though the CEC domain ages were greater than 200 days. The three update/drop samples classified as social engineering was caused by invalid download paths resulting from the false edges described in the next section. Finally the three social engineering downloads misclassified as update/drop was a result of small downloads paths (all were length 3) and high download domain recurrence (all greater than 20 of the 48 hourly buckets).

3.4.2 ATC - Download Path Traceback

To evaluate the accuracy of our download path traceback algorithm (Section 3.3.1), we use the 150 manually reviewed downloads; i.e., our ground truth, from Section 3.4.1. For path traceback, we consider two types of errors for review: (1) missing nodes: the traceback stops short, before reaching the origin of the download path (recall that the traceback algorithm works its way backwards from the download node to the path origin); (2) false node: a node that should not appear in the download path. Table 4 summarizes the results of our evaluation.

Table 4: Download path traceback results.

	Paths	Correctly Traced Back	Missing	False
Drive-By	48	45	3	0
Social	51	46	2	3
Update/Drop	51	47	0	4

The results show that 92% of the download paths were correctly traced back by our system. The 5 with missing nodes all had a *referer* header in the origin node's request, but a matching URL was not contained in the trace. This was likely due to our system not observing all the packets related to those transactions. The 7 with the false nodes were all caused by the "same-domain" heuristic incorrectly connecting the paths of an update and a social engineering download. The heuristic failed because the updates were performed by a malicious executable seconds after the user was socially engineered into downloading it from the same domain as the update.

3.4.3 MDD - Detecting Injection Domains

As discussed in detail in Section 3.3.3, we aim to automatically identify the malicious code injection domains often employed in drive-by download attacks. To achieve this goal, we use a cascade of three classifiers: an *exploit*, a *landing*, and an *injection* classifier (Section 3.3.3). In the following, we evaluate the performance of each one.

To build the training dataset, we use 117 drive-by malware downloads collected and manually labeled during our six-month malware study described in Section 3.2. These 117 drive-by paths contained 246 exploit nodes (notice that it is not uncommon for a drive-by attack to serve more than one exploit, especially when the first exploit attempt fails). There is only one landing node and one injection node per download path.

Table 5: Node labeling for drive-by download paths

Experiment	Classifier	Correctly Labeled	Incorrectly Labeled
Cross-Validation	Exploit	99.19%	0%
	Landing	96.58%	0.17%
	Injection	94.87%	0.07%

We performed 10-fold cross-validation tests using the dataset described above. Table 5 summarizes the results. As can be seen, all classifiers are highly accurate. The results of the the injection page classifier represent the performance of the final injection domain detection task. This is due to fact that all tests were conducted using the three classifiers (exploit, landing, and injection) in cascade mode to mirror an actual deployment of WebWitness' MDD module. Thus, overall, we obtained a minimum of 94.87% detection rate at 0.07% false positives.

There were a total 7 domains mislabeled as injection by our system. The most common error was labeling the exploit domain as the injection domain; i.e., missing the fact that a separate injection domain existed. This was the case for 5 of the 7 mislabeled domains. Since these domains are malicious, blacklisting them will not cause false positives. The other two domains were benign. One of them had an Alexa

rank over $260,000$ and the other above $1,600,000$. To mitigate such false positives, the newly discovered injection domains could be reviewed by analysts before blacklisting. As WebWitness provides the analyst with full details on the traffic collected before the download and the reconstructed download path, this information can make the analyst's verification process significantly less time-consuming.

3.4.4 MDD - Defense Efficacy & Advantages

Domain name and URL blacklisting are commonly practiced defenses [6]. However, blacklists are only effective if the blacklisted domains remain in use for some period of time after they are detected. The longer-lived a malicious domain, the more useful it is to blacklist it. As discussed in Sections 3.3.3 and 3.4.3, WebWitness is able not only to identify the domains from which malware files are downloaded, but also to identify the malicious code injection and exploit domains within drive-by malware download paths. Clearly, these domains are all candidates for blacklisting.

To evaluate the efficacy of blacklisting the code injection domains, we demonstrate the advantages this provides compared to the currently more common approach of blacklisting the exploit and download domains. To this end, we use a set of 88 "complete" injection-based drive-by download paths that we were able to collect from a large academic network. These samples were "complete" paths in the sense that they were manually verified to have an injection, exploit, and malware download node (and related domain).

We evaluate the effect of blocking the different types of drive-by path domains by counting the number of potential victims that would be saved by doing so. Specifically, we define a potential victim as a unique client host visiting a blacklisted domain. Notice that the actual number of hosts that get infected may be smaller than the number of potential victims, because only some of the hosts that visit a malicious domain involved in a drive-by download attack will "successfully" download and run

the malware file (e.g., because an anti-virus blocked the malware file from running on the machine). However, we can use the potential victim count to provide a relative comparison on the effectiveness of blacklisting injection versus exploit and malware download domains.

To count the potential victims, we rely on a very large passive DNS (pDNS) database that spans multiple Internet Service Providers (ISPs) and corporate networks. This pDNS dataset stores the historic mappings between domains and IP addresses, and also provides a unique source identifier for each host that queries a given domain name. This allows us to identify all the unique hosts that queried a given domain in a given timeframe (e.g., a given day). For each injection-based drive-by download paths in our set, we compute the potential victims saved by counting the number of unique hosts that query the injection, exploit, and file download domains in the 30 days following the date when we observed and labeled the download event. Figure 6 shows our results, in which day-0 is the day when we detected a malicious download path (the victims counts are aggregated, per day, for all hosts contacting a malicious domain). We can immediately see that the number of potential victims that query the exploit or file download domains rapidly drops as the exploit domain ages. On the other hand, injection domains are longer lived, and blacklisting them would prevent a much larger number of potential victims from being redirected to new (unknown) and frequently churning exploit and file download domains. Blacklisting the injection domain saves almost 6 *times* more potential victims, compared to blacklisting the exploit domain.

3.4.5 Blacklists & Google Safe Browsing

In this section, we aim to gain additional insights into the advantages that could be provided by our WebWitness' MDD module, compared to existing domain blacklists.

Public Domain Blacklists First, given the entire set of malicious domain names

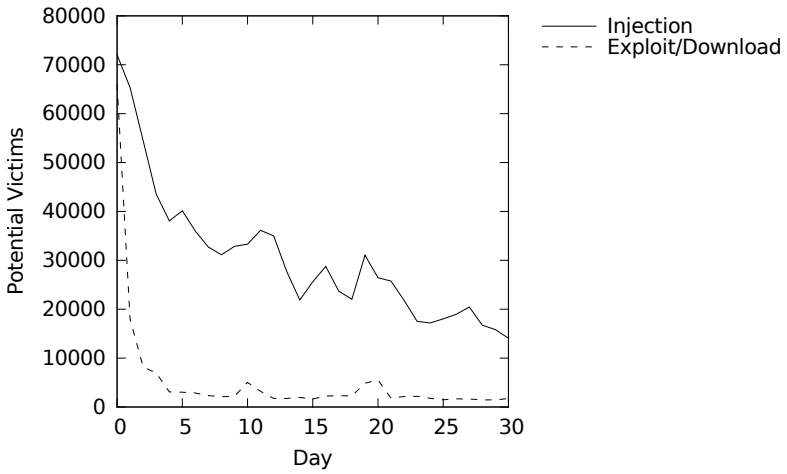

Figure 6: Potential victims saved by blocking the injection versus exploit/download domains on drive-by paths.

related to drive-by downloads discovered during our study and deployment of Web-Witness, we counted how many of these domains appeared in popular public blacklists. We also measured the delay between when we first discovered the domain on a malware download path and when it appeared on a blacklist. This was possible because we repeatedly collected all domain names reported by the following set of public blacklists every day for more than a year: support.clean-mx.de, malwaredomains.com, zeustracker.abuse.ch, phishtank.com and malwaredomainlist.com. Table 6 summarizes our findings.

Table 6: Public blacklisting results.

	Uniq. Domains		Days: Detect to Blacklist		
	Observed	Blacklisted	Min.	Med.	Mean
Exploit/Download	152	9	1	20	29
Injection	52	6	20	31	36

As shown in Table 6, from all drive-by download paths that we were able to identify, reconstruct, and label, we collected a total of 52 unique drive-by code injection domains and 152 unique drive-by exploit and malware file download domains. Overall, less than 10% of these domains ever appeared on a public blacklist. As we can see, more exploit/download domains (a total of 9) were blacklisted, compared to the

injection domains (only 6). Furthermore, we can see that the minimum time it took for an injection domain to appear in at least one blacklist was 20 days, whereas some exploit domains were blacklisted almost immediately (after only one day).

Because injection domains are typically longer lived than exploit domains, and because the same injection domain is often used throughout the course of a drive-by download campaign to redirect users to different (short-lived) exploit domains, identifying and blocking injection domains has a significant advantage. By helping to quickly identify and blacklist injection domains, WebWitness enables the creation of better defenses against drive-by downloads, thus helping to significantly reduce the number of potential malware victims, as we also demonstrated in the previous Section 3.4.4.

Google Safe Browsing For the last few weeks of our deployment of WebWitness, we checked the domain names related to the drive-by download paths reconstructed by our system against Google Safe Browsing (GSB) [6]. Specifically, given a malware download path and its malicious domains, we queried GSB on the next day, compared to the day the malware download was observed. Overall, during this final deployment period we observed 34 drive-by download paths. GSB detected a total of 6 malicious domains that were related to only 4 out of the 34 downloads. The domains GSB detected were used to serve drive-by exploits, the malware file themselves, or were related to ads used to lead the victims to a browser exploit. *None of the domains detected by GSB were injection domains*, even though our 34 download paths included 12 unique injection domains.

It is important to notice, however, that while GSB detected malicious domains related to only 4 out of our 34 drive-by download paths, there may be many more malware downloads that WebWitness cannot observe, simply because they are blocked "up front" by GSB. Because WebWitness passively collects malware download traces from the network whenever a malicious executable file download is identified in the

traffic, it is very possible that in many cases GSB simply prevented users who were about to visit a drive-by-related domain from loading the malicious content, and therefore from downloading the malware file in the first place. Nonetheless, the fact that WebWitness automatically discovered 30 drive-by download paths that were not known to GSB demonstrates that our system can successfully complement existing defenses.

3.4.6 Case Studies

3.4.6.1 Social Engineering

Figure 7 shows the *download path* for an in-the-wild social engineering attack, including the "link" relationships between nodes in the path. The user first performs a search on `www.youtube.com` (A) for a "*facebook private profile viewer*", which is the *root* of the path. Next, the user clicks on the top search result leading to a "*trick*" page on `www.youtube.com` (B), which hosts a video demonstrating a program that supposedly allows the viewing of the private profiles of Facebook users. A textual description under the video provides a link to download a "profile viewer" application through a URL shortener `goo.gl` (C). This shortened URL link redirects the user to `uploading.com` (D), a free file sharing site that prompts the user with a link to start the download. This leads to another `uploading.com` (E) page that thanks the user for downloading the file and opens a new `uploading.com` (F) page that includes an `<iframe>` with source `fs689.uploading.com` (G), from which the executable file is downloaded. The file is labeled as "Trojan Downloader" by some anti-virus scanners.

Notice that no exploit appears to be involved in this attack, and that the user (highly likely) had to explicitly click on various links and on the downloaded malware file itself to execute it.

3.4.6.2 Drive-by

Figure 8 shows the *download path* related to an in-the-wild drive-by download.

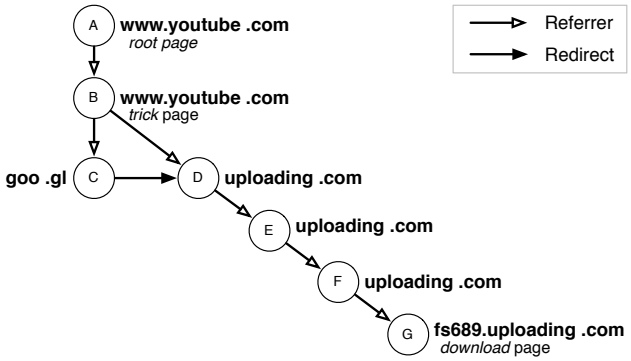

Figure 7: Social engineering download example.

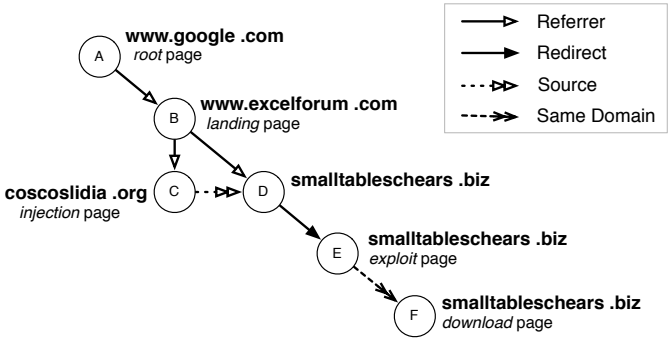

Figure 8: Drive-by download example.

The download path originates from (A) `www.google.com` (the *root* page), where the user entered the search terms "*add years and months together.*" The first link in the search results, which the user clicked on, is for a webpage (B) on `www.excelforum[dot]com` (the *landing* page). Sadly, the page the user landed on was compromised several days earlier, resulting in the addition of a `<script>` tag with source at `coscoslidia[dot]org`, which is the *injection* page. The script is automatically retrieved from (C) and executed, forcing an `<iframe>` to be added and rendered. The source of the frame (D) is on the site `smalltableschears[dot]biz`, from which the content is immediately fetched and included in the page. The newly loaded javascript served by (D) then checks for the presence of vulnerable versions of several browser plugins. It quickly matches a version of the installed Adobe Flash Player to a known vulnerability and dynamically adds another `<iframe>` to the page, which pulls a malicious Flash exploit file from (E) on the same `smalltableschears[dot]biz` site (the *exploit* page). The Flash exploit succeeds and the shellcode fetches a malware binary (labeled as ZeroAccess by some AVs) from (F) on the same domain `smalltableschears[dot]biz` (the *download* page).

3.4.7 "Origin" of Malware Download Paths

Figure 9 shows a breakdown of the drive-by and social engineering "origins" behind the malware downloads. For drive-by downloads, 64% of the download paths started with a search. We noticed that the search query keywords were typically very "normal" (e.g., searching for a new car, social events, or simple tools, as shown in the example in Section 3.4.6.2), but unfortunately the search results linked to hacked websites that acted as the "entry point" to exploit distribution sites and malware downloads.

For social engineering downloads, about 60% of the web paths started with a search. Search engine queries that eventually led to social engineering attacks tended

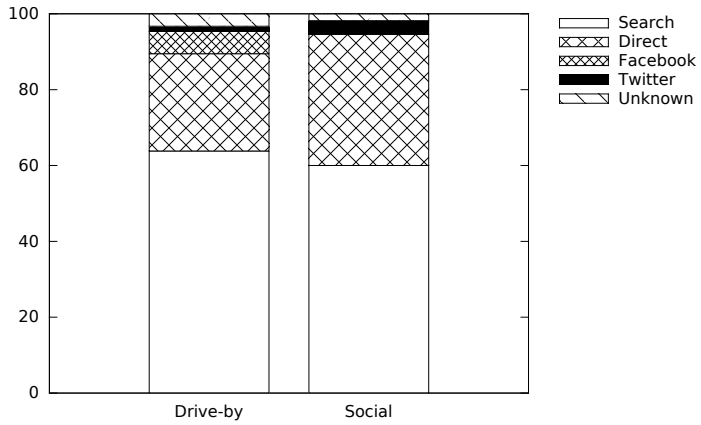

Figure 9: "Root" of malware download paths.

to be related to less legitimate content. For example, the search queries were often related to free streaming links, pirated movies, or pirated versions of popular expensive software. In these cases, the search results contained links offering content relevant to the search, but the related search result pages would also encourage the user to install malicious software disguised as some required application (e.g., a video codec or a software key generator).

The second most common origin is direct links, whereby a user arrives to a webpage directly (e.g., by clicking on a link within a spam email), rather than through a link from another site. Most of these direct links point to a benign website that is either hacked or displays malicious ads.

Facebook and Twitter represent a relatively infrequent origin for malware downloads (7% and 3% of the cases, respectively). While both Facebook and Twitter usually rely on encrypted (HTTPS) communications, we were able to determine if a download path originated from their sites by noticing that Facebook makes sure that all external requests carry a generic `www.facebook.com` referrer [70]. On the other hand, requests initiated by clicking on a link published on twitter carry a referrer containing a `t.co` shortening URL. During our entire deployment, we only observed one case in which a link from Facebook or Twitter led directly to a drive-by exploit

kit. In all other cases, the links led first to a legitimate page that was hacked or that displayed a malicious ad.

For the remaining malicious downloads (less than 3%, overall) we were unable to trace them back to their origin (e.g., due to missing traffic).

During our deployment, we also found that malicious ads are responsible for a significant fraction of the malware downloads in our dataset. Specifically, malicious ads were included in the web path of about 25% of drive-by and 40% of social engineering malware downloads. The malicious ads we observed were typically displayed on relatively unpopular websites. We observed only one example of a malicious ad served on a website with a US Alexa ranking within the top 500.

3.5 Discussion and System Limitations

Our system only collected data during off-peak hours because it was sharing hardware resources with a production network monitoring system whose functionality could not be disrupted. Thus, our data is just a sample of the malicious downloads that occurred during this period. Also, due to the significant efforts required to analyze complex malware download traces, our evaluation ground truth is limited to a representative sample of the malicious downloads that occurred in the monitored network. However, based on our extensive manual analysis, we believe the samples to be very diverse because of the various exploit kits, exploits, social engineering tricks and malware observed, and therefore representative of the overall set of malware downloads that occurred during our deployment.

Because the detection of malicious executable files is outside the scope of this chapter, we have relied on a "detection oracle" to extract malicious download traces from the network traffic. For the sake of this study, we have chosen to rely on multiple AV scanners. It is well known, though, that AV scanners suffer from false positive and negatives. In addition, the labels assigned by the AV are often not completely

meaningful. However, we should consider that using multiple AV scanners reduces the false negatives, and the set of filtering heuristics we discussed in Section 3.2.2 can mitigate the false positives. In addition, we used re-scanning over a period of a month for each of the downloaded executable files we collected, to further improve our ground truth. Finally, we used the AV labels to filter out adware downloads, because we are mainly interested in the potentially most damaging malware infections. We empirically found that the AV labels usually do a decent job at separating the broad adware and malware classes. Also, we manually reviewed all samples of malware downloads in our dataset, to further mitigate possible mislabeling problems.

Attackers with knowledge of our system may try to evade it by using a purposely crafted attack in attempt to alter some of the features we use in Section 3.3 to perform path traceback, categorization and for node labeling. Most likely, the attacker will have as a primary goal the evasion of our traceback algorithm. This, for example, could be done by forcing a "disconnect" between the final malware download node and its true predecessors. Such an attack theoretically may be possible, especially in case of drive-by attacks. In such events the browser is compromised and is (in theory) under the full control of the attacker. Now, if the malware download node is isolated in the reconstructed download path, the cause classifier may label the download event a malware update, thus preventing any further processing of the download path (i.e., any attempt to identify the exploit and injection domains).

However, we should also notice that most drive-by downloads are based on what we refer as "commonly exploitable" content in Section 3.3.3 (e.g., .jar, .swf, or .pdf files that carry an exploit). For such type of drive-by download attacks, the "commonly exploitable" content feature should connect the exploit and the download, if they occur in a small time window. If needed, the time window could be extended by requiring the domain severing the content to be young by checking its "age" before making a connection. Since the exploit must occur before the attacker has control of

the browser, it is more difficult to evade.

3.6 Related Work

Client honeypots actively visit webpages and detect drive-by downloads though observing changes to the system [2, 87, 89, 107, 108] or by analyzing responses for malicious content [7, 44, 93]. These systems tend to have a low false positive rate, but only find malicious websites by visiting them with exploitable browser configurations; also, they have limited range in the quantity of pages they can crawl because they are much slower than static crawlers. Often candidate URLs are selected by filtering content from static crawlers [107, 108, 121], using heuristics to visit parts of the web that are likely more malicious [39] or using search engines to identify webpages that contain content similar to known malicious ones [64].

A number of techniques have been developed to detect drive-by downloads through examining content [44, 46, 72, 112, 114]. Signature based intrusion detection systems, such as Snort [114], passively search network traffic content for patterns of known attacks. Both static [46] and dynamic [44] analysis of JavaScript has been used to detect attacks. The disadvantages of using content is that it is complex and under the control of the attacker. Polymorphic malware and code obfuscation results in missed attacks for signature and static analysis systems, and dynamic analysis can be detected by malware and subverted by altering its execution path [72].

Other systems focus on the redirection chain that leads to drive-by downloads. Stringhini et al [124] create redirection graphs by aggregating redirection chains that end at the same webpage. Features from the redirection graph and visiting users are then used to classify the webpage as malicious or benign. Mekky et al [86] build browsing activity trees using the referrer and redirection headers as well as URLs embedded in the content. Features related to the redirection chain for each tree are extracted and used to classify the activity as malicious or benign. Li et al [80] apply page

rank from the dark and bright side of the web to a partially labeled set of redirection chains to separate benign and malicious web paths. They find the majority of malicious paths are directed through traffic distribution systems. Using features from the redirection chain, Surf [82] detects malicious websites found in search engine results due to search poisoning and WarningBird [78] identifies malicious webpages posted on Twitter. These systems focus on the redirection chain and features extracted from it to classify a web activity as benign or malicious. Whereas, WebWitness provides context to malicious downloads by reconstructing the full download path (not just the redirection chain), classifying the cause of the download (drive-by, social, update) and identifying the roles of the domains involved in the attack.

Static blacklists [6] of domains/URLs and domain reputation systems [28,29] identify malicious websites to prevent users from visiting them. Many of the domains on static blacklists are exploit and download domains that change frequently rendering them less effective. On the other hand, reputation systems only provide a malicious score for a domain and do not indicate their role or give context to an attack. By analyzing the structure of a malicious download, WebWitness can identify the type of attack and the domain roles; providing the highest value domains for blocking and reputation training data.

Web traffic reconstruction has been studied for example in [42, 94, 129]. WebPatrol [42] uses a client honeypot and a modified web proxy to collect and replay web-based malware scenarios. Unlike WebPatrol, WebWitness is not limited to drive-by downloads invoked through client honeypots and can provide context to drive-by and social engineering attacks on real users observed on live networks. ReSurf [129] uses the referrer header to build graphs of related HTTP transactions to reconstruct web-surfing activities. As discussed in Chapter 3 and evaluated in [94], this approach is very limited especially in reconstructing the entire download path of a malicious

executable. Lastly, ClickMiner [94] reconstructs user-browser interactions by replaying recorded network traffic through an instrumented browser. Its focus is on the user's behavior that led to a webpage; whereas, WebWitness identifies the cause and structure of an attack that led to a malicious download.

3.7 Conclusion

We proposed a novel incident investigation system, named WebWitness. Our system targets two main goals: 1) automatically *trace back and label* the chain of events (e.g., visited web pages) preceding malware downloads, to highlight how users reach attack pages on the web; and 2) leverage these automatically labeled in-the-wild malware download paths to better understand current attack trends, and to *develop more effective defenses*.

We deployed WebWitness on a large academic network for a period of 10 months, where we collected and categorized thousands of *live* malware download paths. An analysis of this labeled data allowed us to design a new defense against drive-by downloads that rely on injecting malicious content into (hacked) legitimate web pages. For example, we show that on average by using the results of WebWitness we can decrease the infection rate of drive-by downloads based on malicious content injection by almost *6 times*, compared to existing URL blacklisting approaches.

Even though WebWitness can categorize a malicious download as social engineering, we focused the majority of this chapter on understanding drive-by downloads and creating a defense for them. In the next chapter, we will use the web path and context provided by WebWitness to continue our study of malicious downloads, but with our full attention on studying, detecting and developing a defense for social engineering.

CHAPTER IV

TOWARDS MEASURING AND MITIGATING SOCIAL ENGINEERING MALWARE DOWNLOADS

4.1 Introduction

Most modern malware infections happen via the browser, typically triggered by drive-by download attacks [108] or social engineering [36]. In the previous chapter we used WebWitness to analyze drive-by downloads and there have been numerous studies that are centered on measuring and defending against them [44,58,92,124]. However, malware infections enabled by social engineering attacks remain notably understudied [106]. Furthermore, cyber-criminals increasingly aim their attacks against the weakest link, namely the user, by leveraging increasingly sophisticated social engineering tactics [88]. In this chapter, we examine social engineering downloads using the web paths and context provided by WebWitness.

There is a pressing need for a comprehensive study of social engineering malware downloads that can shed light on the tactics used in modern attacks. In particular, we need to build systems capable of continuously collecting and automatically reconstructing (or "explaining") recent in-the-wild SE malware attacks. At the same time, our research around SE tactics should assist in the development of training and awareness programs that aim to better educate users to recognize such attacks and reduce infection rates. In addition to better security awareness programs for users, we need to build technical solutions that can mitigate SE malware attacks without requiring the user to be in the loop.

To that end, we present a system developed to capture and analyze SE malware downloads *in live* networks. Specifically, we focus on studying *web-based SE attacks*,

namely SE attacks that unfold *exclusively via the web* and that do not require "external" triggers, such as email spam/phishing, etc. An examples of such attack is described in [36]: a user is simply browsing the web, visiting an apparently innocuous blog; his attention is drawn to an online ad that is subtly crafted to mimic a warning about a missing browser plugin; clicking on the ad takes the user to a page that reports a missing codec, which is needed to watch a video; the user clicks on the related codec link, which results in downloading malicious software.

Via a detailed analysis of hundreds of in-the-wild web-based SE malware attacks collected over a period of several months at a large academic network, we attain the following results: (i) we develop a categorization system that expresses how attackers typically gain users' attention, and which are the most common types of deception and persuasion tactics used to trick victims into downloading malicious applications; (ii) we reconstruct the *web path* followed by SE malware victims, and observe that a large fraction of SE attacks are delivered via malicious online advertisement (e.g., served via "low tier" ad networks); (iii) we measure the characteristics of the network infrastructure (e.g., domain names) used to deliver such attacks, and discover a number of features that can be leveraged to distinguish between SE malware and benign (or non-SE) software downloads.

One of our findings shows that a large fraction of SE malware attacks (almost 50%) are accomplished by *repackaging* existing benign applications. For instance, users often download free software that actually comes as a bundle including the software actually desired by the user plus some Adware or other Potentially Unwanted Programs (PUPs). This confirms that websites serving free software are often involved (willingly or not) in distributing malicious software [22, 25].

The second most popular category of attacks is related to alerting or urging the user to install an application that is supposedly needed to complete a task. For instance, the user may be *warned* that they are running an outdated or insecure

version of Adobe Flash or Java, and are offered to download a software update. Unfortunately, by downloading these supposed updates, users would instead receive malicious software. Similarly, users may stumble upon a page that supposedly hosts a video of interest. This page may then inform the user that a specific video codec is needed to play the desired video, for example. The user *complies* by downloading the suggested software, thus causing a malware infection (see Section 4.2 for details).

Another example is represented by fake anti-virus (FakeAV) attacks [111]. In this case, a malicious page alerts the user that their machine is infected, and that an AV software is needed to clean up the machine. In a way similar to the SE attack examples reported above, the user may be persuaded to download (in some cases after a payment) the promoted software, which will infect the user's machine. However, while FakeAVs have been highly popular among attackers in the recent past, in our study of in-the-wild SE malware downloads we find that they represent less than 1% of modern SE attacks. This sharp decline in the number of FakeAV attacks within the last few years is likely due to a combination of technical countermeasures [23] and increased user awareness [24].

As mentioned earlier, a large fraction of SE malware attacks (more than 80%) are initiated via malicious advertisement, and that the "entry point" to these attacks is represented by only a few low-tier advertisement networks. For instance, we found that a large fraction of the web-based SE attacks described above are served primarily via two ad networks: `onclickads[.]net` and `adcash[.]com`.

Guided by these discoveries, we build WebSentry, a novel network defense system aimed at automatically detecting and preventing SE malware downloads. WebSentry can be deployed at the edge of a network or as a web proxy module to monitor all inbound and outbound web traffic, and is able to accurately detect malware download events triggered by ad-based SE attacks.

Chaper Summary. This chapter discusses the following:

- We present the first systematic study of modern web-based SE malware download attacks. For instance, our analysis of hundreds of SE malware attack instances reveals that most such attacks are enabled by malicious online advertisement served through a handful of "low tier" ad networks.

- We find that the most common types of SE malware attacks include *fake updates* for Adobe Flash and Java, and that fake anti-viruses (FakeAVs), which have been a popular and effective infection vector in the recent past, represent less than 1% of all SE malware downloads observed in the wild. Furthermore, we find that existing defenses, such as traditional anti-virus (AV) scanners and popular URL blacklists, are largely ineffective against SE malware downloads.

- To assist the process of understanding the origin of SE malware attacks, we develop a categorization system that expresses how attackers typically gain users' attention, and what are the most common types of deception and persuasion tactics used to trick victims into downloading malicious applications. This makes it easier to track what type of attacks are most prevalent, and may help to focus user training programs on specific user weaknesses and particularly successful deception and persuasion tactics currently used in the wild.

- In addition, we present WebSentry, a network-defense systems that aims to detect web-based SE malware attacks in real time. We deploy WebSentry in a large academic network for a period of eight months, and show that our system is able to accurately detect SE malware download attempts with 91% true positives and only 0.5% false positives. To the best of our knowledge, WebSentry is the first network-based system capable of accurately detecting and mitigating SE malware attacks.

4.2 Collecting & Labeling SE Downloads

In this section we discuss how we collected and identified downloads that were the result of social engineering attacks on the web. In addition we explain how we label the downloads based on the different techniques used by adversaries to trick the user. Based on the observations from this study we define a categorization system for social engineering attacks in the following section.

4.2.1 Overview of Data Collection

We collect and extract the download path of executable file downloads using WebWitness. We deployed WebWitness to a large academic network serving tens of thousands users for a period of two months. During this period we collected a total of $35,638$ executable downloads that did not match our whitelist filter. The whitelist consists of 128 benign domains that were responsible for the majority of the benign executable downloads we observed on the network. We review these downloads in the next sections to identify the ones that are the result of a social engineering attack.

4.2.2 Automatically Filtering Update

The majority of executable downloads observed on a network are updates to software already installed on systems. We can automatically identify and filter out a large fraction of these by examining the length of the download path and the HTTP user-agent. Executable downloads that are updates tend to have very short paths. In fact, they typically consist of a single HTTP request to fetch a new executable.

The user-agent for an update download is often different from the one used by the browser [97]. This is because the application performing the update supplies the user-agent. We determine if the user-agent in an executable download belongs to the update using two methods. First, if there are buffered HTTP transactions from the same client prior to the download, we calculate each user-agent's popularity by the number of unique domains visited. We expect applications performing updates

to visit a small number of domains (typically only one) whereas a browser will visit many due to the fact that most webpages display content from other sites such as advertisements. So, even a user visiting a single webpage will likely create a number of browser HTTP requests to other domains.

Second, we check the user-agent to see if it contains the string "mozilla". Most browsers (FireFox, Chrome, IE, Safari, etc.) still add "mozilla" to their user-agent to help websites understand the browser's features and capabilities. However, many applications performing an update do not include it. Instead the user-agent contains the name of the application performing the update.

To automatically identify update downloads we use the following heuristic: 1) the length of the download path must be one, 2) the user-agent is unpopular or does not contain the string "mozilla". If both conditions 1 and 2 are met, we label the download as an update. Using this simple heuristic we are able to automatically identify almost 95% of the updates observed on the network. Note that this heuristic is evadable; however, our goal is not to filter out malicious updates from the network, which is small, but to quickly filter out the large number of benign updates from our study data so we can focus on social engineering downloads.

4.2.3 Clustering Downloads for Analysis

Filtering the updates reduced the total number of downloads by 61% leaving $13,762$ that require manual analysis for labeling. For the labeling task, we leverage unsupervised learning to maximize efficiency to group together related benign and social engineering downloads. Then we manually label a random sample of downloads and cascade the same label to the rest of the cluster. For this to be successful, the statistical features used for clustering must have the potential to differentiate downloads that are not related. Next, we introduce and briefly discuss the intuition behind these features:

(1) **Filename Similarity**: Benign file downloads that are from the same organization, entity or group tend to have similar filenames. This is also true for social engineering campaigns because the filename often aids in the *deception* of the end user. For example, having the word "adobe" in the filename of a fake flash player upgrade attack.

(2) **File Size Similarity**: Benign files that are identical or variations (i.e., by version) of the same software are usually very close in size. Social engineering campaigns typically infect victims with the *same* malware family. The sizes of the malicious executables per victim do vary due to polymorphism, but the size difference is typically small in respect of the total file size.

(3) **URL Similarity**: A website that is benign will often host all of its executable downloads at the same or very similar structured URLs. Social engineering campaigns often go weeks or even months before a noticeable change in the structure of their URLs is observed. On the other hand, the domains and IP addresses that facilitate their illicit activities tend to change more frequently — so they can avoid the domain name and IP blacklisting.

(4) **Domain Name Similarity**: Files downloaded from the same domain tend to share both file reputation and reason for the actual download. This is true for both benign and malicious downloads. Some social engineering campaigns will reuse terms in their second and third level domains that aid in deceiving the user (e.g, "security", in the case of some Fake AV campaigns).

(5) **Shared Predecessor**: Social engineering attacks that share a common node (or predecessor) in the download path are often related. For example, adversaries performing a social engineering attack, will ofter lure victims by posting links to a web forum or exploit an ad network with weak anti-abuse practices. Now, the actual download domains used in the attack may change (i.e., to avoid blacklisting), however the "tactics" employed by the adversaries to attract the users

attention are often the same. In benign cases, both the download and attention grabbing domain tend to be stable, as the main goal is quality of service towards the end user.

(6) **Shared Hosting**: Social engineering campaigns often reuse the same hosting network after they switch domains. Hosting networks that tolerate abuse (knowingly or otherwise) is a rare and costly resource. On the other hand, domain names are significantly easier to obtain (often without yielding any information that can be used for attack attribution) and can be used as crash-and-burn resource from the adversary. Benign websites do not change hosting very frequently for, again, quality of service but also brand protection reasons.

(7) **HTTP Response Header Similarity**: The headers in an HTTP response are the result of the installed software and configuration of the web server. The set of response headers and their associated values offer a lot of variation. However, most of the web servers for a benign site tend to have common configurations so they respond with similar headers. Also, social engineering campaigns tend to use the same platform and do change their configurations even when they move domains.

For each of the$13,762$ downloads we calculate the pairwise distance between the downloads using the previously descried features. We apply an agglomerative hierarchical clustering algorithm to the derived distances. We chose a conservative cut height θ to error on the side of not grouping related downloads instead of potentially grouping unrelated ones. This process produced $1,205$ clusters resulting in an order of magnitude reduction in the number of items that require manual inspection to label.

4.2.4 Labeling Clusters

For each cluster we randomly sample 10% of the downloads for manual review. For small clusters we sample a minimum 5 downloads and for clusters with < 5 all are reviewed. Our goal is to label each cluster as "likely" benign, social engineering, drive-by download or update. In addition to reviewing the download paths to label a cluster, we use antivirus (AV) labels for the downloaded executables as indicators of being malicious. To increase AV detections we age the downloads for a period of two months before scanning them with more than 40 antivirus (AV) engines using virustotal.com. If we suspect a cluster is malicious, having one or more downloads labeled by AV offers additional confirmation.

We perform an extensive review of each download in our sample from each cluster. First we look for attributes that suggest the download is an update because they require less analysis to confirm. Even though our heuristics in Section 4.2.2 eliminated the majority, some still remain. To determine if a download is an update, we examine the length of the download path and the time between requests. If the length of download path is < 4 or the time between requests is < 1 second, we review the content of the first non-redirect HTTP transaction immediately preceding the download. If it does not contain content suitable for human consumption we label the download as an update.

Next we look for drive-by download indicators [97]. For instance we look for content such as pdf, flash, and java on the path just prior to the download. Browser plugins and extensions that process this type of content often have vulnerabilities that are exploited by attackers. If we suspect it is a drive-by, we inspect the content of the HTTP transactions that precede the suspected attack. This typically requires reverse engineering and deobfuscating javascript. If we identify code that checks for vulnerable versions of browser software, we label the download as drive-by.

If the cluster is not related to update or drive-by downloads, we further examine

the samples to determine if the they are the result of social engineering. For this analysis, we inspect the content of all non-redirect HTTP transactions on the download path. Our goal is to identify the page that contains the link the user likely clicked that ultimately resulted in the download. Since social engineering is an attack on the user, interaction (e.g., clicking a link) is required. Once identified the content can be reviewed to determine if *deception* or *questionable persuasion* techniques were used to trick the user into downloading the binary. If found, we label the cluster as social engineering; otherwise, we label it as "likely" benign.

For the majority of the clusters we label as social engineering, one or more of the downloads are labeled malicious by AV. This provides additional confirmation of our classification. The clustering also aided in the labeling process by allowing us to examine related downloads as a group versus having to label a single download in isolation. From the $1,205$ clusters we label 136 as social engineering giving us a total of $2,004$ such downloads. Analysis of the social engineering clusters allows us to derive a categorization system for further classification of social engineering attacks. We discuss this in the following section.

4.3 SE Download Attack Categorization

The $2,004$ social engineering downloads we labeled in the previous section contain a wide range of depiction and persuasion techniques as well as numerous tactics employed to victimize users. We develop our categorization system by studying the techniques used in these successful attacks that trick real users into downloading and installing malicious software. The categorization system allows us to label different download paths according to; (1) the ways the adversaries get the user's attention and (2) the type of deception and persuasion techniques employed. Our categorization system is shown in Figure 10.

The first step in a social engineering attack is to get the user's attention. This is

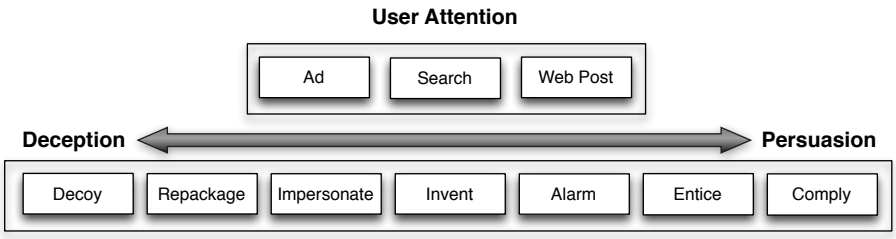

Figure 10: Categorizing social engineering malware downloads on the web.

accomplished using advertisement (ad), search result, web post or a combination of these three. As we will show in our measurements, the most popular of these methods are ads. On-line advertisement allows the attacker to "publish" their malicious ad on a site that is already popular among the targeted victims. In addition, ads help hide (from the average Internet user and security researchers) the malicious campaign and attack infrastructure, simply because they are exposed only to the users that trigger the delivery of the ad (according to their search key words, cookies, referrer, user-agent etc.).

Another method employed to the get the user's attention is search. Search engines get abused through techniques such as black hat search engine optimization. However, we do not limit our definition of search to just search engines. Anytime, a user performs a query to locate specific content on a website we classify it as search. For example, we have observed users become victims of SE attacks while searching for content on a video hosting website.

Web posts are also utilized by attackers to get the user's attention. We define a web post as content that has been added to a website by a visitor and is now available for display to others. Many of the web posts we observed were located within groups of legitimate posts about a topic of interest. The majority of social engineering web posts were related to free software, books, music and movies.

It is not uncommon for these techniques to be combined. For instance, attackers will combine search and ad to get the user's attention. Search engine ads related to the search terms are often displayed before the real results thus increasing the

likelihood of a click. This is abused by attackers. Also, users will search web forums and fall victim to a malicious web post.

After the attacker gets the user's attention, they must convince them to download and install malicious software. This typically involves combining a subset of the deception and persuasion techniques shown in Figure 10. As one scrolls from left to right in the figure, the techniques move from deception towards persuasion. However, none of the techniques involve only deception or persuasion; instead, the different techniques vary in their levels of each. The following is a description of the deception and persuasion techniques:

(1) **Decoy**: An object such as a hyperlink that is purposely placed at a location on the page that will attract users to it and away from the actual object desired by the user. An image of a download button delivered as an ad on a free download site located prior to the actual download link is an example of this technique.

(2) **Repackage**: A benign and malicious executable grouped together and presented to the user as a single download or install. An example, is adware bundled with a benign application downloaded from a free software distribution website.

(3) **Impersonate**: Using specific images, terms and colors to make a malicious executable appear to be a known popular benign application. Also, claiming that an executable provides features or services, but has no intention to supply them, as a way to get the user to download and install the application. Malicious executables pretending to be a Flash Player update by using words like "adobe" and "flash" along with Flash Player images and graphics is an example.

(4) **Invent**: Creating a false reality for the user that compels them to download a malicious executable. For example, alerting the user stating that their machine is infected with malware and instructing them to download a malicious executable, pretending to be AV software, to clean up the fake infection.

(5) **Alarm**: Using fear and trepidation to scare the user into downloading a malicious

executable that promises to safeguard them. An ad claiming that the user's browser is out-of-date and is vulnerable to exploitation is an example.

(6) **Entice**: Attracting users to download a malicious executable by offering features, content or advantage. As an example, a user is displayed an ad for a system optimization utility, that is really a malicious executable, stating that it will "speedup" their PC.

(7) **Comply**: Requiring or appearing to require the installation of a malicious executable before the user can continue or get what they want. A user that is prompted to install a necessary "codec" before they can watch a free movie is an example.

It is important to note that none of the social engineering attacks in our study fall into a single class. Instead they use techniques across two or more of the above categories to trick the user into infecting themselves.

4.3.1 SE Download Classification Examples

In this section we present two SE examples from our observations and classify them using our categorization system. To aid in our discussion we define the notation "a[+b]:1[+2+3]" where the letters are ways of getting the user's attention and the numbers are the deception/persuasion techniques. For example, if a malicious ad uses the deception/persuasion techniques alarm and impersonate then we label it using our notation as ad:alarm+impersonate.

Example A. User searches for "gary roberts free pics" using a popular search engine. A page from a compromised website is returned as a top result. The page contains various content referring to "gary roberts", but it is incoherent and likely only present for blackhat search engine optimization (SEO). However, the user never sees the content because javascript located at the top of the page immediately closes the document then reopens it to inject a script that redirects the user to a page that

says "gary-roberts-free-pics is ready for download. Your file download should automatically start within seconds. If it doesn't, click to restart the download." But the downloaded file is not pictures instead it is malware.

Using our categorization system we classify this attack as "search:entice+decoy+impersonate." Search as the method of gaining the users attention in this example is obvious because the social engineering page appeared in the results of a search engine. The entice part of the attack is the offering of "free" pics of the subject of interest. Decoy is due to the fact that blackhat SEO was used to elevate the social engineering page in the search results above other legitimate pages. Lastly, what the user downloads is not pics of gary roberts; instead, it is a malicious executable impersonating what the user wants.

Example B. A user is watching an episode of "Agents of Shield" on a free video website when they are presented with an ad. The ad, shown in Figure 11, presents the user with the option of downloading an early warning system for Ebola. However, the download does not warn them of an outbreak; instead, it infects the user's system with malware.

We classify this attack as "ad:alarm+impersonate" using our categorization system. The user's attention is gained through an ad, where their fear of Ebola is used to alarm the user into downloading a tracking system. But, what the user downloads only impersonates a tracking systems and is really malware.

4.4 SE Attack Download Measurements

In this section we measure the popularity of the SE user attention and deception/persuasion techniques that convince users to download malicious software. In addition we measure properties of the SE advertisement paths and the downloads themselves. Finally, we examine how AV and current blacklisting approaches detect SE attack downloads.

Figure 11: Social engineering ad for Ebola early warning system.

4.4.1 Prevalence of Attacks

Table 7 shows the number and percentage of downloads for each technique used to get the user's attention. Over 87% of the SE attacks use ads to get the user's attention with 80% being displayed on websites visited by the user. The other 7% of SE ads is a combination of search and ad where the user queries a search engine and is then presented with ads based on the search terms as part of the results. Ads are popular with SE campaigns because it is an efficient way to get the malicious page displayed to the most users.

Table 7: Popularity of SE techniques for getting the user's attention.

User's Attention	Total	Percentage
Ad	1,616	80.64%
Search+Ad	146	7.29%
Search	127	6.34%
Web Post	115	5.74%

Gaining the user's attention is not sufficient to infect them. They must be tricked into downloading and running the malicious executable. Table 8 shows the total and percentage for each technique. The most popular, making up over 48% of the observations, is repackage+entice. It is popular because it is primarily composed of downloads for "free" software that delivers adware or possible-unwanted-programs (PUPs) in addition to what the user desires.

Table 8: Popularity of SE techniques for tricking the user.

Trick	Total	Percentage
Repackage+Entice	972	48.50%
Invent+Impersonate+Alarm	434	21.66%
Invent+Impersonate+Comply	384	19.16%
Repackage+Decoy	155	7.74%
Impersonate+Decoy	46	2.30%
Impersonate+Entice+Decoy	12	0.60%
Invent+Comply	4	0.20%
Impersonate+Alarm	1	0.05%

The next two most popular categories are invent+impersonate+alarm and invent+impersonate+comply comprising 22% and 19% of the SE downloads observed. An example of an invent+impersonate+alarm technique is a fake java update where the user is shown an ad stating "WARNING!!! Your Java Version is Outdated and has Security Risks, Please Update Now!" and uses images associated with java. Ads like this are typically presented to users while they are visiting legitimate websites. In this example, the attacker is *inventing* the scenario that the user's java is out-of-date, *alarming* them with "WARNING!!!" displayed in a pop-up ad and *impersonating* a java update for download to resolve the issue.

The difference between invent+impersonate+alarm and invent+impersonate+comply is the persuasion component; i.e., alarm versus comply. Alarm uses fear, e.g. computer may be compromised, to compel the user to download and install malicious software; whereas, comply imposes a requirement on the user in order to proceed. An example of invent+impersonate+comply is an ad on a free video website that says "Please Install Flash Player Pro To Continue. Top Video Sites Require The Latest Adobe Flash Player Update." In this example the attacker is *inventing* the requirement to install flash player pro and tells the user they must *comply* by downloading a malicious executable *impersonating* flash play before they can continue.

Table 9 shows the popularity of each scam tactic in the invent+impersonate subclasses alarm and comply. Fake flash and java updates are the two most popular in the alarm class. Also, we observed fake browser updates and fake av alerts, but they were much less common, each comprising less than 1% of our observations. Fake

Table 9: Popularity of different scam tactics in the Ad:Invent+Impersonate subclasses.

	Alarm	Comply
Fake Flash	68%	20%
Fake Java	30%	0%
Fake AV	1%	0%
Fake Browser	1%	0%
Fake Codec	0%	22%
Fake Player	0%	58%

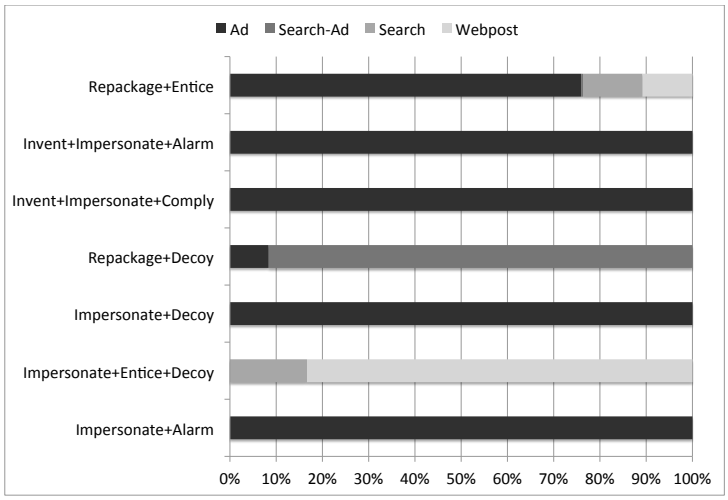

Figure 12: How attackers gain the user's attention per deception/persuasion technique.

flash updates are also common for the comply class; however, the most popular scam tactic is telling the user that a "video update" is required to continue. In these fake player ads, images that resemble flash are used, but not the terms "adobe" or "flash." Lastly, the requirement to install a fake codec in order to watch a video is still tricking users into installing malicious software.

Figure 12 shows the how attackers get the user's attention for each deception/persuasion technique. For instance, ads were the most common technique used in repackage+entice attacks making up 75% of their observations with search and web post contributing the remaining 25%. All of our observations for invent+impersonate+alarm,

invent+impersonate+comply, impersonate+alarm and impersonate+decoy only use ads to get the user's attention.

4.4.2 SE Download Advertisement Paths

The ad path begins at the first HTTP transaction on the download path that is responsible for ad delivery and ends at the download. We identify ads using regular expressions automatically derived from the rules of the popular Adblock Plus browser extension. To measure their effectiveness we compare detections against the set of SE ad downloads we manually labeled. We find that they correctly identify almost 85% of them. For our measurements we use the set of SE ad downloads identified using these regular expressions, and focus on the ad:repackage+entice (entice), ad:invent+impersonate+alarm (alarm) and ad:invent+impersonate+comply (comply) SE classes because they are responsible for over 90% of the ad based SE downloads we observe.

The ad entry point is the first HTTP transaction on the ad path. We label the first HTTP transaction on the download path that matches an ad regular expression the ad entry point. In most cases, the true ad entry point and the one we label are the same, but they can differ due to a missed detection. We measure the popularity by the percentage of downloads that have the same ad entry point domain.

Table 10: Top five ad entry point domains per class by percentage of downloads.

Comply	Alarm	Entice
26% onclickads.net	16% adcash.com	20% doubleclick.net
10% adcash.com	7% onclickads.net	16% google.com
10% popads.net	7% msn.com	12% googleadservices.com
7% putlocker.is	6% yesadsrv.com	11% msn.com
3% allmyvideos.net	4% yu0123456.com	8% coupons.com

Table 10 shows the top 5 domain ad entry points for the comply, alarm and entice classes. The top 3 ad entry points for the comply class are domains that have been abused by adware to inject pop-up advertisements into the user's browsing experience. To determine if these downloads are due to adware pop-ups, we select 10

random samples from the 3 adware abused domains on our list (i.e., onclickads.net, adcash.com, popads.net) and manually examine the content of the HTTP transaction that precedes it. If a request is the result of adware, there should not be a matching URL in the preceding webpage since the ad would have been injected locally at the host. For all of the reviewed samples, we are able to identify the preceding webpage as the source of the ad; thus, the these download are not the result of adware pop-ups.

The top two ad entry points for the alarm class match the comply class, but are in reverse order. The third domain msn.com has a good reputation, but is probably being abused by less reputable ad networks several redirects later at the end of the download path. The top entry domains in the entice class all have very good reputations. This is likely due to the fact that the majority of downloads in this class are for legitimate software that is bundled with PUPs. The domain coupons.com makes the list because it is one of the most popular software downloaded in this class.

Table 11: Top five ad download domains per class by percentage of downloads.

Comply	Alarm	Entice
17% softwaare.net	7% downloaddabs.com	41% imgfarm.com
5% newthplugin.com	4% downloaddado.com	17% coupons.com
5% greatsoftfree.com	4% whensoftisupdated.net	11% shopathome.com
4% soft-dld.com	3% safesystemupgrade.org	5% crusharcade.com
3% younplugin.com	3% onlinelivevideo.org	3% ilivid.com

At the end of the ad path is the download. Table 11 show the most popular download domains for the comply, alarm and entice classes. All of the domains listed for the comply and entice classes are known to serve malicious software with most being adware and PUPs. We measure the age of these domains using a large passive DNS (pDNS) database that stores historic domain name resolutions. We define the domain age as the difference in days from the time it was first recorded in pDNS to the day of the download. All the domains in table that are part of the comply and alarm classes are less then 200 days old with the majority being less than 90 days.

The domains in Table 11 for the entice class are all at least several years old. This is because most of the downloads in this class are for legitimate software that is

bundled with adware or PUPs. For instance, we find a large variety of "free" software that directs users to the domain imgfarm.com for download. This is the reason over 40% of the downloads in the entice class are from that domain.

The middle domains, the ones between the ad entry point and the download on the ad path, tend to be a mix of young and old. In fact, the most popular comply and alarm class middle domains are a 50/50 split of young and old. But, this is not the the case for the entice class where all them are several years old. However, the majority of ad paths for all three classes have at least one middle domain with an age that is less than 200 days.

4.4.3 Downloads Detected by Antivirus

We measure the percentage of downloads that are malware, adware and potentially-unwanted-programs (PUPs) for downloads in the comply, alarm and entice classes. For this measurement the downloads are aged for a period of one month before performing the AV scan to collect AV labels. First, we automatically separate binaries based on strings found in the AV labels. For adware we look for the string "adware" and the names of popular adware applications. We perform the same matching for PUPs looking for the strings "PUP", "PUA" and popular PUP applications. The executable is labeled based on majority matching. If there is a tie in matching between adware and PUP we label the executable as PUP. The majority of binaries can be classified using these heuristics. Most of the ones that remain are malicious, but we manually inspect the labels to verify.

Figure 13 shows the percentage of downloads that are malware, adware and PUP for the comply, alarm and entice classes. The majority of detected downloads in the comply and alarm classes are adware. Only a small amount of malware 3.2% and 2.4% are detected in the comply and alarm classes. This is expected because adware is much more common than malware. The majority of labeled downloads in the entice

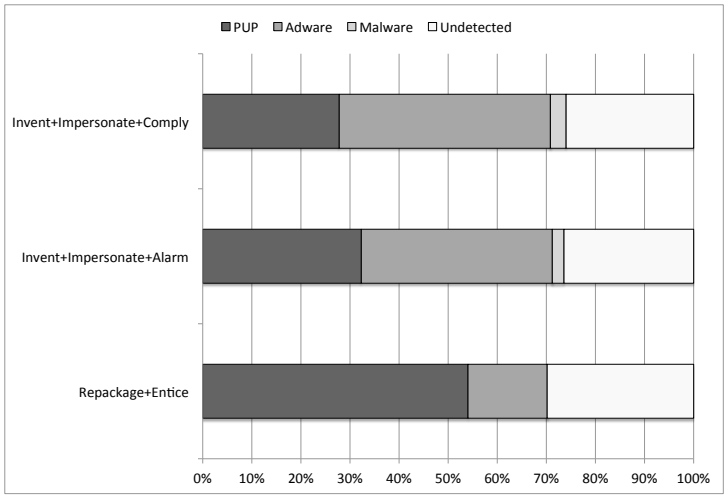

Figure 13: AV detections one month after download.

class are PUPs and no malware was detected. Most of the downloads in the class are for legitimate software that are bundled with PUPs/adware. Lastly, between 70% and 75% of the binaries were labeled by AV after aging them for one month. We suspect that these binaries are also malicious because their downloads are in clusters with AV labeled malicious binaries.

On the day of the download we scanned each executable with AV software. Figure 14 shows the percentage of binaries that would eventually be labeled as malware, adware and PUPs that were detected by AV on the day of the download. The highest percentage of detections across all three social engineering classes was PUPs followed by adware. None of the malware downloads for the comply class were detected on the day of the download. Overall, only about a third of the binaries that would be label by AV were detected on the download day.

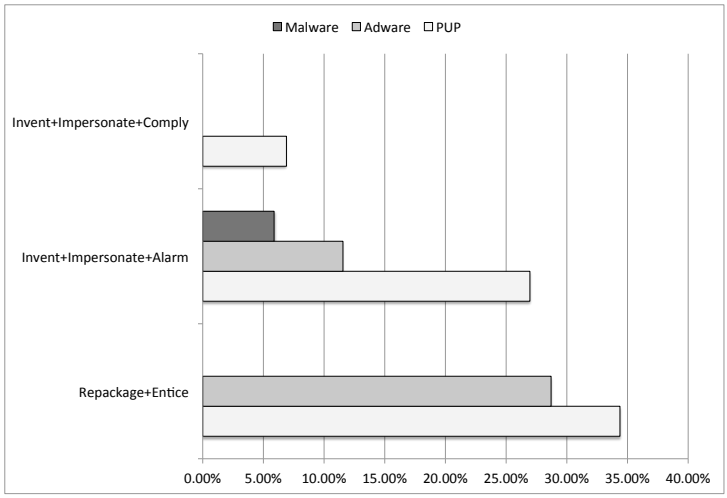

Figure 14: Percentage of AV detections per ad social class on the day of download.

4.4.4 Blacklisting SE Attack Downloads

In this section we measure the types of SE attack downloads that are currently blocked by blacklists. For our blacklist we chose Google Safe Browsing (GSB) because it commonly used in browsers to prevent users from accessing malware sites. We checked each URL on the download path for all downloads collected. It is important to note that this evaluation was performed in the months following our data collection so it is likely that the GSB results do not reflect the true detection rate since some malware URLs may have been aged off the blacklist reducing detections. For the downloads that are detected we use our label to determine if it is social engineering or benign. If it is in the social engineering we determine its class.

Table 12: Google Safe Browsing detections by class.

Class	Downloads	Detections
Likely Benign	10,086	0
Ad:Invent+Impersonate+Comply	384	0
Ad:Invent+Impersonate+Alarm	434	26
Ad:Repackage+Entice	738	2

Over 90% of the GSB malware downloads were in the ad:invent+impersonate+alarm

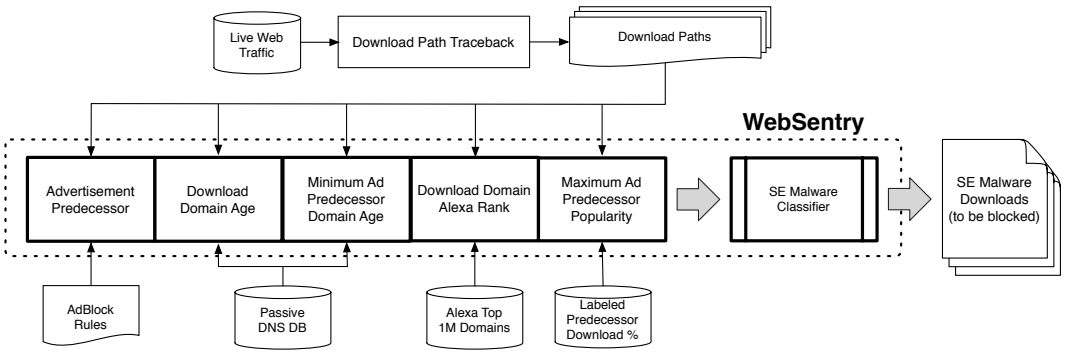

Figure 15: WebSentry system overview.

class. All but one of these was fake flash with the other being fake java. The two detections in the ad:repackage+entice class were for software offering to "scan your PC for free" and "repair Window's errors". None of the downloads in any of the other classes including benign were detected by GSB as malicious.

4.5 Detecting SE Malicious Downloads

Guided by the results of our study of in-the-wild SE malicious downloads, we develop a system called WebSentry that aims to automatically detect the most prevalent types of SE attacks. Specifically, it is designed to detect ad-driven attacks because more than 80% of SE malicious downloads we observed in our study are the result of an ad (see Table 7). WebSentry's components and features are summarized in Figure 15.

The input to WebSentry is a download path. The download path is the sequence of URLs traversed by the victim's browser as the user navigated to the malware download page. The download path is reconstructed as using WebWitness described in the previous chapter.

Features are extracted from the download path for classification. They were selected based on the measurements performed in Section 4.4.2 and are described next:

- **Advertisement Predecessor**. If the download path contains an ad, the value is 1 otherwise it is 0. This feature is calculated by matching regular expressions

automatically derived from the rules written for the popular AdBlock browser extension against URLs on the download path. *Intuition*: most benign downloads are not the result of an ad, but they are used in the majority of SE downloads. In fact, we measure our study data and find that less than 6% of the "likely" benign downloads are the result of an ad.

- **Download Domain Age**. The number of days between the download and the first time we observed the effective second level download domain resolve in pDNS. *Intuition*: the vast majority of benign downloads are from domains that have been active for a year or more because it takes time for a website to establish itself and attract visitors. On the other hand, SE domains are often young because they get blacklisted. Our study data shows that over 80% of the comply and alarm class download domains are less than 1 year in age, but that is true for less than 5% of the "likely" benign downloads.

- **Minimum Ad Predecessor Domain Age**. The minimum domain age for an ad predecessor on the download path. We measure the age of each domain on the ad path and use the minimum calculated age as the feature. *Intuition*: ad networks that consistently direct users to malicious ads are often blacklisted so they move to new domains. Minimum predecessor age is way of measuring the reputation of the ad path. By measuring our study data we find that the majority of ad paths for the comply, alarm and entice class all have domains less than 1 year in age. This is true less than 5% of the "likely" benign class ad based downloads.

- **Download Domain Alexa Rank**. The Alexa rank of the download domain. We measure this features using the effective second level download domain and the Alexa top 1 million list. *Intuition*: malicious executables are more likely to be hosted on unpopular domains because they do not want to be detected

and move domains when they are discovered. Also, we expect popular benign downloads to be hosted on popular domains because they are popular. Measurements on our study data show that over 60% of the benign downloads are from domains with an Alexa rank of $100,000$ or below. The more malicious social engineering downloads, such as those from the alarm class, are from very unpopular domains (very few are even in the top 1 million). Whereas, social engineering downloads that are typically PUPs (e.g., entice class) fall somewhere between in terms of domain popularity.

- **Maximum Ad Predecessor Popularity**. The percentage of SE downloads that share an ad predecessor with an identical effective second level domain. Using the study data we count the number of downloads per domain for both the SE and "likely" benign classes. For domains that are found in more than 1% of the downloads of at least one class, we store them in a lookup table. We calculate this feature by checking the table with domains on the ad path. If the domain is found and more than 1% of the "likely" benign downloads have the domain, it is discarded. Otherwise, we store the SE download percentage. The maximum is used for the feature. *Intuition*: there are some ad networks are more abused than others due to policy and controls; thus, appear more frequently in the download path of SE downloads. Table 10 in Section 4.4.2 shows popular ad entry points for SE downloads.

To construct the SE classifier we use Random Forest [37]. Random Forest is an ensemble learning technique that uses many decision trees, each with a random subset of features, for classification. This helps prevent overfitting that is common for decision trees. We use Random Forest to predict the probability that a download is malicious. This allows us to control the true and false positive rate by adjusting a threshold.

To detect SE downloads, WebSentry monitors live network traffic for executable downloads. When one is discovered, the download path is extracted as described in Section 4.2.1. The features discussed above are computed and input to the SE classifier. The output is the probability that it is malicious. If it exceeds our threshold, WebSentry alerts the user.

4.6 Evaluation

In this section, we evaluate the efficacy of WebSentry in detecting SE downloads. Using downloads collected in the months following our study and system development, we explore classification, feature importance and model selection. In addition, we examine adding a content feature and measure how well the system performs at detecting new SE campaigns.

4.6.1 Ground Truth

To evaluate our SE malware classifier, we collected two separate datasets. The first dataset, $\mathcal{D}_1$, which we use to train WebSentry, consists of malware downloads that occurred during our study of in-the-wild SE malware attacks, as described in Section 4.2. The second dataset, $\mathcal{D}_2$, consists of new SE malware downloads that were collected from the same deployment network in the *three months following* the completion of our initial study (Section 4.2) and *after the design of WebSentry was complete*. Namely, both the feature engineering and the training of WebSentry were completed with no access to the data in $\mathcal{D}_2$. Therefore, we use $\mathcal{D}_2$ as test data to evaluate the accuracy of WebSentry.

We label the download events in datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ using the following labels:

1. **Benign**: This set consists benign executable downloads. To create it we collected downloads from clusters that had no AV detections on the downloaded executable. In addition, we sampled downloads from each AV clean cluster

and manually analyzed the download path to confirm they were not malicious downloads. This set forms our negative class and we use it to measure the false positive rate of our classifier.

2. **Ad-Based**: This set consists of SE downloads where the user's attention was gained using an advertisement. However, we do exclude downloads from this set that are in the ad:repackage+entice class that have a download domain where the effective second level domain is one of the top five domains observed in our study. We exclude these because more than 90% of the downloads in that class are from one of those domains. Including them artificially raises the accuracy of our classifier. Downloads from these domains are adware and, if desired, can easily be detected and blocked using a blacklist. This set is our positive class and we use to measure the true positive rate of our classifier.

3. **Non-Ad-Based**: This set consists of social engineering downloads that are NOT in the Ad-Based set because the user's attention was gained through a search or web post. The social engineering classifier models social engineering downloads that use advertisements to get the user's attention. Therefore, this set is not part of the positive class; however, they are social engineering downloads and we use this set to see how the classifier performs on them.

4. **Entice**: This set consists of downloads in the SE class ad:repackage+entice. However, as discussed in the Ad-Based set description above, downloads in the ad:repackage-entice class that have a download domain where the effective second level domain is one of the top five domains observed in our study are excluded. This is a subset of the Ad-Based set and is used to break the positive class into specific classes of attacks.

5. **Comply**: Downloads that are ad:invent+impersonate+comply comprise this set. This is a subset of the Ad-Based set and is used to break the positive class

into specific classes of attacks.

6. **Alarm**: Downloads of the SE class ad:invent+impersonate+alarm form this set. This is a subset of the Ad-Based set and is used to break the positive class into specific classes of attacks.

4.6.2 SE Classification

WebSentry classifies downloads as social engineering or not using features from the download path. The features used by the SE classifier are designed to identify ad based SE downloads, which are responsible for more than 80% of the social engineering downloads we observed in our study. For this part of the evaluation we use the benign set for our negative class (NOT social engineering) and the ad-based set for our positive class (social engineering). Table 13 reports the confusion matrix for the social engineering classifier.

Table 13: Social Engineering Classifier - confusion matrix.

	Predicted Class	
	Benign	Ad-Based
Benign	9,711	49
Ad-Based	63	655

The classifier correctly identified over 91% of the ad based SE downloads. Furthermore, it has a very low false positive rate of 0.5%. This results in 93% of all detections being true positives even though there are 13 times more benign downloads. If we run the social engineering classifier on the non-ad-based set, 37% are detected as social engineering. The majority of these detections are due to very young download domains (< 30 days old) and a predecessor domain popular with downloads in the ad-based training set.

Table 14: Social Engineering Classifier - subclass performance.

	True Positives	False Negatives
Entice	63 (65%)	34 (35%)
Alarm	412 (98%)	9 (2%)
Comply	180 (90%)	20 (10%)

Figure 14 shows the performance for the subclasses of the ad-based set. The alarm and comply classes have 98% and 90% true positive rates respectfully. Both classes perform very well and are close to the overall class performance of 93%. The entice class, however, performs significantly below the overall classifier. The lower performance is due to downloads of legitimate software from well established domains that are bundling potentially-unwanted-programs (PUPs). There are many websites that offer free legitimate software and use bundled PUPs to generate revenue. As discussed in Section 4.6.1, we did not include downloads from the top five domains of the ad:repackage-entice class in the entice set. PUP downloads from those domains are responsible for over 90% of the PUP downloads and can be prevented by simply blacklisting the domains.

4.6.3 Content Features

Unlike other attacks such as drive-by downloads, the filename in a social engineering attack is an important component because it is shown to the user. Often, it plays a role in the deception by using terms from the software the attack is imitating and at a minimum it should not raise the user's suspicion.

Table 15: Popular filename tokens.

Comply	Alarm	Benign
player	setup	setup
flash	flash	loader
flashplayer	adobe	32
setup	setup.exe	network
setup.exe	player	wizard
chrome	hd	installer
ie	video	update
media	java	64

Table 15 shows the top 8 tokens in terms of download percentage extracted from the filenames for comply, alarm and benign classes. Notice that popular tokens extracted from benign download filenames are very different from the two social engineering classes. This holds for the vast majority of tokens that are common in more than 1% of the observations. However, there is some overlap such as "setup", which

is found in all three classes. For the benign class "setup" is typically a component of a larger filename such as "xxxxxxx-setup.exe", whereas "setup.exe", as can be seen from Table 15, is a common filename for the two social engineering classes. Using a simple filename like "setup.exe" helps to minimize user suspicion.

We also observe tokens from the social engineering filenames that aid in the deception like "java", "flash" and "adobe". For instance, if the user is told their flash player is outdated having the terms "flash" and "adobe" in the filename aids in convincing the user that the update is legitimate. We do not expect benign downloads to contain tokens from popular software packages because they are not trying to appear to be those packages. Specially, we only expect those names to appear in the filenames of downloads from the vendors that distribute the software.

Before we can tokenize the filename we must extract it from the request. In HTTP, the filename can appear in the path of the URL or the content-distribution header. Once it is extracted we split it into tokens. First the filename is partitioned into substrings by splitting on common separators such as "_", "-" and ".". Next, for each substring, we start with the first character and find the longest prefix consisting of a minimum of two characters that matches a word in a large english dictionary. If there is no match, we move to the next character and repeat the algorithm. When a match is found the word is our token. On a match, one additional token is created for characters preceding the match that are not part of any tokens. When the algorithm reaches the end of a substring, all preceding characters that are not part of a token become the finial one.

Once the tokens are extracted, we assign each a class popularity based on the percentage of filenames the token appeared in a given class. All tokens that have greater than a 1% popularity are saved for feature calculation. To calculate the filename feature we tokenize the filename as described above then find the token with the highest popularity for the social engineering class that is not in the benign set.

The popularity of that token is the feature.

Table 16: Social Engineering Classifier + filename feature - confusion matrix.

	Predicted Class	
	Benign	Ad-Based
Benign	9,709	51
Ad-Based	51	667

Table 16 shows the results of the social engineering classifier with the additional filename feature. The true positive rate is 93%, which is almost a 2% improvement, with only 2 additional false positives. We did not include this feature in the set defined in Section 4.5 because we believe this feature is more evadable than the others because the attacker has complete control over it, even through it often plays a role in the deception.

Increasing the true positive performance by adding an additional feature is useful. However, we do not want the classifier to perform poorly if the attacker attempts to evade this feature. To determine the performance impact of filename feature evasion on the classifier, we use the same training set, but modify the positive class in the testing set to have popular filenames from the benign set. This resulted in a 3% drop in classifier performance to 90%. This is 1% drop in performance when filename evasion is used compared to not using the filename at all. Thus, to get the improvement of having this feature without the potential evasion downside, we train two classifiers – one with the filename feature and one without. If either classifier identifies a download as social engineering, it is labeled as social engineering by WebSentry. We do not believe having the two classifiers will greatly increase false positives because the 49 downloads in Table 13 are a subset of the 51 downloads in Table 16.

4.6.4 SE New Campaign Detection

In this section, we evaluate WebSentry's ability to detect downloads from new social engineering campaigns versus new downloads from known campaigns. To separate downloads into campaigns, we cluster them using the features and algorithm defined

in Section 4.2.3. Based on the manual analysis we performed on the clusters during our study, we believe they are a good approximation for campaigns. Therefore, we treat each cluster as a unique campaign for our evaluation.

We define a social engineering campaign as new if it is not in our training set. To evaluate WebSentry's performance on downloads from new campaigns, we train n classifiers where n is the total number of campaigns. Each classifier is trained using downloads from all campaigns but the one that represents a new campaign. The test set consists of downloads from the new campaign and the benign set that were observed in the months following those in the training set.

Table 17: New campaigns.

	Known TP	New TP
Ad:Invent+Impersonate+Alarm	97%	95%
Ad:Invent+Impersonate+Comply	92%	91%
Ad:Repackage+Entice	72%	27%
Ad:Repackage+Decoy	87%	87%
Ad:Impersonate+Decoy	100%	100%

Table 17 shows the true positive rate for the classifier trained on all campaigns versus leaving the testing campaign out of the training set. All, but Ad:Repackage+Entice, perform well at finding new campaigns with only a minor or no decrease in performance. The reason for the 45% decrease in performance for Ad:Repackage+Entice is that the majority of downloads are for PUPs that are bundled with legitimate software from well established domains. An important feature for classifying these downloads is predecessor popularity. Removing the campaign from the training set removes the predecessor domains between the first advertisement related HTTP transaction and the download. This causes the lower performance for this class of social engineering attacks. However, since new campaigns are much less common for this class since the downloads are from well established domains, the lower performance is less of an issue.

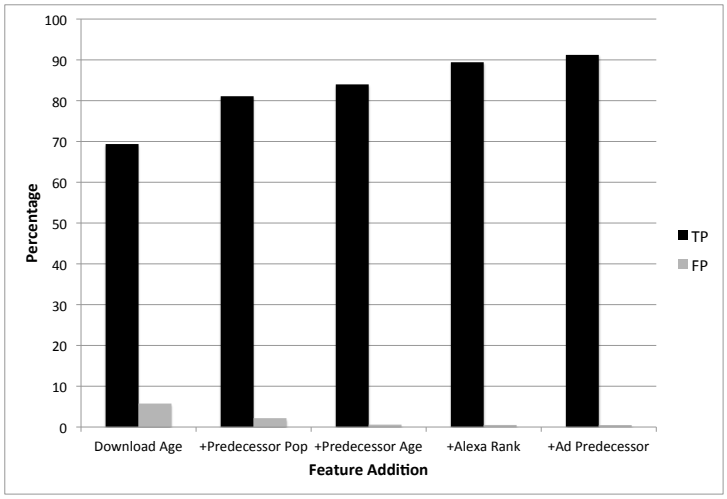

Figure 16: Feature importance using forward feature selection.

4.6.5 Feature Importance

We examine feature importance by performing forward feature selection. We begin by training n classifiers (where n equals the number of features) each with a unique single feature from our feature set. We evaluate each classifier on our test set and keep the feature that performs the best. Then we train $n-1$ classifiers with the best performing feature from the previous step and one of the $n-1$ remaining features. Then the two features from the best performing classifier are retained and used in the next iteration. This process continues until we run out of features or adding a feature no longer improves performance. We measure classifier performance using information gain with the top performing classifier having the highest score.

Figure 16 shows the true and false positive rates measured using forward feature selection. The single feature that provides the largest information gain is download domain age. Using only that feature we have a 69% true positive rate and a 6% false positive rate. Notice that both the true positives and false positives continue to improve as we add additional features. Thus, all the features are important to

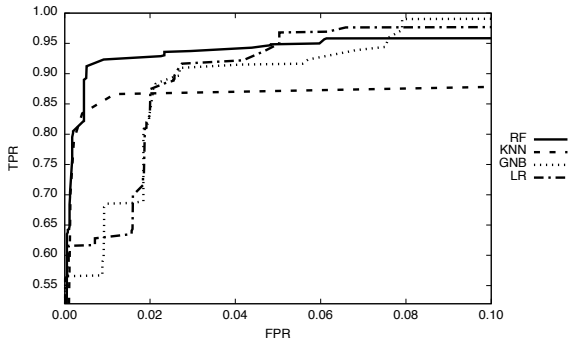

Figure 17: ROC-curve of different models for the Social Engineering Classifier

achieve the performance of our classifier.

4.6.6 Model Selection

For model selection we use our ground truth with the benign set for our negative class (NOT social engineering) and the ad-based set for our positive class (social engineering). We train on data labeled during our study and test on data collected in the months that followed. We evaluate four different models: random forest, k-nearest neighbors, gaussian naive bayes and logistic regression. Figure 17 shows the ROC-curve comparing the performance of the different models.

Random forest outperforms the other models when a false positive rate below 5% is preferred. Both gaussian naive bayes and logistic regression perform better once the false positive rate exceeds 8%. Due to a high base rate of benign downloads (93% are benign), even a 5% false positive rate results in low precision. In fact, at 5% it is only around a 50% chance that the download is social engineering. For these reasons, we chose random forest for our model.

4.7 Discussion and System Limitations

Our study focused exclusively on social engineering (SE) attacks on the web that result in a malicious download. We ignored SE attacks that occur over different mediums (e.g., email) and ones that have other objectives such as phishing. This

limited the types of the SE attacks we studied, but the narrower focus allowed us to examine these attacks in detail. Furthermore, the download and installation of malware offers the attacker many more options for monetization than simply stealing information from the user. As our defenses for drive-by downloads continue to improve, attackers will increase their use of SE attacks for malware distribution. By narrowing the scope of our study we were able to focus our investigation on the tricks and techniques used for SE malware downloads on the web.

Only a small percentage of the SE downloads collected during the study were identified by AV as malware. The majority were adware or PUP. However, AV signatures are far from perfect and we were very conservative with our labeling. For instance, if a single AV identified the executable as adware or PUP we labeled it as such even though others identified it as a trojan. Furthermore, 25% of the downloads are still unlabeled due to lack of AV detection. Therefore, malware downloads are likely under represented in our measurements. As far as the taxonomy and detection, the SE techniques that deliver adware are identical to ones that result in malware; therefore, apply equally well to both SE adware and malware downloads.

The executable downloads for the SE study were collected from a single network. One could argue that this restricted visibility may result in missing classes of SE attacks that are common on other networks that are in different industries or have a different user demographic. Even though it was a single network, it was very large, serving tens of thousands users, and diverse consisting of students, faculty and staff of different ages, cultures and backgrounds. Furthermore, we designed our taxonomy with enough abstraction that the specific tactics and details of an attack can change without altering its classification. As for detection, WebSentry's features are not dependent on the deception and persuasion techniques of an attack, but rely on components such as delivery, which are more difficult for the attacker to modify and are common to many attack classes.

For the SE study we presume an ad click to be caused by the user. Yet, we cannot be certain since our observations are at the network level and not the host. However, it seems unlikely that malicious software would be the cause of the clicks. For one thing, we are not aware of any malicious software that updates itself or installs additional malicious software using ads. Typically, these are hosted at specific URLs designated for that purpose. Furthermore, it would be difficult for the malicious software to get the correct ad shown to the user when visiting a site due to ad syndication. Also, downloading malicious software by clicking on an ad provides no benefits. Software updates are the most common download observed on the network so it is easy for malicious updates to blend into the traffic. For these reasons, we believe that the vast majority of downloads, most likely all of them, are due to a user clicking on the ad not malicious software already on the victim's machine.

Since WebSentry is designed to detect ad based SE downloads, an attacker could evade the system by using search or a web post to get the user's attention instead of an advertisement. Ads, however, are popular with attackers because they can "publish" their SE ad on a site that is already popular with the targeted victims. In addition, ads are only shown to users that trigger their delivery thus reducing exposure that could result in discovery.

Another way an attacker may try to evade the system is to host the executable download on a free file sharing site. This could result in a download with an age > 1 year and a high Alexa score. However, the ad predecessor features "minimum age" and "maximum popularity", which are harder for the attacker to control, would be unaffected. Therefore, it is likely the attack would still be detected. Also, simply knowing that a download is due to an ad implies there is a 50% probability that it is malicious. Furthermore, if hosting malicious downloads on free hosting sites becomes popular then a feature "Free File Hosting" could be added to WebSentry – it is unlikely that many ad based benign downloads are served from free file hosting

sites.

4.8 Related Work

4.8.1 Social Engineeing Taxonomies

Social engineering is an attack on the user not technology. The fundamental concepts that are employed to exploit the user are rooted in modern psychology specifically in the study of persuasion [43] and deception [127]. On the Internet these techniques are applied across mediums such as email, instant messaging, social networks and websites. Several social engineering taxonomies have been proposed by researchers [26, 66, 76, 77, 90, 91]. These taxonomies take a broad view of the entire social engineering domain. Because of the large variance of techniques (e.g., phishing, dumpster diving), tactics (e.g., fear, greed) and mediums (e.g, email, telephone) used in social engineering attacks, these taxonomies are high level and place attacks in large buckets. We created a categorization system for classifying social engineering attacks on the web that result in a malware download. This is a much more restricted domain and is at the level where most of the existing taxonomies end. By focusing our categorization system on this subset, we are able to separate attacks into classes specific enough to allow for the design of defenses that apply to all attacks within a class, but are general enough so that different types of attacks (e.g., fake av, fake java update, fake flash) that use similar methods are not separated.

4.8.2 Social Engineering Detection on the Web

Researchers have also examined specific types of social engineering attacks. Fake AV has been a popular topic. The infrastructure and operations of fake AV were studied in [123]. Mavrommatis et al [85] perform an empirical analysis of fake AV to understand its prevalence, domain name characteristics and malware distribution. In addition, researchers have developed systems to detect fake AV websites [49, 73]. In [73], they develop a web crawler that uses a classifier to detect fake AV websites.

The features used by the system are primarily content based (e.g., image similarity, content keyword) and take time to collect resulting in an average classification time of 36 seconds. Another system [52], trains a classifier to detect malicious trick banners on a webpage using visual properties such as image size, color and placement. Our work is different because it is a general approach to studying and developing defenses for social engineering attacks on the web that result in a malicious download. It is not limited to specific attack types such as fake AV and trick banners, but has broader application and provides a higher level of attack abstraction.

MadTracer [81] uses features extracted from short segments of advertisement paths collected through active crawling to detect malvertising including drive-by download, scam and click-fraud. For its feature set it uses frequency of nodes, node roles, domain registration/expiration dates and URL similarity. Our work is different because we study social engineering attacks on the web that result in a malicious download not malvertising. Even though malvertising is commonly used in social engineering attacks, that was not the focus of [81]. In fact, the only class of social engineering download attack they detect and discuss is fake AV, which comprises less than 1% of the attacks we discover.

4.9 Conclusion

We study SE attacks by collecting and labeling SE downloads on a live network. From these labeled downloads, we created a categorization system that expresses how attackers gain the user's attention and trick them into downloading a malicious executable. Also, by reconstructing the download path followed by SE victims, we observe that a large fraction of SE attacks are delivered using malicious online advertisement served by low tier ad networks. Lastly, we construct WebSentry using features inspired by our SE download path measurements. We show that WebSentry

detects the vast majority of SE downloads and has a low false positive rate. Furthermore, the features are generic enough that it can detect new SE campaigns that are not in its training set.

However, detecting and preventing malware downloads at the network level is not a complete solution. Many hosts will already be infected at the time of the defense deployment. Also, new infections should be expected because malware can be delivered through other mediums, e.g., USB, and hosts will visit the network that are not under the control of the organization, e.g., bring your own device. In the next chapter, we develop a technique for detecting infected hosts on the network. Thus, providing a more comprehensive solution for malware defense.

CHAPTER V

EXECSCENT: MINING FOR NEW C&C DOMAINS IN LIVE NETWORKS WITH ADAPTIVE CONTROL PROTOCOL TEMPLATES

5.1 Introduction

Defense against the initial download is only a partial solution to the malware problem. Hosts will continue to become infected either through other mediums or while visiting insecure networks. Thus, a network malware defense system needs to both prevent initial infections and detect hosts that are already infected. The previous chapters focused on preventing infections that result from a malware download. This chapter examines how to detect hosts already infected by learning the structure their C&C protocol.

Code reuse is common practice in malware [69,75]. Often, new (polymorphic) malware releases are created by simply re-packaging previous samples, or by augmenting previous versions with a few new functionalities. Moreover, it is not uncommon for the source code of successful malware to be sold or leaked on underground forums, and to be reused by other malware operators [55].

Most modern malware, especially botnets, consist of (at least) two fundamental components: a *client agent*, which runs on victim machines, and a *control server* application, which is administered by the malware owner. Because code reuse applies to both components[1], this naturally results in *many different malware samples sharing*

[1]For example, web-based malware control panels can be acquired in the Internet underground markets and re-deployed essentially *as is*, while the client agents can be obtained using *do-it-yourself* malware creation kits [47].

a common command-and-control (C&C) protocol, even when control server instances owned by different malware operators use different C&C domains and IPs.

In this chapter, we present ExecScent, a novel system that aims to mine new, previously unknown C&C domain names from *live* enterprise network traffic (see Figure 18). Starting from a *seed* list of known C&C communications and related domain names found in malware-generated network traces, ExecScent aims to discover new C&C domains by taking advantage of the commonalities in the C&C protocol shared by different malware samples. More precisely, we refer to the C&C protocol as the set of specifications implemented to enable the malware control application logic, which is defined at a higher level of abstraction compared to the underlying transport (e.g., TCP or UDP) or application (e.g., HTTP) protocols that facilitate the C&C communications. ExecScent aims to automatically learn the unique traits of a given C&C protocol from the seed of known C&C communications to derive a *control protocol template* (CPT), which can in turn be deployed at the edge of a network to detect traffic destined to new C&C domains.

ExecScent builds *adaptive* templates that also learn from the *traffic profile* of the network where the templates are to be deployed. The goal is to generate *hybrid* templates that can *self-tune* to each specific deployment scenario, thus yielding a better trade-off between true and false positives for a given network environment. The intuition is that different networks have different traffic profiles (e.g., the network of a financial institution may generate very different traffic compared to a technology company). It may therefore happen that a CPT could (by chance) raise a non-negligible number of false positives in a given network, say Net_A, while generating true C&C domain detections and no false positives in other networks. We take a pragmatic approach, aiming to automatically identify these cases and lowering the "confidence" on that CPT *only* when it is deployed to Net_A. This allows us to lower the overall risk of false positives, while maintaining a high probability of detection in

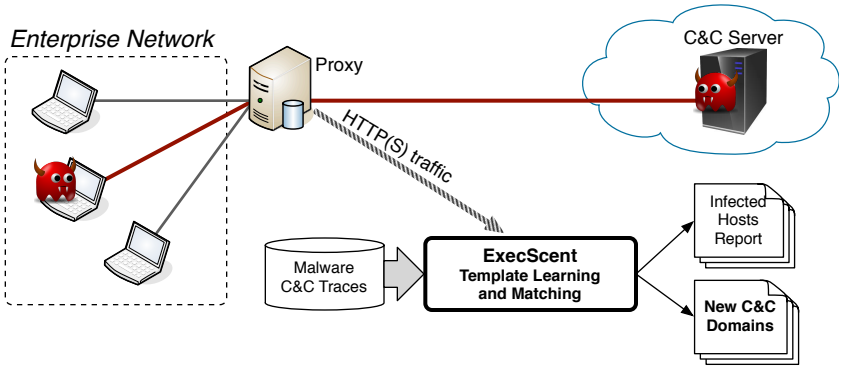

Figure 18: ExecScent deployment overview. Adaptive control protocol templates are learned from both malware-generated network traces and the live network traffic observation. The obtained adaptive templates are matched against new network traffic to discover new C&C domains.

other networks. We further motivate the use of adaptive templates in Section 5.3.

ExecScent focuses on HTTP-based C&C protocols, because studies have shown that HTTP-based C&C communications are used by a large majority of malware families [115] and almost all known mobile bots [133]. Moreover, many enterprise networks employ strict egress filtering firewall rules that block all non-web traffic. This forces malware that target enterprise networks to use HTTP (or HTTPS) as the communication protocol of choice. It is also important to notice that many modern enterprise networks deploy web proxies that enforce SSL man-in-the-middle[2] (SSL-MITM). Therefore, enterprise networks can apply ExecScent's templates at the web proxy level to discover new C&C domains even in cases of HTTPS-based C&C traffic.

In summary, we discuss the following in this chapter:

- We present ExecScent, a novel system for mining new malware C&C domains from live networks. ExecScent automatically learns C&C traffic models that can adapt to the deployment network's traffic. This *adaptive* approach allows us to greatly reduce the false positives while maintaining a high number of true

[2]See `http://crypto.stanford.edu/ssl-mitm/`, for example.

positives. To the best of our knowledge, ExecScent is the first system to use this type of adaptive C&C traffic models.

- We implemented a prototype version of ExecScent, and deployed it in three different large networks for a period of two weeks. During the deployment, we discovered many new, previously unknown C&C domains and hundreds of new infected machines, compared to using a large up-to-date commercial C&C domain blacklist.

- We deployed the new C&C domains mined by ExecScent to six large ISP networks, discovering more than 25,000 new infected machines.

5.2 System Overview

The primary goal of ExecScent is to generate control protocol templates (CPTs) from a seed of known malware-generated HTTP-based C&C communications. We then use these CPTs to identify new, previously unknown C&C domains.

ExecScent automatically finds common traits among the C&C protocol used by different malware samples, and encodes these common traits into a set of CPTs. Each template is labeled with the name of the malware family or (if known) criminal operator associated with the C&C traffic from which the CPT is derived. Once a CPT is deployed at the edge of a network (see Figure 18), any new HTTP(S) traffic that matches the template is classified as C&C traffic. The domain names associated with the matched traffic are then flagged as C&C domains, and attributed to the malware family or operator with which the CPT was labeled.

Figure 19 presents an overview of the process used by ExecScent to generate and label the CPTs. We briefly describe the role of the different system components in this section, deferring the details to Section 5.4.

Given a large repository of malware-generated network traces, we first reconstruct all HTTP requests performed by each malware sample. Then, we apply a *request*

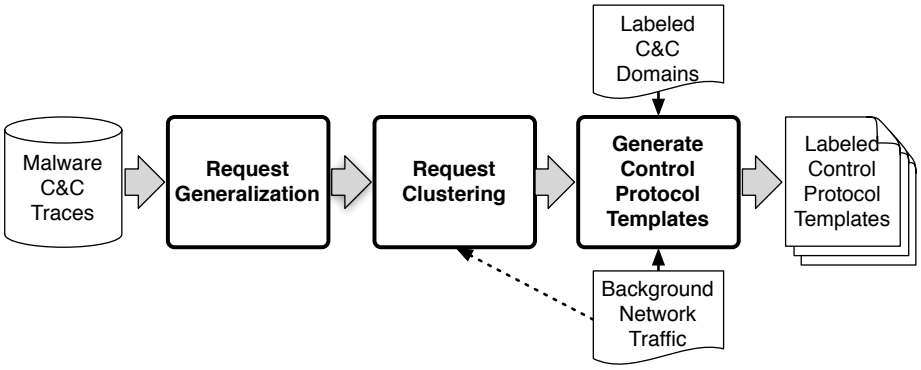

Figure 19: ExecScent system overview.

generalization process, in which (wherever possible) we replace some of the request parameters (e.g., URL parameter values) with their data type and length, as shown in the example in Figure 20. Notice that ExecScent considers the entire content of the HTTP requests, not only the URLs (see Section 5.3.2), and the generalization process is applied to different parts of the request header. The main motivation for applying the generalization step is to improve the accuracy of the *request clustering* process, in which we aim to group together malware-generated requests that follow a similar C&C protocol.

Once the malware requests have been clustered, we apply a *template learning* process in which we derive the CPTs. Essentially, a CPT summarizes the (generalized) HTTP requests grouped in a cluster, and records a number of key properties such as the structure of the URLs, the set of request headers, the IP addresses contacted by the malware, etc. Furthermore, the templates associate a malware-family label to each template (see Section 5.4.4 for details).

Before the templates are deployed in a network, we *adapt* the CPTs to the "background traffic" observed in that network. In particular, for each template component (e.g., the generalized URL path, the user-agent string, the request header set, etc.), we compute how frequently the component appeared in the deployment network. CPT components that are "popular" in the background traffic will be assigned a

lower "match confidence" for that network. On the other hand, components that appear very infrequently (or not at all) in the traffic are assigned a higher confidence. We refer to these "rare" components as having high *specificity*, with respect to the deployment network's traffic. The intuitions and motivations for this approach are discussed in more detail in the next section.

After deployment, an HTTP request is labeled as C&C if it matches a CPT with *high similarity and specificity*. That is, if the request closely matches a CPT and the matching CPT components have high specificity (i.e., rarely appeared) in that particular deployment network.

5.3 Approach Motivations and Intuitions

In this section, we discuss the intuitions that motivated us to build adaptive control protocol templates. Furthermore, we discuss the advantages of considering the entire content of C&C HTTP requests, rather than limiting ourselves to the URL strings, as done in previous work [104, 130].

5.3.1 Why Adaptive Templates?

As most other traffic models, ExecScent's CPTs, which are derived from and therefore can match C&C traffic, may be imperfect and could generate some false positives. To minimize this risk, ExecScent builds *adaptive* control protocol templates that, besides learning from known malware-generated C&C traffic, also learn from the traffic observed in the network where the templates are being deployed. Our key observation is that different enterprise networks have different traffic profiles. The traffic generated by the computer network of a financial institute (e.g., a large bank) may look quite different from traffic at a manufacturing company (e.g., a car producer) or a technology company (e.g., a software-development company). It may therefore happen that a CPT could (by chance) raise a non-negligible number of false positives in a given network, say Net_X, and several true detections with no or very few false

positives in other networks. Intuitively, our objective is to automatically identify these cases, and lower the "confidence" on that template when it matches traffic from Net_X, while keeping its confidence high when it is deployed elsewhere.

For example, assume Net_B is a US bank whose hosts have rarely or never contacted IPs located, say, in China. If an HTTP request towards an IP address in China is found, this is by itself an anomalous event. Intuitively, if the request also matches a CPT, our confidence on a correct match (true C&C communication) can be fairly high. On the other hand, assume Net_A is a car manufacturer with partners in China, with which Net_A's hosts communicate frequently. If an HTTP request in Net_A matches a CPT but is directed towards an address within one of the IP ranges of the manufacturer's partners, our confidence on a correct match should be lowered.

More specifically, consider the following hypothetical scenario. Assume we have a template τ that matches an HTTP request in both Net_A and Net_B with a *similarity* score s. For simplicity, let us assume the score s is the same for both Net_A's traffic and Net_B's traffic. Suppose also that the server's IP (or it's `/24` prefix) associated with the matching traffic is ip_a for Net_A and ip_b for Net_B. Also, suppose that ip_a is "popular" in network Net_A, whereas ip_b has very low popularity in Net_B because it has never been contacted by hosts in that network. Because ip_a is very popular in Net_A, meaning that a large fraction (e.g., more than 50%) of the hosts in Net_A has contacted the domain in the past, it is likely that the template τ is fortuitously matching benign traffic, thus potentially causing a large number of false positives in Net_A. On the other hand, because ip_b has very low popularity in Net_B, it is more likely that the match is a true detection, or that in any case τ will generate very few (potentially only one) false positives in Net_B. Consequently, based on a model of recent traffic observed in Net_A and Net_B, we should lower our confidence in τ for the matches observed in Net_A, but not for Net_B. In other words, τ should automatically *adapt* to Net_A to "tune down" the false positives. At the same time,

keeping the confidence in τ high for Net_B means that we will still be able to detect C&C communications that match τ, while keeping the risk of false positives low. We generalize this approach to *all* other components of ExecScent's templates (e.g, the structure of the URLs, the user-agent strings, the other request headers, etc.), in addition to the destination IPs.

Overall, our confidence on a match of template τ in a given network Net_X will depend on two factors:

- *Similarity*: a measure of how closely an HTTP request matches τ.
- *Specificity*: a measure of how specific (or *rare*) are the components of τ with respect to Net_X's traffic.

An HTTP request is labeled as C&C if it matches a CPT with both high similarity and high specificity. We show in Section 5.5 that this approach outperforms C&C models that do not take such specificity into account.

5.3.2 Why Consider All Request Content?

Malware C&C requests typically need to carry enough information for a malware agent running on a victim to (loosely) authenticate itself with the C&C server. Intuitively, the C&C server wants to make sure that it is talking to one of its *bots*, thus avoiding exposure of its true nature or functionalities to crawlers or security researchers who may be probing the server as part of an investigation. This is often achieved by using a specific set of parameter names and values that must be embedded in the URL for the C&C requests to be successful. Previous work on automatic URL signature generation has shown promising results in such cases [104, 130]. However, some malware (e.g., TDL4 [27]) exchanges information with the C&C by first encrypting it, encoding it (e.g., using base-64 encoding), and embedding it in the URL path. Alternatively, identifier strings can also be embedded in fields such as `user-agent` (e.g., some malware samples use their MD5 hash as `user-agent` name),

encoded in other request headers (e.g., in the `referrer`), or in the body of POST requests. Therefore, only considering URLs may not be enough to accurately model C&C requests and detect new C&C domains, as supported by our experimental results (Section 5.5).

5.4 System Details

We now detail the internals of ExecScent. Please refer to Section 5.2 for a higher-level overview of the entire system.

5.4.1 Input Network Traffic

As we mentioned in Section 5.1, ExecScent focuses on HTTP-based malware, namely malware that leverage HTTP (or HTTPS) as a base network protocol on top of which the malware control protocol is "transported". To this end, ExecScent takes in as input a feed of malware-generated HTTP traffic traces (in our evaluation, we use a large set of malware traces provided to us by a well-known company that specializes in malware defense).

It is worth remembering that while some malware may use HTTPS traffic as a way to evade detection, this does not represent an insurmountable obstacle in our deployment scenarios (see Figure 18). In fact, many enterprise networks, which represent our target deployment environment, already deploy web proxy servers that can perform SSL-MITM and can therefore forward the clear-text HTTP requests to ExecScent's template matching module, e.g., using the ICAP protocol (RFC 3507). Also, malware samples that appear to be using HTTPS traffic may be re-analyzed in a controlled environment that includes an SSL-MITM proxy interposed between the (virtual) machine running the sample and the egress router. After all, HTTPS-based malware that do not support or choose not to run when an SSL-MITM proxy is present will also fail to run in enterprise networks that have a similar setting, and are therefore of less interest.

5.4.2 Request Generalization

As we discuss in the following sections, to obtain quality control protocol templates we first need to group similar C&C requests. To this end, an appropriate similarity metric needs to be defined before clustering algorithms can be applied. Previous works that propose URL-centric clustering systems [104, 130] are mainly based on string similarity measures. Essentially, two URLs are considered similar if they have a small edit distance, or share a number of substrings (or tokens). However, these systems do not take into account the fact that URLs often contain variables whose similarity is better measured according to their data *type* rather than considering specific sequences of characters. Consider the two hypothetical C&C requests in Figure 20. Taken as they are (Figure 20a), their distance is relatively large, due to the presence of several different characters in the strings. To avoid this, ExecScent uses a set of heuristics to detect strings that represent data of a certain type, and replaces them accordingly using a placeholder tag containing the data type and string length (Figure 20b).

For example, we would identify "fa45e" as lowercase hexadecimal because it contains numeric characters and the alphabetic characters are all valid lowercase hexadecimal digits. The data types we currently identify are *integer*, *hexadecimal* (upper, lower and mixed case), *base64* (standard and "URL safe") and *string* (upper, lower and mixed case). In addition, for integer, hexadecimal and string we can identify the data type plus additional punctuation such as ":" or "." (e.g., 192.168.1.1 would be identified as a data type of integer+period of length 11). Furthermore, our heuristics can easily be extended to support data types such as IP address, MAC address, MD5 hash and version number.

This generalization process allows us to define a better similarity metric (Section 5.4.7), which is instrumental to obtaining higher quality C&C request clusters. Notice also that while previous works such as [104, 130] focus only on URL

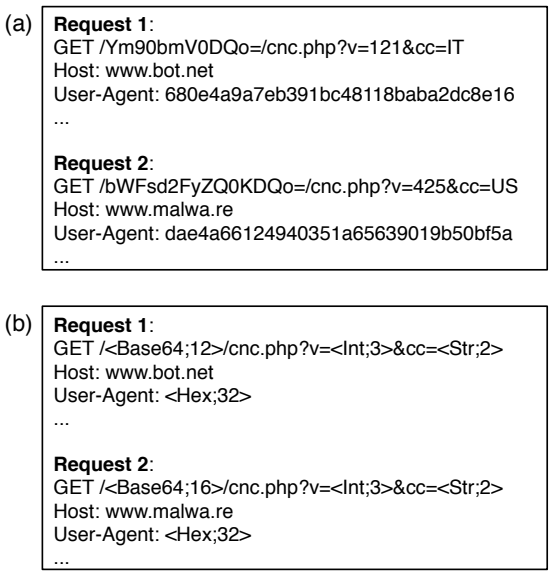

Figure 20: Example C&C requests: (a) original; (b) generalized.

strings, ExecScent takes the entire request into account. For example, in Figure 20 the `user-agent` strings are MD5s, and can be generalized by replacing the specific MD5 strings with the appropriate data type and length information.

5.4.3 Request Clustering

Before extracting the templates, we group together similar C&C requests. This clustering step simply aims to assist the automatic CPT generation algorithm, improving efficiency and yielding templates that are at the same time *generic* enough to match similar (but not identical) C&C communications in new traffic, and *precise* enough to generate very few or no false positives.

We perform C&C request clustering in two phases. During the first phase, we coarsely group C&C requests based on their destination IPs. Specifically, given two C&C requests, we group them together if their destination IPs reside in `/24` (or class C) networks that *share a DNS-based relationship*. Namely, we consider two `/24` networks as related if there exists at least one domain name that within the last 30 days resolved to different IP addresses residing in the two different networks. To find

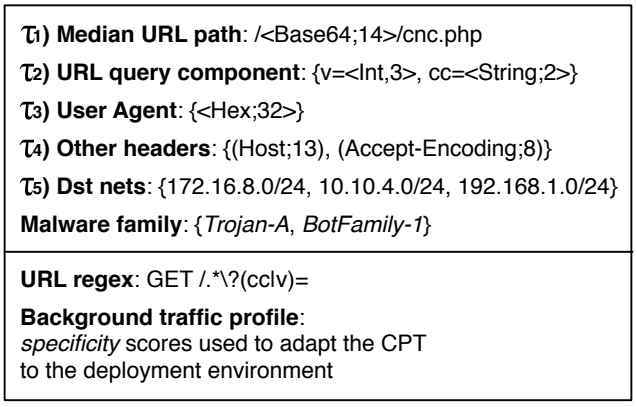

Figure 21: Example C&C requests: (a) original; (b) generalized.

such relationships, we rely on a large passive DNS database [53].

In the second phase, we consider one coarse-grained cluster at a time, and we further group a cluster's C&C requests according to a content similarity function. We use an agglomerative hierarchical clustering algorithm to group together C&C requests within a coarse-grained cluster that carry similar generalized URLs, similar `user-agent` strings, similar numbers of HTTP header fields and respective values, etc. When measuring the similarity between two requests, we take into account both the *similarity* and *specificity* of the requests' content, where the specificity (or low "popularity") can be measured with respect to a dataset of traffic recently collected from different networks (dashed arrow in Figure 19). For a more detailed definition of the similarity function used in the clustering step, we refer the reader to Section 5.4.7.

5.4.4 Generating CPTs

Once C&C requests have been clustered, a control protocol template (CPT) is generated from each cluster. At this stage, we consider only clusters that contain at least one HTTP request to a known C&C domain. Each template represents a summary of all C&C requests in a cluster, and contains the following components, as also shown in Figure 21:

τ_1) *Median URL path*: median path string that minimizes the sum of edit distances from all URL paths in the requests (see [48] for a definition of median string). *Intuition*: although the URL path may vary significantly from one malware installation to another, we observed many cases in which there exist "stable" path components that are unique to a specific malware family or operation.

τ_2) *URL query component*: stores the set of parameter names, value types and lengths observed in the query component [33] of each of the URLs. *Intuition*: URL parameters are often used by malware to convey information about the infected host, such as its OS version, a unique identifier for the infected machine, etc.

τ_3) *User-agent*: the set of all different (generalized) user-agent strings found in the requests. *Intuition*: the user-agent is one of the most abused HTTP headers by malware, and is sometimes used as a loose form of authentication.

τ_4) *Other headers*: the set of other HTTP headers observed in the requests. For each header, we also store the length of its value string. *Intuition*: the set of header names, their order and values are sometimes unique to a malware family.

τ_5) *Dst. networks*: the set of all destination `/24` networks associated with the C&C requests in the cluster. *Intuition*: in some cases, the C&C server may be relocated to a new IP address within the same (possibly "bullet-proof") network.

- *Malware family*: the (set of) malware family name(s) associated to the known C&C requests in the cluster.

In addition, each CPT includes the following deployment-related information:

- *URL regex*: to increase the efficiency of the template matching phase (Section 5.4.6), each template includes a regular expression automatically generated

from the set of URL strings in the requests. The URL regex is intentionally built to be very generic and is used during deployment for the sole purpose of filtering out traffic that is extremely unlikely to closely match the entire template, thus reducing the cost of computing the similarity between HTTP requests in live traffic and the template.

- *Background traffic profile*: information derived from the traffic observed in the deployment environment within the past W days (where W is a system parameter). This is used for computing the specificity of the CPT components, thus allowing us to adapt the CPT to the the deployment network, as explained in detail in Section 5.4.5.

Notice that a CPT acts as the *centroid* for the cluster from which it was derived. To determine if a new request is similar enough to a given cluster, we only need to compare it with the CPT, rather than all of the clustered C&C requests. Therefore, CPTs provide an efficient means of measuring the similarity of a new request to the C&C protocol used by the clustered malware samples.

5.4.5 Adapting to a Deployment Network

As explained in Section 5.3.1, once the CPTs are deployed, an HTTP request is labeled as C&C if it matches a CPT τ with both high *similarity* and *specificity*. To this end, we first need to compute a specificity score for each element of the k-th component τ_k of τ, which indicates how "unpopular" that element is with respect to the traffic profile in the deployment network (notice that $k = 1, \ldots, 5$, as shown in Figure 21 and Section 5.4.4).

For example, to compute the specificity scores for τ_3, we first compute a *host-based popularity* score hp_{ua_i} for each `user-agent` string ua_i in the set τ_3. We consider the number of hosts hn_{ua_i} in the deployment network that generated an HTTP request

containing ua_i during the last W days, where W is a configurable time-window parameter. We define $hp_{ua_i} = \frac{hn_{ua_i}}{\max_j\{hn_{ua_j}\}}$, where the max is taken over all `user-agent` strings ua_j observed in the deployment network's traffic. Similarly, we compute a *domain-based popularity* score dp_{ua_i}, based on the number of distinct destination domain names dn_{ua_i} with one or more HTTP requests that contain ua_i. We define $dp_{ua_i} = \frac{dn_{ua_i}}{\max_j\{dn_{ua_j}\}}$. The intuition is that a `user-agent` string can only be considered truly popular if it spans many hosts and domains. On the other hand, we do not want to consider a ua_i as very popular if it has high host-based popularity (e.g., "Windows-Update-Agent") but low domain-based popularity (e.g., because the only domain on which it is used is `microsoft.com`). Finally, we define the specificity score for ua_i as $\sigma_{3,ua_i} = 1 - min(hp_{ua_i}, dp_{ua_i})$. In a similar way, we compute a specificity score σ_{4,hd_l} for each header element hd_l in τ_4.

To compute the specificity scores for τ_5, we simply compute the host-based popularity hp_{net_i} for each `/24` network prefix $net_i \in \tau_5$, and we define a separate score $\sigma_{5,net_i} = (1 - hp_{net_i})$ for each prefix.

5.4.5.1 URL Specificity

Computing the specificity of the components of a URL is more complex, due to the large variety of unique URLs observed every day on a given network. To address this problem, we rely on a supervised classification approach. First, given a dataset of traffic collected from a large network, we extract all URLs, and learn a *map* of URL *word frequencies*, where the "words" are extracted by tokenizing the URLs (e.g., extracting elements of the URL path, filename, query string, etc.). Then, given a new URL, we translate it into a feature vector in which the statistical features measure things such as the average frequency of single "words" in the tokenized URL, the average frequency of word bigrams in the query parameters, the frequency of the file name, etc. (to extract the frequency values for each word found in the URL we lookup

the previously learned map of word frequencies).

After we translate a large set of "background traffic URLs" into feature vectors, we train an SVM classifier [41] that can label new URLs as either *popular* or *unpopular*. To prepare the training dataset we proceed as follows. We first rank the "background URLs" according to their domain-based popularity (i.e., URLs that appear on requests to multiple sites on different domain names are considered as more popular). Then, we take a sample of URLs from the top and from the bottom of this ranking, which we label as *popular* and *unpopular*, respectively. We use this labeled dataset to train the SVM classifier, and we rely on the max-margin approach used by the SVM [45] to produce a model that can generalize to URLs not seen during training.

During the operational phase (once the SVM classifier is trained and deployed), given a URL u_i, we can first translate u_i into its corresponding feature vector v_i, as described above, and feed v_i to the SVM classifier. The classifier can then label u_i as either *popular* or *unpopular*. In practice, though, rather than considering these class labels, we only take into account the classification score (or confidence) associated with the *popular* class[3]. Therefore, the SVM's output can be interpreted as follows: the higher the score, the more u_i "looks like" a popular URL, when compared to the large set of URLs observed in the background traffic. Finally, the specificity score for the URL is computed as $\sigma_{u_i} = 1 - p_{u_i}$, where p_{u_i} is the SVM output for URL u_i.

Now, let us go back to consider the template τ and its URL-related components τ_1 and τ_2 (see Figure 21). We first build a "median URL" u_m by concatenating the median URL path (τ_1) to the (sorted) set of generalized parameter names and values (τ_2). We then set the similarity scores $\sigma_1 = \sigma_2 = \sigma_{u_m}$, where σ_{u_m} is the specificity of u_m.

[3]We calibrate the classification scores output by the SVM classifier using the method proposed by Platt [105].

5.4.6 Template Matching

Template matching happens in two phases. As mentioned above, each template contains an URL regular expression automatically derived from the C&C requests in a cluster. Given a new HTTP request r, to test whether this request matches a template τ, we first match r's URL to τ's URL regex. It is worth noting that, as mentioned in Section 5.4.4, the URL regex is intentionally built to be very generic, and is merely used to efficiently filter out traffic that is extremely unlikely to match the entire template. Furthermore, we check if the destination IP of r resides within any of the `/24` prefixes in τ (specifically in component τ_5). If neither the URL regex nor the destination IP have a match, we assume r does not match τ. Otherwise, we proceed by considering the entire content of request r, transforming r according to the request generalization process (see Section 5.4.2), and measuring the overall *matching score* $S(r, \tau)$ between the (generalized) request r and the template τ.

In summary, the score S is obtained by measuring the similarity between all the components of the request r and the respective components of the template τ. These similarity measures are then weighted according to their specificity, and the matching score $S(r, \tau)$ is computed as the average of all weighted component similarities. A detailed definition of the similarity functions and how specificity plays an explicit role in computing $S(r, \tau)$ is given in Section 5.4.7.

If $S(r, \tau)$ exceeds a tunable detection threshold θ, then the request r will be deemed a C&C request and the domain name associated with r (assuming r is not using a hardcoded IP address) is classified as C&C domain and labeled with the malware family associated to τ. Furthermore, the host from which the request r originated is labeled as compromised with τ's malware family.

5.4.7 Similarity Functions

5.4.7.1 CPT matching score

To determine if a new HTTP request r matches a CPT τ, we compute a *matching score* $S(r, \tau)$ as follows:

$$S(r, \tau) = \frac{\sum_k w_k(s_k, \sigma_k) \cdot s_k(r_k, \tau_k)}{\sum_k w_k(s_k, \sigma_k)} \cdot \sigma_d \quad (1)$$

where s_k is a similarity function that compares each element τ_k of τ (Section 5.4.4) with its respective counterpart r_k of r, and where w_k is a *dynamic weight* (whose definition is given below) that is a function of both the similarity s_k and the specificity σ_k of the k-th component of τ. The denominator scales $S(r, \tau)$ between zero and one.

The factor σ_d is the *specificity of the destination domain* d of request r, which is computed as $\sigma_d = 1 - \frac{m_d}{\max_i\{m_{d_i}\}}$, where m_d is the number of hosts in the deployment network's traffic that queried domain d, and $\max_i\{m_{d_i}\}$ is the number of hosts that queried the most "popular" domain in the traffic. Accordingly, we use σ_d to decrease the matching score $S(r, \tau)$ for low-specificity domains (i.e., domains queried by a large number of hosts). The intuition is that infections of a specific malware family often affect a relatively limited fraction of all hosts in an enterprise network, as most modern malware propagate relatively "slowly" via drive-by downloads or social engineering attacks. In turn, it is unlikely that a new C&C domain will be queried by a very large fraction (e.g., $> 50\%$) of all hosts in the monitored network, within a limited amount of time (e.g., one day).

In the following, we describe the details of the similarity functions $s_k(\cdot)$ used in Equation 1. In addition, we further detail how the specificity value of each component is selected, once the value of $s_k(\cdot)$ has been computed (for the definition of specificity, we refer the reader to Section 5.4.5).

s_1 - Given the path of the URL associated with r, we measure the normalized edit distance between the path and the CPT's median URL path τ_1. The URL path

specificity σ_1 is computed as outlined in Section 5.4.5.

s_{2a} - We measure the Jaccard similarity [4] between the set of parameter names in the URL query-string of r and the set of names in τ_2. The specificity of the parameter names σ_{2a} is equal to σ_2 (see Section 5.4.5).

s_{2b} - We compare the data types and lengths of the values in the generalized URL query-string parameters (see Section 5.4.2). For each element of the query string, we assign a score of one if its data type in r matches the data type recorded in τ_2. Furthermore, we compute the ratio between the value length in r and in τ_2. Finally, s_{2b} is computed by averaging all these scores, whereby the more data types and lengths that match, the higher the similarity score. As in s_{2a}, we set $\sigma_{2b} = \sigma_2$.

s_3 - We compute the normalized edit distance between the (generalized) `user-agent` string in r, and each of the strings in the set τ_3. Let d_m be the smallest of such distances, where m is the closest of the template's `user-agent` strings. We define $s_3 = 1 - d_m$, and set the specificity $\sigma_3 = \sigma_{3,m}$.

s_4 - Given the remaining request header fields in r, we measure the similarity from different perspectives. First, we compute the Jaccard similarity j between the set of headers in r and the set τ_4. Furthermore, we consider the order of the headers as they appear in r and in the requests from which τ was derived. If the order matches, we set a variable $o = 1$, otherwise we set $o = 0$. Finally, for each header, we compare the ratio between the length of its value as it appears in r and in τ_5, respectively. The similarity s_4 is defined as the average of all these partial similarity scores (i.e., of j, o, and the length ratios). We set the specificity score $\sigma_5 = \min_l\{\sigma_{5,hd_l}\}$, where the hd_l are the request headers.

[4] $J = \frac{|A \cap B|}{|A \cup B|}$

s_5 - Let ρ be the destination IP of request r. If ρ resides within any of the `/24` network prefixes in τ_5, we set $s_5 = 1$, otherwise we assign $s_5 = 0$. Assume ρ is within prefix $n \in \tau_5$ (in which case $s_5 = 1$). In this case, we set the specificity $\sigma_5 = \sigma_{5,n}$.

The dynamic weights $w_k(\cdot)$ are computed as follows:

$$w_k(s_k, \sigma_k) = \hat{w}_k \cdot \left(1 + \frac{1}{(2 - s_k \cdot \sigma_k)^n}\right) \tag{2}$$

where $\hat{w}_k$ is a *static* weight (i.e., it takes a fixed value), and n is a configuration parameter. Notice that $w_k \in [\hat{w}_k(1 + \frac{1}{2^n}), 2\hat{w}_k]$, and that these weights are effectively normalized by the denominator of Equation 1, thus resulting in $S(r, \tau) \in [0, 1]$ (since $s_k \in [0, 1], \forall k$, and $\sigma_d \in [0, 1]$, by definition).

The intuition for the dynamic weights $w_k(\cdot)$ is that we want to give higher weight to components of a request r that match their respective counterpart in a CPT τ with both high similarity and high specificity. In fact, the weight will be maximum when both the similarity and specificity are equal to one, and will tend to the minimum when either the similarity or specificity (or both) tend to zero.

In summary, *similarity measures the likeness of two values*, whereas *specificity measures their uniqueness* in the underlying network traffic. The dynamic weights allow us to highlight the *rare structural elements* that are common between a CPT and a request, so that we can leverage them as the dominant features for detection. Because rare structural elements differ in their importance across malware families, by emphasizing these "unique features" we are able to detect and distinguish between different malware families.

5.4.7.2 Similarity function for clustering phase

In Section 5.4.3, we have described the C&C request clustering process. In this section we define the function used to compute the similarity between pairs of HTTP requests, which is needed to perform the clustering.

Given two HTTP requests r_1 and r_2, we compute their similarity using Equation 1. At this point, the reader may notice that Equation 1 is defined to compare an HTTP request to a CPT, rather than two requests. The reason why we can use Equation 1, is that we can think of a request as a CPT derived from only one HTTP request. Furthermore, if we want to include the specificity scores, which are used to make the weights w_k dynamic, we can use a dataset of traffic previously collected from one or more networks (see dashed arrow in Figure 19).

5.5 Evaluation

In this section, we describe the data used to evaluate ExecScent (Section 5.5.1), how the system was setup to conduct the experiments (Section 5.5.2), and present the experimental results in different live networks (Section 5.5.3). Furthermore, we quantify the advantage of modeling entire HTTP requests, rather than only considering URLs, and of using adaptive templates over "static" C&C models (Section 5.5.4). In addition, we show the benefits obtained by deploying new C&C domains discovered by ExecScent into large ISP networks (Section 5.5.5).

5.5.1 Evaluation Data

5.5.1.1 Malware Network Traces

We obtained access to a commercial feed of malware intelligence data (provided to us by a well known security company), which we used to generate the control protocol templates (CPTs). Through this feed, we collected about 8,000 malware-generated network traces per day that contained HTTP traffic. Each network trace was marked with a hash of the malware executable that generated the network activity, and (if known) by the related malware family name.

5.5.1.2 Live Network Traffic

To evaluate ExecScent, we had access to the *live* traffic of three large production networks, which we refer to as UNetA, UNetB, and FNet. Networks UNetA and UNetB are two different academic networks based in the US, while FNet is the computer network of a large North-American financial institution. Table 18 reports statistics with respect to the network traffic observed in these three networks. For example, in UNetA we observed an average of 7,893 distinct active source IP addresses per day. In average, these network hosts generated more than 34.8M HTTP requests per day, destined to 149,481 different domain names (in average, per day).

Table 18: Live network traffic statistics (avg. per day)

	UNetA	UNetB	FNet
Distinct Src IPs	7,893	27,340	7,091
HTTP Requests	34,871,003	66,298,395	58,019,718
Distinct Domains	149,481	238,014	113,778

5.5.1.3 Ground Truth

To estimate true and false positives, we rely on the following data:

- CCBL: we obtained a large black-list containing hundreds of thousands of C&C domains provided by a well known security company, which we refer to as CCBL. It is worth noting that CCBL is different from most publicly available domain black-lists for two reasons: 1) the C&C domains are carefully vetted by professional threat analysts; 2) the domains are labeled with their respective malware families and, when available, a malware operator name (i.e., an identifier for the cyber-criminal group that operates the C&C).

- ATWL: we derived a large white-list of *benign* domain names from Alexa's top 1 million global domains list (`alexa.com`). From these 1M domains, we filtered out domains that can be considered as *effective* top level domains[5] (TLDs),

[5] `http://publicsuffix.org`

such as domains related to dynamic DNS services (e.g., dyndns.org, no-ip.com, etc.). Next, we discarded domains that have not been in the top 1M list for at least 90% of the time during the entire past year. To this end, we collected an updated top domains list every day for the past year, and only considered as benign those domains that have *consistently* appeared in the top 1M domains list. The purpose of this filtering process is to remove possible noise due to malicious domains that may became popular for a limited amount of time. After this pruning operations, we were left with about $450,000$ popular domain names[6].

- PKIP: we also maintain a list of parking IPs, PKIP. Namely, IP addresses related to *domain parking* services (e.g., IPs pointed to by expired or unused domains which have been temporarily taken over by a registrar). We use this list to prune ExecScent's templates. In fact, CPTs are automatically derived from HTTP requests in malware-generated network traces that are labeled as C&C communications due to their associated domain name being in the CCBL list (Section 5.4). However, some of the domains in CCBL may be expired, and could be currently pointing to a parking site. This may cause some of the HTTP requests in the malware traces to be erroneously labeled as C&C requests, thus introducing noise in ExecScent's CPTs. We use the PKIP to filter out this noise.

- *Threat Analysis*: clearly, it is not feasible to obtain complete ground truth about all traffic crossing the perimeter of the live networks where we evaluated ExecScent. To compensate for this and obtain a better estimate of the false and true positives (compared to only using CCBL and ATWL), we performed an extensive manual analysis of our experimental results with the help of professional

[6]More precisely, second level domains (2LDs).

threat analysts.

5.5.2 System Setup

To conduct our evaluation, we have implemented and deployed a Python-based proof-of-concept version of ExecScent. In this section we discuss how we prepared the system for live network deployment.

5.5.2.1 Clustering Parameters

As discussed in Section 5.4.3, to generate the CPTs, we first apply a request clustering step. The main purpose of this step is to improve the efficiency of the CPT learning process. The clustering phase relies on a hierarchical clustering algorithm that takes in as input the height at which the dendrogram (i.e., the "distance tree" generated by the clustering algorithm) needs to be cut to partition the HTTP requests into request clusters.

To select the dendrogram *cut height*, we proceeded as follows. We considered one day of malware traces collected from our malware intelligence feed (about 8,000 different malware traces). We then applied the clustering process to these traces, and produced different clustering results by cutting the dendrogram at different hights. For each of these different clustering results, we extracted the related set of CPTs, and we tested these CPTs over the next day of malware traces from our feed with varying matching thresholds. The obtained number of false positives, i.e., misclassified benign domains (measured using ATWL), and true positives, i.e., new correctly classified C&C domains (measured using CCBL), are summarized in Figure 22 and Figure 23, respectively. Notice that although in this phase we tested the CPTs over malware-generated network traces, we can still have false positives due to the fact that some malware query numerous benign domain names, along with C&C domains.

As Figures 22 and 23 show, per each fixed CPT matching threshold, varying the dendrogram cut height does not significantly change the false positives and true

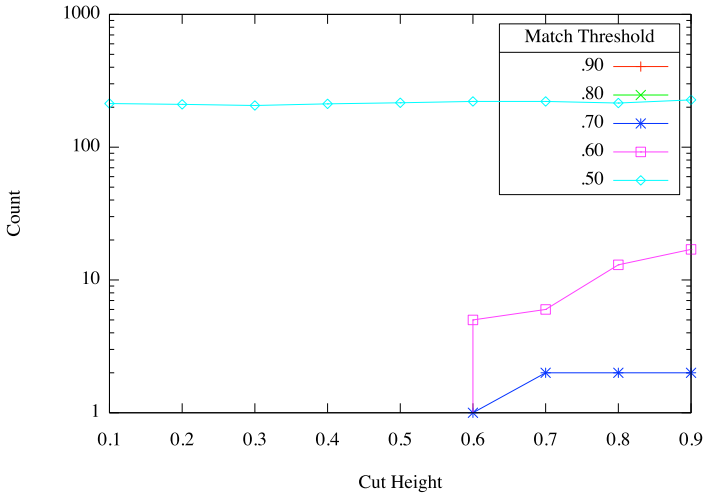

Figure 22: Effect of the dendrogram cut height (FPs).

positives. In other words, the CPT matching results are not very sensitive to the specific value of the clustering parameter. We decided to finally set the value of the cut hight to 0.38, which we use during all remaining experiments, because this provided good efficiency during the CPT generation process, while maintaining high CPT quality.

5.5.2.2 CPT Generation

To generate the CPTs used for the evaluation of ExecScent on live network traffic (Section 5.5.3), we initially used two weeks of malware traces collected from our malware intelligence feed. To label the *seed* of C&C HTTP requests in the malware traces, we used the CCBL black-list. We also use the list of parking IPs PKIP to prune CPTs related to parked C&C domains, as mentioned in Section 5.5.1.3. Once this initial set of CPTs was deployed, we continued to collect new malware traces from the feed, and updated the CPT set daily by adding new CPTs derived from the additional malware traces. More precisely, let D_1 be the day when the initial set of CPTs was first deployed in a live network, and let C_1 be this initial CPT set. C_1 is generated from the malware traces collected during a two-week period immediately

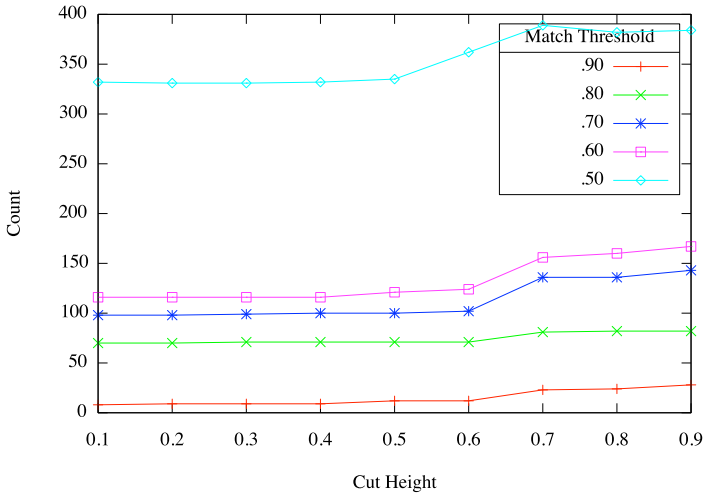

Figure 23: Effect of the dendrogram cut height (TPs).

before day D_1. The CPTs set C_1 was then used to detect new C&C domains during the entire day D_1. At the same time, during D_1 we generated additional CPTs from the malware traces collected on that day, and added them to set C_1. Therefore, at the end of day D_1 we had an expanded set C_2 of CPTs, which we deployed on day D_2, and so on. At the end of the deployment period we had just over $4,000$ distinct CPTs.

To adapt the CPTs to the traffic of each deployment network (see Section 5.4.5), we proceeded in a similar way. We built a background traffic profile based on all HTTP traffic observed at each deployment network during the two days immediately before day D_1, and used this profile to adapt the initial set of CPTs C_1. Then, every day we updated the traffic profile statistics based on the new live traffic observed on that day, and used this information to further adapt all the CPTs. Notice that the set of CPTs deployed to different networks are different, in that they adapt differently to each deployment network (using that network's background traffic profile).

Table 19: Live network results over a two-week deployment period

	UNetA				UNetB				FNet			
Detection Threshold	.62	.65	.73	.84	.62	.65	.73	.84	.62	.65	.73	.84
All C&C Domains	68	66	46	25	36	32	24	10	2	2	2	1
New C&C Domains	35	34	26	13	21	18	15	4	2	2	2	1
Distinct Malware Families	17	17	14	8	14	12	10	4	1	1	1	1
Number of Infected Hosts	114	105	98	37	185	150	147	21	7	7	7	7
Number of New Infected Hosts	91	90	86	25	145	135	133	11	7	7	7	7
FP Domains	133	118	114	0	152	117	105	0	109	63	49	0
FP Domains (reduced CPT set)	25	13	10	0	40	26	22	0	30	23	16	0

5.5.3 Live Network Deployment Results

To evaluate ExecScent, we deployed it in three different large networks, UNetA, UNetB, and FNet, for a period of two weeks. We generated the set of adaptive CPTs as explained above (Section 5.5.2.2), using a total of four weeks of malware-generated network traces (two weeks before deployment, plus daily updates during the two-week deployment period). The CPT matching engine was deployed at the edge of each network.

The detection phase proceeded as follows. For each network, we logged all HTTP requests that matched any of the adapted CPTs with a matching score $S \geq 0.5$, along with information such as the destination IP address of the request, the related domain name, the source IP address of the host that generated the request, and the actual value of the score S. This allowed us to compute the trade-off between the number of true and false positives for varying values of the detection threshold θ. Specifically, let h be a request whose matching score S_h is above the detection threshold θ, and let d be the domain name related to h. Consequently, we label h as a C&C request, and classify d as a C&C domain. We then rely on the CCBL and ATWL lists and on manual analysis (with the help of professional threat analysis) to confirm whether the detection of d represents a true positive, i.e., if d is in fact a C&C domain, or a false positive, in which case d is not a C&C domain.

Figure 24 summarizes the overall number of true positives and false positives obtained during the two-week deployment period over the three different live networks,

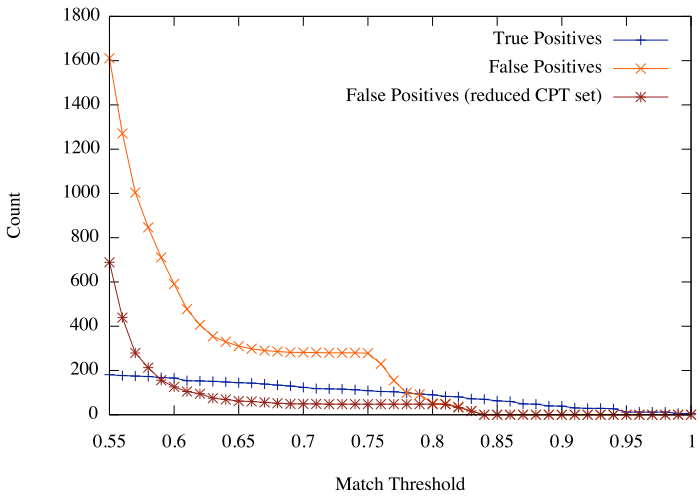

Figure 24: CPT detection results for varying detection thresholds.

while Table 19 shows a breakdown of the results on the different networks for a set of representative detection thresholds. For example, in Table 19, consider UNETA with a detection threshold of 0.65. During the two-week deployment period, we detected a total of 66 C&C domains, of which 34 are new, previously unknown C&C domains that were not present in our commercial black-list, CCBL. The 66 C&C domains were related to 17 distinct malware families. Overall, we detected 105 infected hosts, 90 of which were new infections related to the 34 previously unknown C&C domains. This means that 90 ($\simeq$ 86%) of the infected hosts could not be detected by simply relying on the CCBL black-list.

The CPTs generated 118 false positives, namely domain names that we misclassified as C&C domains. We noticed that most of these false positives were generated by only two CPTs (the same two CPTs generated most false positives in all networks). By subtracting the false positives due to these two "noisy" CPTs, we were left with only 13 false positives, as shown in the last row of Table 19. The false positives marked with "*reduced CPT set*" in Figure 24 are also related to results without these two CPTs. Overall, within the entire two-week test period ExecScent generated a quite manageable number of false positives, in that a professional threat analyst

could analyze and filter out the false C&C domains in a matter of hours.

Notice that the low number (only two) of new C&C domains found in the FNET network was expected. In fact, FNET is a very sensitive financial institution, where many layers of network security mechanisms are already in use to prevent malware infections. However, our findings confirm that even well guarded networks remain vulnerable.

5.5.3.1 Pushdo Downloader

It is worth clarifying that all results reported in Figure 24 and Table 19 have been obtained after discounting the domains detected through a single CPT that was causing hundreds of misclassifications. Through a manual investigation, we easily found that ExecScent had correctly learned this CPT, which actually models the HTTP-based C&C communications of a PUSHDO downloader variant [122]. This particular variant purposely replicates its C&C requests, and sends them to a large number of *decoy* benign domain names. The malware does this to try to hide the true C&C domain in plain sight, among a large set of benign domains. However, while this makes it somewhat harder to find the true C&C among hundreds or even thousands of benign domains (which requires some manual analysis effort), it makes it very easy to identify the fact that the source hosts of these requests, which matched our PUSHDO CPT, are infected with that specific malware variant.

5.5.3.2 UNETB *Deployment Results*

The results we obtained for the UNETB deployment have been obtained in a slightly different way, compared to UNETA and FNET. Because of the higher volume of traffic in UNETB our proof-of-concept implementation of the CPT match engine could not easily keep pace with the traffic. This was due especially to the fact that our match engine software was sharing hardware resources with other production software that have to be given a much higher priority. A few weeks after conducting

the experiments reported here, we implemented an optimized version (written in C, rather than Python) that is almost $8x$ faster; thus, it can easily keep up with the traffic on UNetB.

To compensate for the performance problems of our prototype implementation, during the two-week deployment period we only considered the traffic for every other day. That is, we only matched the CPTs over about seven days of traffic in UNetB, effectively cutting in half the traffic volume processed by ExecScent.

5.5.4 "Static" and URL-Only Models

In this section we compare the results of ExecScent's *adaptive templates*, to "static" (i.e., non-adaptive) templates, which only learn from malware-generated traces and do not take into account the traffic profile of the deployment network, and to URL-based C&C request models, which only use information extracted from URLs.

To obtain the "static" models, we simply took ExecScent's CPTs and "turned off" the specificity parameters. In other words, we set the specificity scores in Equation 1 to zero (with the exception of σ_d, which is set to one), essentially turning the dynamic waits w_k into their static counterparts $\hat{w}_k$ (see Section 5.4.7). In the following, we refer to these static (non-adaptive) templates as "Specificitiy-Off" models.

To obtain the URL-based models, again we "turn-off" the specificity information, and also ignore all components of ExecScent's CPT apart from URL-related components. Effectively, in Equation 1 we only use the similarity functions s_1, s_{2a}, and s_{2b} defined in Section 5.4.7. We refer to these templates as "URL-Only" models.

To perform a comparison, we deployed the ExecScent CPTs and their related "Specificity-Off" and "URL-Only" models to UNetA, UNetB, and FNet for a period of 4 days. Figure 25 and 26 summarize the overall true and false positives, respectively, obtained by varying the detection threshold $\theta \in [0.6, 1]$. As can be seen

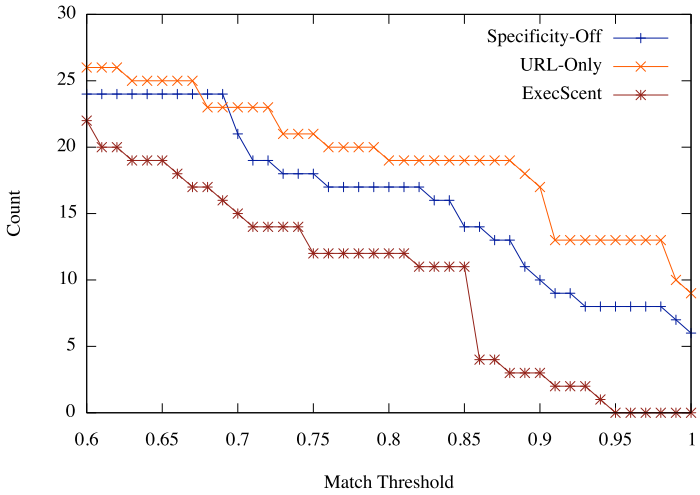

Figure 25: Comparing C&C models - true positives

from the figures, ExecScent's adaptive templates outperform the two alternative models, for detection thresholds $\theta < 0.85$. Unless we are willing to sacrifice a large fraction of all true positives, compared to the numbers obtained at $\theta = 0.6$, the "Specificity-Off" and "URL-Only" models will generate a very large, likely unsustainable, number of false positives (notice the log scale on the y axes of Figure 26).

5.5.5 Deployment in ISP Networks

We were also able to evaluate the results of ExecScent over six large ISP networks serving several million hosts. We proceeded as follows: given 65 new C&C domains discovered by ExecScent during the live network deployment described in Section 5.5.3, we deployed the domains to the six ISPs for an entire week, during which we monitored all DNS traffic. Each day, we counted the number of distinct source IP addresses that queried any of the 65 C&C domains. We found a maximum of 25,584 of distinct source IPs that in any given day queried these C&C domains. In other words, the new C&C domains discovered by ExecScent allowed us to identify 25,584 new potential malware infections across the six ISP networks.

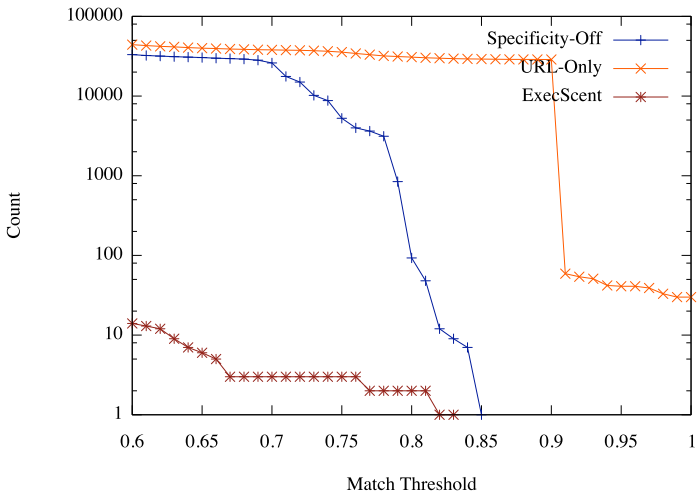

Figure 26: Comparing C&C models - false positives

5.6 Limitations

An attacker who gains knowledge of how ExecScent works may try to avoid detection by mutating her botnet's C&C protocol every time the C&C server is relocated to a new domain. One possible approach would be to implement a new protocol that can be deployed on all the clients (i.e., malware agents) and servers (i.e., malware controllers) before switching to the new domain. However, this would substantially increase the complexity of managing the botnet and hurt its *agility*. Furthermore, for moderate to large botnets the updates would take time to deploy and a mistake in the update procedure could result in losing parts of or the entire botnet.

Another evasion approach may consist in injecting noise into the C&C protocol to make it appear "different". For example, an attacker may randomly generate the C&C URL path or name-value pairs in the query-string, when making a request. However, if a malware agent needs to convey enough information to (loosely) authenticate itself to the C&C server, then at least one request component must have some form of "structured" data. Since ExecScent measures similarity by protocol structure and gives more weight to the shared unique components, it is non-trivial for an attacker

to avoid detection on all deployment networks. In fact, several malware families we detect during our evaluation of ExecScent use such types of techniques to try to avoid detection via regular expressions.

An attacker may also try to "mislead" the detector by injecting noise into the domain name matches. For instance, an attacker may send requests to many decoy benign domains using the same malware C&C requests sent to the true C&C server. This is the approach used by the PUSHDO malware variant we discovered during our evaluation. This type of noisy malware is actually easy to identify, because of the number of unique destination domains contacted by a single host that match one particular CPT within a short period of time. Thus, detecting the infected hosts is easy. However, this makes it somewhat more difficult to determine the true C&C domains among all other domains. In this case, a threat analyst must review the domains, before they can be added to a blacklist; but at the same time, a security administrator can be immediately alerted regarding the infected hosts, thus enabling a prompt remediation.

Blending into the background traffic is another technique that may be used to avoid detection. For example, an attacker may choose "common" data types and values for their C&C protocol components. For some components such as the URL path it may be easy to select a popular value (e.g., "index.html"). However for many of the components, the "commonality" is relative to the deployment network's traffic profile. Therefore, an attacker would need to customize the protocol based on the infected machine's network. This may be difficult to do, because most network hosts have limited or no visibility into the traffic produced by other hosts in the same network. Therefore, although a C&C protocol may carry some "common" components, ExecScent's adaptive CPTs may still be able to use those components that are *specific* (i.e., non-popular) in the deployment network to detect the C&C requests.

Finally, ExecScent's CPTs depends on the malware traces and labeled C&C requests from which they are derived. Thus, ExecScent requires at least one or a few malware samples from a malware family, before its C&C protocol can be modeled and detected. In this case, though, malware code reuse plays to our advantage. A few samples of a malware family whose code has been reused elsewhere (because it was sold or leaked) will in fact facilitate the detection of future malware strains. Note that ExecScent in principle requires only a single sample to generate a CPT, thanks in particular to the request generalization process (Section 5.4.2). That being said, the quality of a CPT can be significantly improved when more than one sample sharing the same C&C protocol are available.

5.7 Related Work

Grouping malware based on features extracted from HTTP requests has beed studied for example in [38, 102, 104, 109]. Specifically, Perdisci et al. [102, 104] proposed a system for clustering malware samples that request similar sets of URLs. In addition, token-subsequences are extracted from the URLs, and used to detect infected hosts on live networks. In [38], information about HTTP request methods and URL parameters are used to cluster similar malware samples. The authors describe their clustering technique as a manual process and mention replacing it with an automated system in the future.

A recently proposed system FRIMA [109] clusters malware samples into families based on protocol features (e.g., same URL path) and for each family creates a set of network signatures. The network signatures are token-sets created from byte strings that are common to a large percentage of the network traffic within a cluster. To reduce false positives, network signatures are pruned by removing the ones that match any communication in the authors' benign traffic pool.

Automated network signature generation has also been studied for detecting worms

[74,98,119]. The generated signatures typically consist of fixed strings or token subsequences that can be deployed in an intrusion detection system. AutoRE [130] extends the automated signature generation process to produce regular expressions that can be used to match URLs in emails for the purpose of detecting spam emails and group them into spam campaigns.

Our work focuses on automatic *template* generation for detecting C&C communications and attributing them to a known malware family. In particular, our main focus is not on clustering malware samples per se. Rather, we apply clustering techniques mainly as an optimization step to generate high quality control protocol templates. Furthermore, we do not limit ourselves to only considering URLs or to extracting sets of common tokens. More importantly, our C&C templates are *adaptive*, in that they learn from the traffic of the network where they are to be deployed, thus self-tuning and automatically yielding a better trade-off between true and false positives.

Jackstraws [67], executes malware in an instrumented sandbox [54] to generate behavior graphs of the system calls related to network communications. These system-level behavior graphs are then compared to C&C graph templates to find new C&C communications. ExecScent is different because it relies only on network information, and does not require malware to be executed in an instrumented sandbox (e.g., it can use traces collected from "bare metal" execution or live networks) to learn the templates. Furthermore, unlike Jackstraws [67], ExecScent learns *adaptive* templates, which allow us to identify new C&C domains in live networks.

5.8 Conclusion

We presented ExecScent, a novel system that can discover new C&C domain names in *live* enterprise network traffic. ExecScent learns *adaptive* control protocol templates (CPTs) from both examples of known C&C communications and the "background traffic" of the network where the templates are to be deployed, yielding a better

trade-off between true and false positives for a given network environment.

We deployed a prototype version of ExecScent in three large networks for a period of two weeks, discovering many new C&C domains and hundreds of new infected machines, compared to using a large up-to-date commercial C&C domain blacklist. We also compared ExecScent's adaptive templates to "static" (non-adaptive) C&C traffic models. Our results show that ExecScent outperforms models that do not take the deployment network's traffic into account. Furthermore, we deployed the new C&C domains we discovered using ExecScent to six large ISP networks, finding over 25,000 new malware-infected machines.

In this and the previous two chapters we explored detection and defense solutions for malware downloads and infections. All of the techniques we discussed rely on deep packet inspection (DPI). The compute resources required by DPI grow with the size of the network and exceed that of single CPU even on small enterprise networks. To scale DPI it must be multithreaded and concurrently process traffic. However, not all packets can be processed concurrently and the way they are scheduled can significantly impact performance. In the next chapter we examine packet scheduling for DPI to maximize performance so our detection and defense systems can protect large enterprise networks.

CHAPTER VI

PACKET SCHEDULING FOR DEEP PACKET INSPECTION ON MULTI-CORE ARCHITECTURES

6.1 Introduction

Deep packet inspection (DPI) is the process of examining the non-header content of a packet by a system that is not an endpoint in the communication. Most DPI systems reconstruct communication streams and maintain state information for large numbers of concurrent connections. When a packet arrives each layer is fully parsed and inspected. State information associated with the communication stream at each layer is updated and stored. All of this work requires many CPU cycles. Fortunately, the work is highly parallelizable; therefore, multi-core architectures can be used to increase the available CPU cycles. So, that raises the question of how packets should be scheduled on a multi-core platform. In this chapter we explore this question using the Protocol Analysis Module (PAM) as our DPI system. The DPI processing performed by PAM is a necessary component of WebWitness, WebSentry and ExecScent described in the previous chapters. Thus, a scheduler that maximizes PAM performance will have a significant impact on the performance of these systems.

PAM [18,19,56,131] is a multithreaded DPI software application that fully parses all protocol layers and emulates the state transitions of both the client and server at each layer. A PAM thread can process the link, internet, and transport layers of any packet in parallel with any other packet. However, at the application layer not all packets can be processed in parallel. For TCP based protocols, packets on the same flow must be processed serially. In fact, TCP segments on a flow must be processed in sequence order because of the way PAM emulates the state transitions of client and

server applications, which is necessary for reconstructing HTTP transactions required by the systems described in this thesis. In addition, a small number of UDP based protocols must be processed serially as well and in application layer sequence order because they implement sessions at the application layer. However, the majority of UDP packets and other protocols can be processed in parallel.

In this chapter we perform the following:

- Design and implement two new DPI packet scheduling algorithms. One is designed to maximize work balance and the other cache affinity.
- Compare our two packet scheduling algorithms against a packet scheduling algorithm commonly used in network applications.
- Demonstrate that scheduling packets for cache affinity is more important than balancing the work load. In fact, for one of the network captures we used in our evaluation, scheduling packets for cache affinity improved throughput by 38%.

The remainder of this chapter is organized as follows. The next section presents related work. In Section 6.3, we describe the packet scheduling algorithms we designed, implemented, and evaluated. In Section 6.4, we explain our testing methodology and present the results. In Section 6.5, the chapter is concluded.

6.2 Related Work

Packet distribution, load balancing, and scheduling are the terms typically used for the process of assigning a packet to a resource that will perform work on it (in this chapter these terms are used interchangeably). Packet scheduling occurs at many places on the network. Packets are scheduled on physical links for multi-path routing [84], physical links for link aggregation [17], distributed network processors for high-speed links [51], forwarding engines for parallel forwarding [116–118], transactions for cluster-based Internet services [79], and L7-filter threads for QoS [20, 61, 62]. Although the work

being scheduled varies widely, many of the concepts and techniques can be applied across the packet scheduling domain.

6.2.1 Packet Scheduling Overview

6.2.1.1 Packet Based Scheduling

In packet based scheduling, packets are assigned to resources on a packet-by-packet basis. Round-robin is an example of a simple packet based scheduling algorithm. More complex algorithms assign packets to the least loaded resource (e.g., the resource with the smallest number of outstanding bytes to process). Packet based scheduling does a good job of balancing the load across resources. However, packet reordering can occur and impact network performance. In addition, packet based scheduling in shared memory systems can be cache inefficient resulting in reduced performance. Packet based scheduling algorithms are commonly found as a scheduling option in network devices and processors.

6.2.1.2 Flow Based Scheduling

Flow based scheduling [62] maintains a table of active flows where each flow is associated with a resource. Packets are mapped to flows and scheduled on the resource associated with their flow. New flows are assigned to the least loaded resource. Since all packets on the same flow are assigned to the same resource, per flow packet order is maintained. Also, it is cache efficient for the same reason. There are a few drawbacks to this method. First, a table of active flows must be maintained. This table can require a large amount of memory when there are many active flows. In addition, it can take a significant number of clock cycles to perform the lookup. Finally, flows are not equal in the number of packets, bytes, and processing associated with them. Thus, it is difficult to assign new flows to resources so that the work is balanced.

6.2.1.3 Fixed Hash Scheduling

Direct hash and indirect hash [40] are the two most common fixed hash scheduling algorithms. Direct hash applies a hash function to a subset of the 5-tuple and uses the result modulo the number of resources to determine the resource assignment. Indirect hash uses the result of the hash function modulo the size of the indirection table as an index into it. Each bin in the indirection table is associated with a resource and packets that map to a bin are processed by that resource. Indirect hash allows for unequal resource weights and the indirection table can be tuned for adaptive hash scheduling. For fixed hash scheduling, packets with the same 5-tuple hash to the same value; so, they are assigned to the same resource. Therefore, per flow packet ordering is maintained and it is cache efficient. In addition, it is stateless since there is no table to maintain. The downside to this method is lack of control over resource assignment and this can result in load imbalances. Fixed hash scheduling algorithms are commonly found as a scheduling option in network devices and processors.

6.2.1.4 Adaptive Hash Scheduling

Adaptive hash scheduling [11, 61, 84, 116] attempts to combine the simplicity of fixed hash scheduling with the ability to change the resource assignment when the load is imbalanced. Receive-side scaling (RSS) [11] is an adaptive hash scheduling algorithm used to balance packets arriving on a network adapter to CPUs. RSS uses indirect hashing to schedule packets. When a load imbalance is detected, the host protocol stack tries to balance the traffic by calculating a new indirection table. The authors of [84] describe and evaluate several adaptive indirect hashing algorithms for multi-path routing. Adaptive hash scheduling can improve fixed hash scheduling load imbalances. However, it adds overhead and the adjustments are reactive typically occurring after the network has been impacted.

6.2.1.5 Flow Burst Scheduling

Flow burst scheduling [51, 71, 118] tries to combine the workload balance of packet based scheduling with the cache affinity and per flow packet ordering of flow based scheduling. As in flow based scheduling, a flow table is maintained; however, it only needs to contain flows that have packets in the system. A flow entry contains the number of packets currently in the system or a timestamp of the last packet that mapped to that flow. When a packet arrives it is mapped to a flow entry using a subset of the 5-tuple. If flow entry exists and there are packets in the system on that flow, the packet is assigned to the resource processing the other packets. On the other hand, if the flow entry does not exist or there are no packets in the system on that flow, the packet is assigned to the least loaded resource. Since all packets on the same flow in the system at the same time are processed by the same resource, it is cache efficient and per-flow packet ordering is maintained. Also, the balance of the work is better than flow based scheduling because flows are not fixed to a resource and can be reassigned between packet bursts. However, maintaining the flow entries can be expensive and imbalances can still occur.

6.2.2 Packet Scheduling For DPI

Most of the packet schedulers in the literature are designed for applications that are not as complex as DPI. The information used to make scheduling decisions for those applications typically does not apply to DPI. For instance, the number of packets or bytes enqueued to a thread. The applications in the literature essentially have a fixed processing time per packet or byte. In contrast, the number of instructions PAM needs to process a packet or byte varies dramatically depending on the protocol and its attributes. For example, a packet with an application layer protocol that PAM does not recognize may require 600 clock cycles to classify; whereas, an identically sized HTTP packet with compressed content may require 30,000 clock cycles to inspect.

Thus, it is difficult to determine the amount of work assigned to a PAM thread when making a scheduling decision.

Maintaining per-flow packet order on egress is an important feature of most packet schedulers. This is because they were designed for inline devices and packet reordering within a flow can have a negative impact on the performance of the network. This is due to the fact that reordering packets within a TCP flow can result in duplicate data segment transmissions, a reduced data transmission rate, and burstiness [32,35]. Requiring per flow packet order on egress does not always mean that packets cannot be processed out-of-order. There are systems that restore per flow packet order on egress [57]. Also, per flow packet ordering at egress is only important for systems that are inline and DPI systems are not always deployed inline. However, inline or not, PAM requires that TCP segments be processed in sequence order. If PAM receives a TCP segment out-of-order it saves a copy until the missing segments arrive. This can impact performance when a large number of TCP segments are received out-of-order. So, maintaining per flow packet ordering for packet processing can improve PAMs performance by reducing the number of TCP segment that must be saved.

As for the multithreaded state of other DPI systems, Snort 3.0 [14, 114] is currently available in beta for download. It supports a multithreaded execution model that allows multiple analysis engines to operate on the same traffic simultaneously. Therefore, unlike PAM it does not utilize packet and connection level parallelism. In the case of Bro [101], a multithreaded version is not currently available. However, in [100] the authors describe an architecture that could be used to create a multithreaded Bro. The architecture uses a flow based scheduling algorithm to distribute packets for analysis. Thus, an improved packet scheduling algorithm could significantly increase performance.

When deployed inline, the goals of a DPI packet scheduler are to maximize

throughput, minimize average latency, bound maximum latency (e.g., one millisecond), and minimize the number of packets that are reordered within a flow. When a DPI system is not inline, most of those goals are no longer important. In fact, the goals become maximize throughput and bound maximum latency at a much higher value (e.g., one second).

6.3 DPI PACKET SCHEDULERS

The goal of packet scheduling for DPI is to maximize the amount of network traffic that can be inspected without noticeably impacting it. The ideal packet scheduler has the following properties:

- Load Balancing: Work is evenly distributed across all threads.
- Low Scheduling Overhead: The cost of scheduling packets (in terms of memory and CPU cycles) is very small in comparison to the work performed on the packet.
- Per-Flow Ordering: Packets on the same flow at egress are in their arrival order.
- Cache Affinity: Packets are scheduled on threads that have their associated data structures already in cache.
- Minimal Packet Delay Variation: Minimal variation in the latency added to packets on the same flow.

In this section we describe the design and implementation of the three packet scheduling algorithms we evaluate with PAM. The first algorithm we describe is Direct Hash (DH). It is an algorithm that is commonly used to schedule packets and we use it in our evaluation for comparison purposes. The other two algorithms are of our own design. We created Packet Handoff (PH) to maximize load balancing. However, to maximize this property, tradeoffs were made such as per-flow ordering and cache

affinity. The other algorithm we designed is Last Flow Bundle (LFB). Its goal is to maximize cache affinity. But, like PH it sacrifices other properties (e.g., packet delay variation). No single packet scheduler can have all of the properties of our ideal packet scheduler; so, our goal is to determine the properties that are most important for achieving maximum inspection with minimal network impact.

The DH and LFB packet schedulers only use the source and destination IP addresses instead of the entire 5-tuple for the flow identifier (FID). We chose not to include the transport layer ports because that would require more parsing, they are not included in fragmented packets (except for the first fragment), and it did not have a significant impact on the distribution of packets with our network captures. As for our hash function, we chose a 16-bit CRC because it is known to provide good load distribution [40]. In addition, we always hash the smallest IP address first so that packets in both directions on a connection go to the same thread.

6.3.1 Direct Hash (DH)

Direct Hash (DH) is a simple fixed hash scheduling algorithm that is widely used; that is why we selected it for comparison. When a packet arrives, the packet scheduler parses the data link and network layers to extract the FID. The FID is then hashed and the result modulo the number of threads determines the PAM thread that will process the packet. DH has several appealing properties:

- Stateless: There is no state to maintain. The information needed to distribute the network traffic is contained in each packet. So, there is no additional memory overhead or table lookups.

- Per-Flow Ordering: By using a subset of the 5-tuple as input to the hash function, packets on the same flow will hash to the same value so they will be processed by the same thread. Thus, per-flow packet order is maintained.

- Cache Affinity: Packets on a flow are always processed by the same thread. So only memory associated with the flows assigned to a thread will be in the processors cache. This increases the likelihood of a cache hit when processing a packet. Furthermore, packets on a flow often arrive in bursts, referred to as packet trains in [68], resulting in temporal locality that can be used to improve the cache hit rate by assigning the packet train to the same processor.

The drawbacks of using DH are:

- Load Imbalance: There is no control over how packets are distributed. The authors of [117] prove that a direct hash on FIDs cannot balance the workload due to the Zipf-like [134] probability distribution of flows found in real-world network traffic.

- Header Parsing: the packet scheduler must understand how to parse all of the protocol headers that contain FID fields. In addition, it must be able to skip over lower level headers to reach FID headers. Network traffic, even at the lower levels, can be complicated (e.g., contain multiprotocol label switching (MPLS), virtual LAN (VLAN), stacked VLANs, IPv6 extension headers). Therefore, the packet scheduler must be complex enough to extract the FID fields on these networks.

- High Distribution Overhead: The header parsing, hashing, and distribution are in the data plane and executed on every packet. So it is important for them to be efficient. However, esoteric network traffic and the complexity of a good hash function can make the distribution of packets the bottleneck.

6.3.2 Packet Handoff (PH)

Packet Handoff (PH) is a packet scheduling algorithm we designed with the goal of maximizing the concurrency available in DPI. It is a packet based scheduling algorithm, where the least loaded thread gets the next packet. Therefore, this algorithm is the best at distributing the work evenly across threads. Two types of queues are used to distribute packets. When a packet is received the packet scheduler places the packet in the receive queue (RQ). Threads that are not busy (i.e. not currently processing a packet) go to the RQ to get their next packet. If the RQ is empty they wait (i.e. spin) for a packet to arrive.

After a thread dequeues a packet from the RQ it parses the link, internet, and transport layers of the packet. If the packets transport protocol is TCP the 5-tuple is extracted by the PAM thread and used as the FID. PAM tracks the transport and application layer state associated with a TCP connection in a connection entry (CE). All CEs are stored in a connection table and the FID is used to map a packet to a CE in the table. The connection table is shared by all threads. If a CE mapping is not found, a new entry is created and the thread sets itself as the owner (this is done by setting a flag in the CE). If the thread finds the CE, it checks to see if another thread owns it. If it is not owned, the thread sets itself as the owner. If the CE is owned, this indicates that another PAM thread is currently processing a packet associated with it. Packet on the same TCP flow must be processed in sequence order. Therefore, instead of waiting for the owning thread to complete its processing and release ownership, the thread enqueues the packet in the connection queue (CQ) associated with the CE and returns to the RQ for its next packet.

Once a thread owns a CE, it processes the application layer of its current packet and updates the CE as needed (no synchronization required). When the thread is finished processing its current packet it checks to see if any packets have been placed in the CQ associated with the CE it owns. If no packets are in the CQ it removes

itself as the owner of the CE (by setting a flag in the CE) and proceeds to the RQ to for its next packet. If there is a packet in the CQ, the thread retains CE ownership and dequeues its next packet from the CQ instead of the RQ.

A PAM thread can only own one CE at a time. CE ownership occurs when a packet maps to a CE that is not currently owned. A PAM thread releases CE ownership when it finishes processing its current packet and the CQ associated with the CE is empty. Any PAM thread can enqueue a packet in the CQ associated with a CE, but only the current owner of the CE can dequeue packets. Packets that are enqueued to a CQ receive the cache benefits of having their application layer processed by the same thread. However, when CE ownership is released there may be packets in the system (either being processed or in the RQ) that map to the CE. There is a high probability that those packets will be processed by a different thread and interleaved with packets on other connections; thus, they will not receive cache affinity benefits.

The advantages of PH are:

- Load Balancing: Threads pull packets from a queue (i.e. from the RQ) when they are not busy. The link, internet, and transport layer of all packets are processed in parallel. Packets are only enqueued to a PAM thread when it owns a CE and there are other packets being processed by other threads that map to the owned CE. If n PAM threads are processing packets and there are at least n packets in the receive queue that map to different CEs, no threads will be idle.

- Low Distribution Overhead: The packet scheduler does not need to parse packet headers or compute a hash. All it does is enqueue packets into the RQ. Therefore, the scheduling overhead is low.

The disadvantages of PH are:

- Out-of-Order Packets: Per-flow packet ordering is not preserved. If two packets on the same TCP flow are processed at the same time by different threads,

there is a race to acquire ownership of the CE. If the thread with the higher packet number acquires ownership, then a copy of the packet will be saved. In addition, if PAM is inline the packet will be transmitted out-of-order.

- Cache Affinity: There is no mapping of packets that are part of the same flow to the same thread. Therefore, cache lines associated with CEs move from processor to processor as they are looked up and updated. In addition to the cache coherence overhead, the amount of cache lines available to store connections is reduced, increasing memory accesses.

- Queue Overhead: All threads compete to dequeue packets from the RQ. In our implementation, RQ contention was not an issue. This was due to the fact that our average packet size was over 300 bytes for all network captures and there were at most 14 threads pulling packets from the RQ. A smaller average packet size or more threads could cause the RQ to become the bottleneck. In addition to the RQ, CQ contention can limit performance when a large percentage of the packets are handed off to a single thread.

6.3.3 Last Flow Bundle (LFB)

Last Flow Bundle (LFB) is a flow burst scheduling algorithm we designed with the goal of maximizing cache affinity. It schedules packets similar to the way Last Processor (LP) schedules tasks [120]. The idea is to process all of the packets in the system that map to the same flow on a single thread and not interleave the processing of packets that map to other flows. LFB uses two types of queues, a single receive queue (RQ) and a table of flow bundle queues (FBQ), to distribute packets.

As in the DH method, when a packet arrives it is processed by the packet scheduler to extract the FID that will be used as the input to the hash function. However, instead of using the result of the hash function to select the thread that will process the packet, it is used to map the packet to a FBQ in the FBQ table. If the selected

FBQ is empty, then the packet is enqueued in it and the RQ. On the other hand, if the selected FBQ is not empty, the packet is only enqueued in it.

When the system is initialized, all PAM threads proceed to the RQ to get their first packet. After a packet is processed by a PAM thread, it is dequeued from the associated FBQ (i.e. the FBQ the packet scheduler mapped the packet to) by the PAM thread. If the FBQ is not empty after the newly processed packet is dequeued, the PAM thread selects the packet at the head of the FBQ to process next, but does not dequeue it. This process repeats until the FBQ is empty. Once the FBQ is empty the PAM thread returns to the RQ for its next packet.

By enqueuing the packet in both the RQ and the selected FBQ, when the selected FBQ is empty, the packet scheduler permits any non busy PAM thread (i.e. a PAM thread not currently associated with a FBQ) to process it. However, by enqueuing the packet only in the selected FBQ when it is not empty, the packet scheduler ensures that only the PAM thread that is currently associated with that FBQ will process it. PAM threads dequeue packets from the RQ before they process them. Contrarily, PAM threads dequeue packets from the FBQ after they have fully processed them. Leaving a packet in the FBQ during processing informs the packet scheduler that a PAM thread is currently associated with that FBQ. The FBQ association occurs when a PAM thread dequeues a packet from the RQ. A PAM thread is associated with a FBQ until it is empty. A packet is placed in the RQ only when there are no other packets in the system mapping to the same FBQ. Therefore, at any given time, there is at most one packet in the RQ that maps to a given FBQ.

The advantages of LFB are:

- Per-flow Ordering: Packets on the same flow hash to the same FBQ. One PAM thread is assigned to a FBQ until all of the packets in the system that map to it have been processed. Thus, packets on a flow cannot be reordered because they are processed serially in the order they arrived in the system.

- Cache Affinity: Packets that map to the same FBQ and are in the system at the same time are processed by a single thread. So packet trains will be processed on the same processor increasing the likelihood of a cache hit on the data structures associated with the flow. In addition, a thread will choose to process a packet that maps to the previous packets FBQ over packets that arrived earlier, but did not map to the same FBQ. This increases the cache hit rate by eliminating interleaved flow processing that could evict cache lines associated with the previous packets flow.

- Load Balancing: The FBQ table is over two orders of magnitude larger than the number of PAM threads. So, the number of flow collisions is small compared to DH. FBQ assignments are dynamic. A PAM thread is only assigned to a FBQ while there are packets that map to it in the system. When a thread finishes processing the last packet in the system on a FBQ, it goes to the RQ to get its next packet. If there are n threads processing packets and there are at least n packets in the system that map to unique FBQ, no thread will be idle.

The disadvantages of LFB are:

- Header Parsing: Like direct hash, the packet scheduler must be complex enough to skip over the preceding protocol layers and extract the FID.

- Increased Packet Delay Variation (PDV) [12]: Packets on the same flow are intentionally processed together for improved efficiency. However, this causes packets in a packet train to jump ahead of older packets in the queue resulting in higher latency for those packets. If enough packets jump ahead the PDV will increase.

- High Distribution Overhead: In addition to the overhead described in the DH section, LFB requires the packet scheduler to enqueue a packet in two places if it maps to an empty FBQ.

6.4 PERFORMANCE EVALUATION

This section describes our testing methodology, the data used for the evaluation, and the results. Our goal was to understand the impact each scheduling algorithm has on throughput and latency. Therefore, for each packet scheduling algorithm we measured the following:

- Maximum raw throughput: This measures how fast a packet scheduling algorithm can process a network capture and ignores packet timing information.
- Maximum scaled throughput: This measures how fast a packet scheduling algorithm can process a packet capture using the timing information in the capture.
- Average packet latency: The average amount of time a packet spends in the system.
- Maximum packet latency: The maximum time that any packet spent in the system.

6.4.1 Testing Methodology

We designed a test harness that takes a network capture as input and collects the throughput and latency measurements described above. The test harness begins by loading all of the packets from the capture into memory to eliminate disk I/O from the measurements. The packets are parsed by the packet scheduler during load to extract the FID and calculate the hash. All of the PAM threads wait on a barrier during this phase. When all of the packets are in memory they proceed through the barrier and record the value of the cycle counter. After the packet scheduler enqueues the last packet in the capture, it enqueues sentinel packets to let the PAM threads know they are done. When a PAM thread dequeues a sentinel packet it calculates the difference between the value of the current cycle counter and the recorded start

value. The largest cycle count recorded by a PAM thread is then used in the time and bandwidth calculations.

6.4.1.1 Measuring Throughput

To measure the maximum raw throughput, the packet scheduler enqueues packets as fast as it can and only pauses when a queue is full. The largest cycle count recorded by a PAM thread is then used to calculate throughput. Measuring the maximum scaled throughput is more complicated. After the packets have been loaded into memory, the packet scheduler scans the timestamps recorded for each packet during capture to calculate the average and maximum bandwidth (we define the maximum bandwidth to be the maximum bit rate sustained for a minimum of one millisecond). Then the packet scheduler scales the timestamp of each packet so that the new average bandwidth matches a program argument provided to the test harness. When processing begins, the packet scheduler uses the cycle counter and the adjusted timestamp of each packet to enqueue them at their new specified time. Periodically, the packet scheduler will check on the number of outstanding packets. If that number exceeds a specified maximum value it considers those packets dropped. In an actual DPI appliance, there are a fixed number of packet buffers. Once that value is exceeded, incoming packets are dropped. We consider PAM to be oversubscribed if it drops a packet; thus, unable to perform at that bandwidth. We chose 4096 as the maximum number of outstanding packets because latency starts to increase significantly when it is exceeded.

6.4.1.2 Measuring Latency

Packet latency is measured by recording the value of the cycle counter when a packet is enqueued and calculating the difference in the cycle counter after a PAM thread processes it. The latency associated with receiving and forwarding of packets that would be present in a DPI appliance is not measured by our test harness. However,

our goal is to measure the latency relative to each scheduling algorithm so we believe our measurement to be sufficient. Our latency measurements were taken at 80% of maximum scaled throughput.

6.4.1.3 Evaluation Platform

We evaluated the packet scheduling algorithms on a system running a Linux 2.6 kernel with two 2.53 GHz Intel Quad-Core Xeon E5540 processors [21] with 4 gigabytes of RAM. Hyper-threading was enabled and processor affinity was set so that each PAM thread and the packet scheduler executed on different hardware threads. In addition, software threads were assigned so that no two executed on the same core unless there were more software threads than cores. Furthermore, the priority of all software threads was set to real-time.

The Xeon E5540 processors are based on Intels Nehalem microarchitecture. They have a memory controller per processor and are connected via the Quick Path Interconnect (QPI) to create a ccNUMA architecture. Each processor has 4 cores and each core has its own L1 (32 KB instruction/32 KB data) and L2 (256 KB) cache. All of the cores on a processor share an inclusive L3 (8 MB) cache. The cache line size is 64 bytes. Cache coherency between processors is maintained by the MESIF protocol.

6.4.2 Network Captures

We used three network captures to evaluate the performance of the packet scheduling algorithms. They consist of real network traffic from locations in networks where a DPI appliance is deployed. They contain the entire payload of every packet so that we can determine the number of clock cycles PAM requires to process each packet. The clock cycle measurements in this section were taken by recording the clock cycle counter just before entering PAMs packet processing function and then recording the difference when PAM returned. Table 20 shows the following information for each

capture:

- Packets: The total number of packets.
- Connections (Conns): The total number of unique TCP and UDP connections.
- Average Mb/s: The average bandwidth of the network traffic during capture.
- Maximum Mb/s: The maximum bandwidth of the network traffic that was sustained for a minimum of one millisecond during capture.
- Average Cycles Per-Packet (CPP): The average number of clock cycles PAM expends processing a packet.
- Connection Density (CD): The average number of unique connections per one hundred packets.

Table 20: Network capture attributes.

	Packets	Conns.	Avg. Mb/s	Max Mb/s	Avg. CPP	CP
Dominant	454,988	2,224	13	38	4,814	7
Many	1,567,397	105,691	27	74	5,658	50
Balanced	1,236,710	43,357	57	75	5,203	29

=

We chose these three captures because we wanted to evaluate the packet scheduling algorithms under varying network conditions using real network traffic. Figure 27 shows the clock cycle distribution of the top ten connections from each capture. The top ten connections in the Dominant capture are responsible for over 50% of the total clock cycles. The traffic in the capture is dominated by a single TCP connection. It is responsible for over 50% of the packets and 27% of the clock cycles. The top ten connections in the Many capture are responsible for less than 0.4%. This means there is a much lower variance in clock cycles spent on each flow than in the other captures. In addition, packets on the different connections are spread throughout the capture having an average 50 unique connections per 100 packets. The Balanced captures

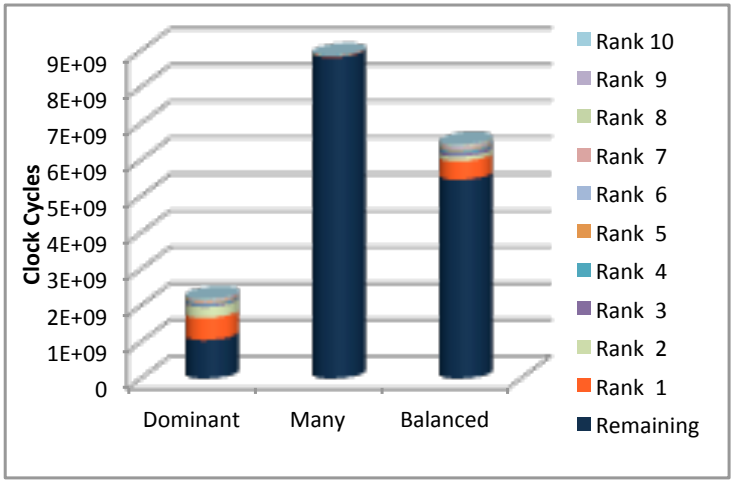

Figure 27: Top ten connections by clock cycle.

top ten connections contribute about 15% of the total clock cycles. The dominant connection in the Balanced capture is responsible for 8%. This places the Balanced capture between the other two in terms of connection clock cycle distribution.

6.4.3 Throughput Results

6.4.3.1 Dominant Capture Raw Throughput

Figure 28 shows the maximum raw throughput results for the Dominant capture. The LFB algorithm has the highest throughput at every thread count tested. It shows near linear scalability from 1 to 4 threads. After that, as more threads are added, the throughput increases cease. This is because of the dominating TCP connection that is responsible for the majority of the packets. At 5 threads and above, a single thread processes all of the packets on that connection and nothing else. So, 3.8 gigabits per second is how fast a single thread can process that connection. Adding additional threads only decreases the average latency of the packets on the other connections. At 14 threads the performance decreases. This is due to the fact that the hardware thread on the core that inspected the dominating TCP connection shared cycles with another hardware thread that processed 11,803 packets on other connections.

The LFB algorithm performed well on this capture for two reasons. First the

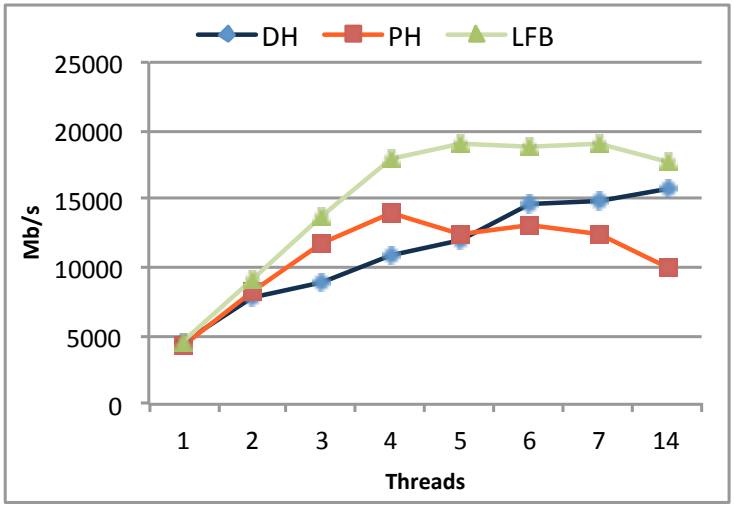

Figure 28: Dominant Capture - raw throughput.

work was optimally balanced in the sense that a single thread only processed packets on the dominating TCP connection. The work balance cannot be improved because PAM requires that packets on the same TCP connection be processed sequentially. Second the work was optimally scheduled for performance. Every packet on the dominating TCP connection was processed on the same core and no packets from other connections were processed on the core. Therefore, the data associated with the dominating TCP connection stayed in cache, improving the cache hit percentage, resulting in fewer cycles per instruction.

The DH algorithm also received the cache benefit of processing the dominating TCP connection on the same core. However, the thread that processed it also had to process packets on other connections. The dips and peaks in the graph show the thread numbers that resulted in more or less packets mapping to the thread that processed the dominating TCP connection. At 14 threads, no other packets mapped to the thread processing the dominating TCP connection. However, the hardware thread sharing the core ultimately limited the throughput to approximately 12% lower than LFB.

The PH algorithm had the worst performance. This is because at 14 threads over

67% of packets were enqueued (i.e. handed off) to other threads. As the number of threads increased so did the number of handoffs. Handing off a packet to another thread is expensive because the data link, internet, and transport layer information is copied and enqueued with the packet to avoid processing those layers again. Also, enqueuing a packet to another thread requires an atomic operation since there could be multiple producers (i.e. multiple packets on the same connection being enqueued by different PAM threads). For these reasons the performance decreased with more than 4 threads.

6.4.3.2 Many Capture Raw Throughput

The Many capture does not contain a single TCP connection that dominates the bandwidth or the packet processing cycles. In fact, the fastest TCP connection is responsible for less than 0.4% of the total packets. Figure 29 shows the maximum raw throughput results for the Many capture. LFB has the highest throughput at every thread count tested. This is because of the cache benefits each thread gets from preferring to process a packet on the flow bundle it just processed over packets that may have arrived earlier. Even when there is just 1 thread, LFB performs better than the other algorithms because of this. At 14 threads LFB is around 25% faster than DH.

DH performs better than PH because the workload is almost perfectly balanced and it, like LFB, gets the cache benefits of having the same thread process every packet on a flow. Nevertheless, because there are more interleaving packets on different flows, it does not perform as well as LFB. PH is about 15% slower than DH even with less than 10% of the packets being handed off to other threads. The slower performance is due to cache misses from the data associated with the packets connection entry not being in cache.

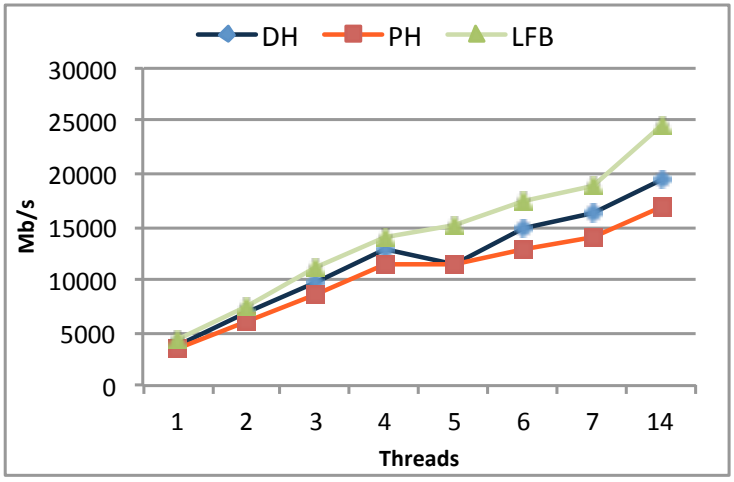

Figure 29: Many Capture - raw throughput.

6.4.3.3 Balanced Capture Raw Throughput

The throughput results for the Balanced capture are presented in Figure 30. Again, LFB has the highest throughput at every thread count tested because of the cache benefits of the scheduling algorithm. At 14 threads it is 38% faster than DH. DH performs well at 5 threads, but does not do as well at 4, 6, 7, and 14 due to load imbalances. In fact, PH outperforms DH at 4 threads. This shows that a load imbalance can outweigh the cache benefits if it is large enough. However, cache misses limit PHs scalability above 4 threads.

6.4.3.4 Scaled Throughput

The maximum scaled throughput results for all captures and algorithms are shown in Figure 31. The throughput for all algorithms decreased in comparison to the maximum raw throughput because of bandwidth spikes that are from 1.5 to 3 times the average and last for several milliseconds.

Notice that the throughput decline of the LFB algorithm for the Dominant capture is significantly less than the others. We suspect this is due to the fact that it processes packets in a different order than the rest. For example, assume there is a

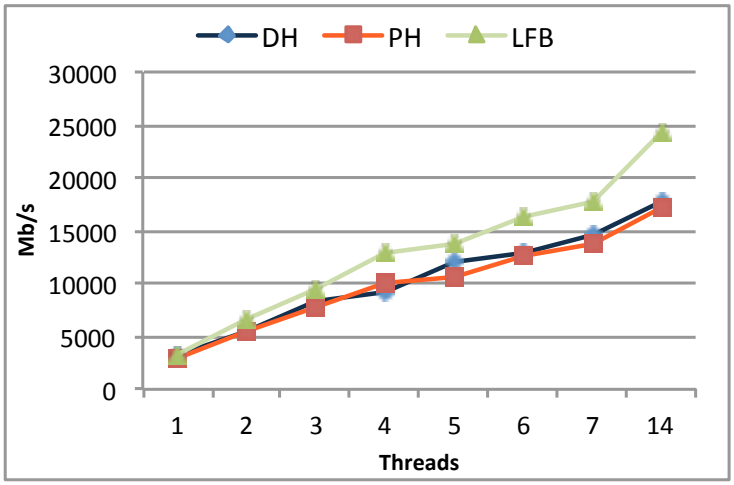

Figure 30: Balanced Capture - raw throughput.

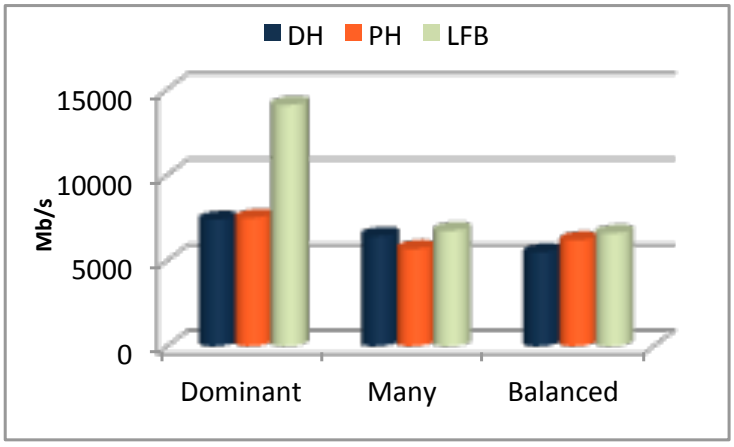

Figure 31: Scaled throughput.

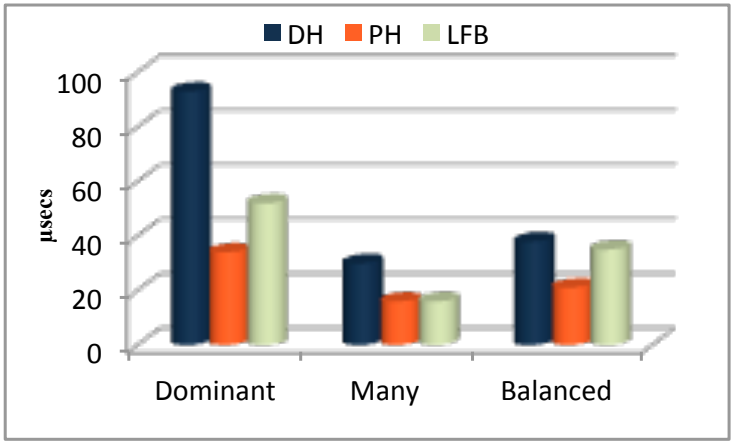

Figure 32: Average latency.

three millisecond burst of packets. If the packets that are least expensive, in terms of clock cycles, are processed first then more packet buffers will be available for arriving packets than if they were processed in their original order; thus, increasing the maximum scaled bandwidth. Another interesting observation is that PH performed better than DH on the Dominant capture and the Balanced capture. This is due to load imbalances during packet bursts resulting in dropped packets; therefore, reduced throughput.

6.4.4 Latency Results

Figure 32 shows the average latency of all three packet scheduling algorithms for each capture. These numbers represent the mean time taken to process a packet at 80% of the maximum scaled throughput for the packet scheduling algorithm. The latency numbers are all below 100 microseconds. DH has the highest average latency for all three captures. This is because of work imbalances causing packets to wait in queues even when the system is lightly loaded.

The maximum latency results are presented in Figure 33. LFB has the highest maximum latency for the Dominant and Balanced captures. This is due to the order the packets are processed. During a burst of traffic, packets in the RQ can starve

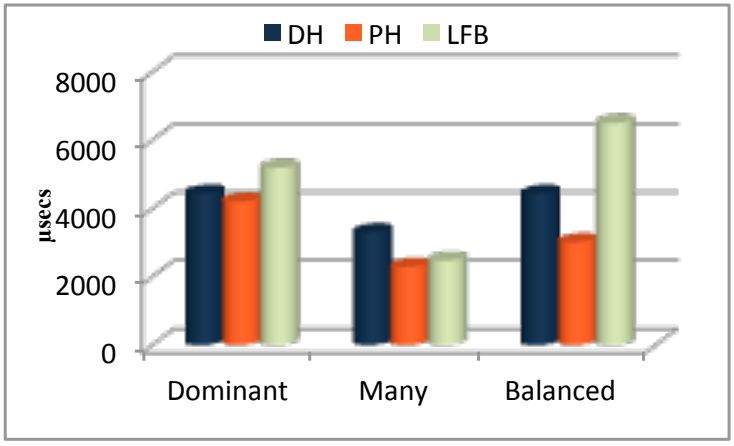

Figure 33: Maximum latency.

while packets on an index in the FBQ are processed. This causes the packets sitting in the RQ to accumulate latency while waiting to be processed. This does not happen with the Many capture because of the high connection density and the large number of connections that prevent RQ starvation.

6.4.5 Cache Measurements

Figure 34 shows the average number of L1, L2, and L3 cache misses per packet with seven PAM threads for the three packet scheduling algorithms and network captures. For the Dominant capture we see the considerable number of cache misses for PH resulting from the large number of packet handoffs. As for DH and LFB, the average number of cache misses per packet is almost identical for the Dominant capture; yet, LFB had 29% higher throughput. This is because DH processed around 8% more packets on the core that processed the dominant connection. These additional packets were more expensive to process due to their associated state not being in cache causing the average cache misses per packet to be higher on the dominant core for DH (see Figure 35). So, more packets plus a higher average number of cache misses per packet caused DH to be slower.

The Many capture produced the highest number of cache misses per packet for DH

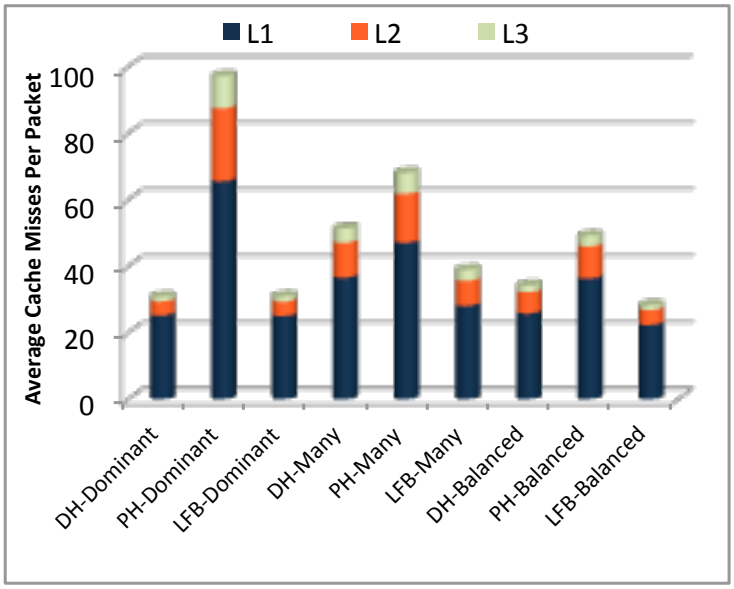

Figure 34: Average cache misses per packet.

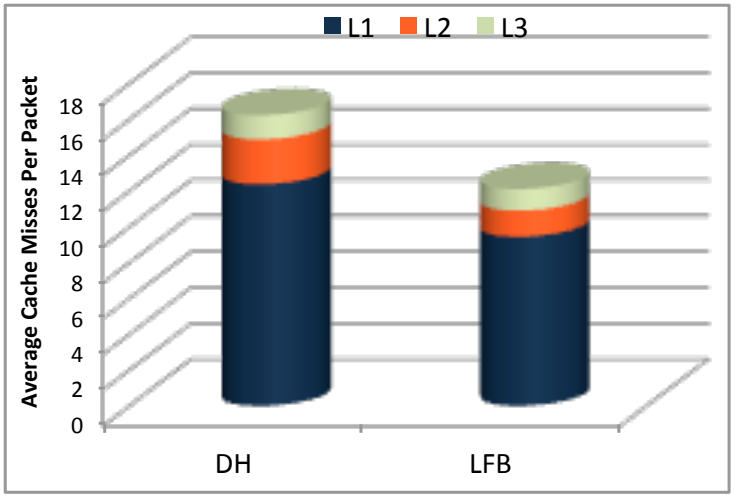

Figure 35: Dominant core average cache misses per packet.

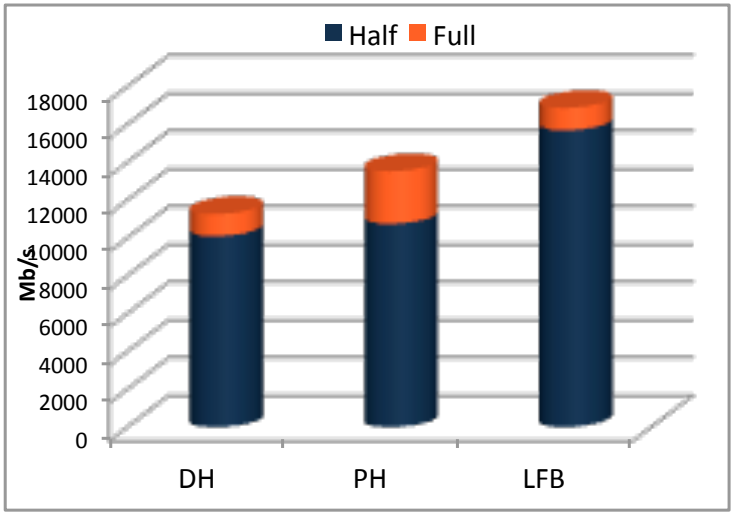

Figure 36: Dominant Capture with half L3 Cache.

and LFB. This is due to the large number of concurrent connections (i.e. connection density) in the capture that results in more interleaved connection processing. As for the Balance capture, LFB had the fewest cache misses, as it did for all of the network captures. DH, in general, had more cache misses than LFB because of load imbalances and more connection interleaving.

In order to determine how the systems cache size impacted the performance of the three scheduling algorithms, we measured their throughput with half of the L3 cache available. Since there is no way to disable half of the L3 cache on the processor, we created a cache clobbering thread to run on one core of each processor. The cache clobbering thread allocates 4MB of cache aligned memory and reads the first byte of each cache line in a loop; thus, half of the L3 is invalidated with each pass. Figure 36, 37 and 38 show the results of these experiments.

For the Dominant capture, LFB performance only declined by around 7%. L3 cache misses did increase, but only by a small amount on the core processing the dominant connection. This is because the state for the dominant connection easily fits in the smaller L3 cache and it is accessed enough to keep it in cache. As for DH, it experienced a higher percentage decrease. This is due to an increase in cache misses

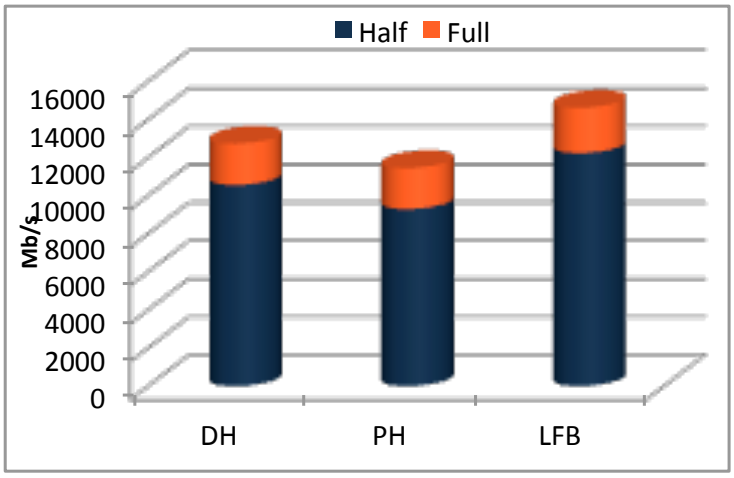

Figure 37: Many Capture with half L3 Cache.

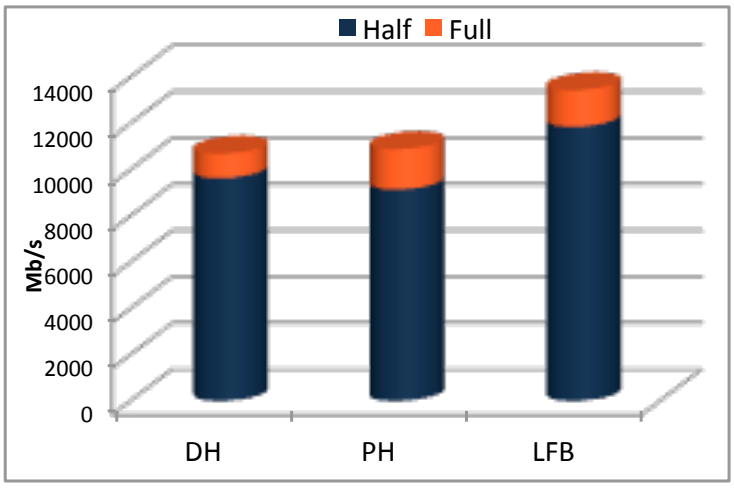

Figure 38: Balanced Capture with half L3 Cache.

on other connections that were being processed on the same core as the dominant connection.

The results for the Many capture show a similar percentage throughput decrease for all three packet scheduling algorithms. This is because of the large number of connections (i.e. many connections) that are processed concurrently resulting in the need of a larger L3 to cache all of the state. In the case of the Balanced capture, DH had the lowest decrease in throughput. This is the result of the uneven distribution of work across the cores. The core assigned the most work by DH did not increase its number of L3 cache misses by the same proportion as the other cores. Thus, even though its increase in L3 cache misses is slightly higher than LFB, it did not have as much of an impact on its performance.

6.4.6 Discussion

The LFB algorithm is the best performer in terms of throughput. This is because of the temporal locality cache benefit gained by preferring to process packets that map to the FBQ of the last packet over packets that may have arrived earlier. But, reordering packets for optimal temporal locality can have a negative impact on latency. In fact, we observed this effect on the Balanced capture. The DH algorithm also benefits from temporal locality because packets on the same flow bundle are always processed by the same thread. In addition, the DH algorithm does not reorder packets so a high maximum latency is less of an issue. However, interleaving packets on different flows and load imbalances reduce its maximum throughput. The PH algorithm produces the most balanced load in terms of packets processed by threads; yet, its throughput is typically lower than the other algorithms. This is because it does not exploit the cache benefits of temporal locality in the network traffic.

6.5 CONCLUSION

In this chapter, we discussed the design and implementation of two packet scheduling algorithms we invented. Each one was designed to maximize a different attribute of our ideal scheduler. We compared our two packet schedulers against DH, an algorithm commonly used to schedule packets in network applications. The results show the importance of cache affinity in packet scheduling. In fact, LFB, our packet scheduler that maximizes cache affinity, outperformed the other two schedulers in terms of throughput for all network captures. For the Balanced network capture, LFBs throughput was 38% faster than the next best scheduler. Our results show that scheduling packets for cache affinity is more important for throughput than balancing the workload evenly. All in all, by utilizing multithreaded DPI and scheduling packets based on cache affinity, we can protect large enterprise networks from malware downloads and infections with WebWitness, WebSentry and ExecScent.

CHAPTER VII

CONCLUSION

7.1 Summary of Contributions

The research presented in the previous chapters confirms our thesis statement that analyzing and modeling the network behavior of malware using DPI improves our understanding of malware downloads and infections. Furthermore, we demonstrated how to leverage this knowledge to create more effective network defenses by developing WebWitness, WebSentry and ExecScent. Specifically we examined the web browsing activities before a malicious download to study both drive-by and social engineering attacks. Then using insights gained from these studies we developed defenses for both. In addition, we examined the C&C communication of malware to engineer a set of features that we used to learn a malware's families protocol, adapt to a deployment network and identify infected hosts based on their communication. Lastly, we showed how these systems can be scaled to large enterprise networks by optimizing multithreaded DPI using cache affinity packet scheduling. Our malware studies and defense systems were the result of addressing the following eight research problems:

- Identify the sequence of web pages visited by a user that led to a malicious download with only network visibility; i.e., reconstruct the download path from observed HTTP transactions.
- Determine the reason for a malicious download by using features that can be extracted from the download path.
- Leverage the download paths to better understand current attack trends and develop more effective defenses.

- Learn the structure of malware communication from packet captures of malware executed in a sandbox or from an infected host.
- Detect infected hosts and new C&C domains by comparing the observed protocol structure to learned malicious communication.
- Adapt to the deployment network's communication profile to provide a better tradeoff between true and false positive.
- Determine if maximizing workload balance or cache affinity provides the best performance.
- Identify the packet scheduling performance tradeoffs in regards to throughput and latency.

To provide context to attacks we propose WebWitness, a malicious download investigation system. It automatically identifies the web path taken by the user to download a malicious executable. Also, it labels important nodes on path such as the landing, exploit and infector web pages as well as classifies the cause of the download as drive-by, social engineering or update. Using these paths we study attacks that result in malware downloads and propose a new more effective defense against drive-bys. We show that on average WebWitness can decrease the infection rate of drive-by downloads based on malicious content injection by almost 6 times compared to existing URL blacklisting approaches.

Leveraging WebWitness, we perform a comprehensive study of SE malicious downloads. We collect hundreds of in-the-wild web based SE malware attacks and analyze them in detail. From our analysis we develop a categorization system that classifies an SE download based on how the attacker gains the user's attention and the deception/persuasion techniques employed to trick the user. We find that over 80% of SE downloads use advertisements to get the user's attention and that most are

served from "low tier" ad networks. Furthermore, we discover a number of features that can be used to identify SE downloads and implement a detection system called WebSentry.

To discover infected hosts and new C&C domains, we develop ExecScent a system that learns the protocol structure of malware families. ExecScent creates adaptive control protocol templates (CPTs) that are based on examples of known C&C communications and the background traffic of the network where the templates are to be deployed. Learning from both provides a better trade-off of true and false positive for a given network environment. Our results show that ExecScent outperforms blacklisting (even large commercial up-to-date blacklist) by identifying new C&C domains and infected hosts even when the attackers change domains and hosting infrastructure.

Scheduling packets for multithreaded DPI to maximize throughput is challenging and has tradeoffs. We designed and implemented two packet scheduling algorithms to compare maximizing workload balance and cache affinity. Also, we compared both algorithms to direct hash an industry standard algorithm. Our results show that scheduling for cache affinity maximizes DPI throughput across all of the traffic mixes we tested. In fact, for some traffic mixes it outperformed the other schedulers by almost 40%. Scheduling using only cache affinity maximizes system throughput, but can starve some flows resulting in their packets having high latency. Thus for inline DPI, a hybrid approach is recommended to prevent flow starvation where cache affinity scheduling is primary and workload balance scheduling is secondary.

7.2 Discussion and Limitations

The defense systems described in this thesis focused exclusively on malware infections and communications over the HTTP protocol. While it is true that attackers use other protocols (IRC) and mediums (USB sticks), HTTP remains at the top in popularity for both infecting hosts [8] and C&C communications [13]. The browser (HTTP)

is a common infection vector because it is widely used to communicate with other hosts, it provides the resources required for social engineering and vulnerabilities are plentiful. As for C&C communication, HTTP is popular because it is allowed out of most networks, even those with strict egress filtering, and it blends into the background traffic (users browsing the web). We targeted our systems at HTTP because of its popularity, but many of the ideas (e.g., learning the structure of malware communication) can be directly applied to other protocols.

One may think that attackers could avoid our defense systems by simply performing their malicious activities over encrypted web traffic, using HTTPS. However, it is worth noting that in sensitive networks (e.g., enterprise and government networks) it is now common practice to deploy SSL Man-In-The-Middle (MITM) proxies, which allow for inspecting and recording the content of both HTTP and HTTPS traffic (perhaps excluding the traffic towards some whitelisted sides, such as banking applications, etc.). Therefore, our systems could simply work alongside such SSL MITM proxies.

Gaining the necessary visibility for DPI on some networks can be challenging. For instance, in a proxy environment, our systems need to be deployed between the internal network and proxy instead of at egress. Otherwise, they would be unable to associate HTTP transactions with hosts. Load balanced network links can also be an issue when multiple HTTP transactions from the same host need to be correlated (WebWitness) or when there is asymmetric routing. Also there are networks where the bandwidth is too high to inspect all egress traffic (e.g., ISPs). On these networks, we must rely on triggers from DNS (suspicious domain lookup) and netflow (suspicious behavior) to partition the network traffic and only direct network communication of suspicious hosts through DPI. While these systems are faced with these deployment challenges, these are problems that have solutions. They just require engineering effort.

7.3 Future Work

The ideas and systems developed in thesis can be applied and extended to new research. This sections describes severals avenues of research that build on the work presented in this thesis to study and solve interesting problems.

Phishing Study. Phishing continues to be a major threat to users despite over a decade of defense development by the security community. We propose a phishing study that explores how people reach phishing websites and examines the phishing tactics that are most successful. WebWitness can be used to capture the web paths followed by users to phishing websites by replacing its executable trigger with that of a known phishing domain list. Also, we can use WebWitness to follow the next n minutes of traffic from the user after they land on the phishing website to determine if the attack is successful.

Protocol Clustering on Live Networks. The structure of a protocol defines how information is exchanged between a client and server, or between peers. By protocol we are not referring to TCP, UDP, HTTP; but to the way data is structured on top of these transport and application layer protocols for information exchange. As discussed in Chapter 5, related malware can be clustered and identified by their protocol structure. By grouping related protocols on a live network, clients and servers using the same protocol structure can be identified. Protocols that are rare in terms of clients or new to the network may be suspicious as well as protocols that move domains. Furthermore, grouping network communication by protocol structure would aid network security investigators by allowing them to examine groups of related suspicious traffic instead of single communications in isolation.

Infector to C&C Domain. The vast majority of infector domains that host exploit kits are only active for a single day as discussed in Chapter 3. This is because there are many detection tools that identify exploits and malware downloads. When discovered

the infector domains and URLs are quickly blacklisted thus the attackers change them frequently. When observed from passive DNS (pDNS), the infector domains have a very unique signature. On the day prior to hosting the exploit kit the domain has zero to a small number of clients that resolve the domain. On the day it hosts the exploit kit there are often hundreds or even thousands of clients that resolve it. Then in the following days the number of clients resolving the domain on any given day is very few (often one or two).

This unique signature of infector domains makes them relatively easy to identify in pDNS. Since they are no longer in use, they themselves are not that interesting. However, by following the clients and the domains they resolve after visiting an infector domain often leads to the C&C domain. The idea is that clients that get infected with the same malware from the same infector will visit the same C&C domain. Observing overlapping unpopular domains in a subset of these clients in the minutes or hours following their visit to the same infector domain may provide a candidate list of C&C domains. Also, clients can be followed in the other direction (domains prior to infection) to find compromised websites directing users to exploit kits.

Automated Social Engineering Training. The most popular social engineering attacks are detected and labeled by WebSentry. In most cases, the trick page is also easily identified. Since the content of the trick page is captured by WebSentry, it can be used to recreate the social engineering attack without the potential harm of downloading a malicious executable. The latest attacks can be simulated and periodically shown to users in their browser as they surf the web as part of an organization's ongoing security training. If they fall for the social engineering tactic, they can be informed of their mistake and directed to training materials for recognizing social engineering attacks.

Network Directed Host Analysis. Detecting malware at the network level has

several advantages over host detection. For instance, network monitoring has no impact on the performance of the host and cannot be disabled by malware. However, the host context and process information at the network level is very limited. Having host level information would improve detection. It could be obtained by the network directing a host agent to monitor the process generating the suspicious communication. Thus, the analysis overhead on the host would be limited to a single process and only run when a host is identified as suspicious by the network. This additional context gained from the host agent would aid in determining if the host is compromised.

7.4 Concluding Remarks

This thesis explored analyzing and modeling the network behavior of malware using DPI to improve our understanding of malware downloads and infections. We used insights gained from this process to improve detection, annotation and network defenses. There are four important contributions. First is WebWitness, a system that provides context to malware downloads by identifying the web path, annotating nodes of interest and determining the cause. Next we performed a systematic study of SE downloads on the web. From this study, we created a categorization system for classifying attacks and developed a defense that is effective against the most common SE attacks. Third, we designed and implemented ExecScent a novel system that adapts its detection to identify infected hosts and new C&C domains. Lastly we explored packet scheduling for DPI and discovered that scheduling packets for cache affinity maximizes throughput allowing DPI to be deployed on networks with speeds exceeding 10 Gb/s. All of the systems in this paper have been or are in the process of being productized by industry.

REFERENCES

[1] "A brief history of malware obfuscation." http://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2.

[2] "Capture-hpc client honeypot." https://projects.honeynet.org/capture-hpc.

[3] "Cryptolocker ransomware." http://www.secureworks.com/cyber-threat-intelligence/threats/cryptolocker-ransomware/.

[4] "Cybercrime 2015: An inside look at the changing threat landscape." https://www.emc.com/collateral/white-paper/rsa-white-paper-cybercrime-trends-2015.pdf.

[5] "A forecast of the cyberthreat landscape in 2015." http://www.scmagazine.com/a-forecast-of-the-cyberthreat-landscape-in-2015/article/388921/.

[6] "Google safe browsing api." https://developers.google.com/safe-browsing/.

[7] "Honeyc." https://projects.honeynet.org/honeyc.

[8] "How malware bypasses our most advanced security measures." http://www.darkreading.com/perimeter/how-malware-bypasses-our-most-advanced-security-measures/a/d-id/1318974.

[9] "New carberp variant heads downunder." http://www.symantec.com/connect/blogs/new-carberp-variant-heads-down-under.

[10] "The real story of stuxnet." http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet.

[11] "Receive Side Scaling (RSS)." http://www.microsoft.com/whdc/device/network/ndis_rss.mspx/.

[12] "RFC 3393." http://tools.ietf.org/html/rfc3393.

[13] "Sandboxing and signature strategies are simply not able to keep up with constantly morphing malware." http://www.darkreading.com/vulnerabilities---threats/new-research-from-damballa-80--of-malware-still-favors-http/d/d-id/1140445?print=yes.

[14] "Snort 3.0." http://www.snort.org/snort-downloads/snort-3-0/.

[15] "What is flame?." http://www.kaspersky.com/flame.

[16] "The zeroaccess rootkit." https://nakedsecurity.sophos.com/zeroaccess2/.

[17] "IEEE Std 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks Link Aggregation," 2008. http://standards.ieee.org/getieee802/download/802.1AX-2008.pdf.

[18] "NSS Labs. IBM ISS GX6116 Intrusion Prevention System achieves NSS labs gold award and certification," 2008. http://nsslabs.com/2008/ibm-iss-gx6116-intrusion-prevention-system-achieves-nss-labs-gold-award-and-
html.

[19] "Proventia Network Intrusion Prevention System," 2008. http://www-935.ibm.com/services/us/index.wss/offerfamily/iss/a1030570/.

[20] "Application Layer Packet Classifier for Linux (L7-filter)," 2009. http://l7-filter.sourceforge.net.

[21] "Intel Xeon Processor E5540," 2009. http://ark.intel.com/Product.aspx?id=37104.

[22] "The download.com debacle: What CNET needs to do to make it right," 2011. https://www.eff.org/deeplinks/2011/12/downloadcom-debacle-what-cnet-needs-do-make-it-right.

[23] "Huge decline in fake av following credit card processing shakeup," 2011. http://krebsonsecurity.com/2011/08/huge-decline-in-fake-av-following-credit-card-processing-shakeup/.

[24] "Fake virus alert malware (fakeav) information and what to do," 2013. http://helpdesk.princeton.edu/kb/display.plx?ID=1080.

[25] "Heres what happens when you install the top 10 download.com apps," 2015. http://www.howtogeek.com/198622/heres-what-happens-when-you-install-the-top-10-download.com-apps/?PageSpeed=noscript.

[26] Abraham, S. and Chengalur-Smith, I., "An overview of social engineering malware: Trends, tactics, and implications," *Technology in Society*, vol. 32, no. 3, pp. 183 – 196, 2010.

[27] Antonakakis, M., Demar, J., Stevens, K., and Dagon, D., "Unveiling the network criminal infrastructure of tdss/tdl4." https://www.damballa.com/downloads/r_pubs/Damballa_tdss_tdl4_case_study_public.pdf.

[28] Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., and Feamster, N., "Building a dynamic reputation system for DNS," in *the Proceedings of 19th USENIX Security Symposium (USENIX Security '10)*, 2010.

[29] Antonakakis, M., Perdisci, R., Lee, W., Dagon, D., and Vasiloglou, N., "Detecting Malware Domains at the Upper DNS Hierarchy," in *the Proceedings of 20th USENIX Security Symposium (USENIX Security '11)*, 2011.

[30] Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., and Dagon, D., "From throw-away traffic to bots: detecting the rise of dga-based malware," in *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, (Berkeley, CA, USA), pp. 24–24, USENIX Association, 2012.

[31] Bakhshi, T., Papadaki, M., and Furnell, S., "A practical assessment of social engineering vulnerabilities," in *2nd International Symposium on Human Aspects of Information Security & Assurance (HAISA 2008)*, pp. 12–23, 2008.

[32] Bennett, J., Partridge, C., and Shectman, N., "Packet reordering is not pathological network behavior," *Networking, IEEE/ACM Transactions on*, vol. 7, pp. 789–798, Dec 1999.

[33] Berners-Lee, T., Fielding, R., and Masinter, L., "RFC3986 - Uniform Resource Identifier (URI): Generic Syntax," 2005.

[34] Bilge, L., Kirda, E., Kruegel, C., and Balduzzi, M., "Exposure: Finding malicious domains using passive dns analysis," in *NDSS*, 2011.

[35] Blanton, E. and Allman, M., "On making tcp more robust to packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 32, pp. 20–30, Jan. 2002.

[36] Bott, E., "Social engineering in action: how web ads can lead to malware," 2011.

[37] Breiman, L., "Random forests," *Mach. Learn.*, vol. 45, Oct. 2001.

[38] Caballero, J., Grier, C., Kreibich, C., and Paxson, V., "Measuring pay-per-install: the commoditization of malware distribution," in *Proceedings of the 20th USENIX conference on Security*, SEC'11, (Berkeley, CA, USA), pp. 13–13, USENIX Association, 2011.

[39] Canali, D., Cova, M., Vigna, G., and Kruegel, C., "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, (New York, NY, USA), ACM, 2011.

[40] Cao, Z., Wang, Z., and Zegura, E., "Performance of hashing-based schemes for internet load balancing," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 332–341 vol.1, 2000.

[41] Chang, C.-C. and Lin, C.-J., "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[42] Chen, K. Z., Gu, G., Zhuge, J., Nazario, J., and Han, X., "Webpatrol: Automated collection and replay of web-based malware scenarios," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, (New York, NY, USA), ACM, 2011.

[43] Cialdini, R. B. Pearson Education, 5th ed., 2000.

[44] Cova, M., Kruegel, C., and Vigna, G., "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), ACM, 2010.

[45] Cristianini, N. and Shawe-Taylor, J., *An introduction to support Vector Machines: and other kernel-based learning methods*. New York, NY, USA: Cambridge University Press, 2000.

[46] Curtsinger, C., Livshits, B., Zorn, B., and Seifert, C., "Zozzle: Fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX Conference on Security*, SEC'11, (Berkeley, CA, USA), USENIX Association, 2011.

[47] Danchev, D., "Leaked DIY malware generating tool spotted in the wild," 2013. `http://blog.webroot.com/2013/01/18/leaked-diy-malware-generating-tool-spotted-in-the-wild/`.

[48] de la Higuera, C. and Casacuberta, F., "Topology of strings: Median string is np-complete," *Theoretical computer science*, vol. 230, no. 1, pp. 39–48, 2000.

[49] Dietrich, C. J., Rossow, C., and Pohlmann, N., "Exploiting visual appearance to cluster and detect rogue software," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, (New York, NY, USA), pp. 1776–1783, ACM, 2013.

[50] Dinaburg, A., Royal, P., Sharif, M., and Lee, W., "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, (New York, NY, USA), pp. 51–62, ACM, 2008.

[51] Dittmann, G. and Herkersdorf, A., "Network processor load balancing for high-speed links," 2002.

[52] Duman, S., Onarlioglu, K., Ulusoy, A. O., Robertson, W., and Kirda, E., "Trueclick: Automatically distinguishing trick banners from genuine download links," in *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, (New York, NY, USA), pp. 456–465, ACM, 2014.

[53] Edmonds, R., "ISC Passive DNS Architecture," 2012. `https://kb.isc.org/getAttach/30/AA-00654/passive-dns-architecture.pdf`.

[54] Egele, M., Scholte, T., Kirda, E., and Kruegel, C., "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, pp. 6:1–6:42, Mar. 2008.

[55] Fisher, D., "Zeus source code leaked." `http://threatpost.com/en_us/blogs/zeus-source-code-leaked-051011`.

[56] Franke, H., Nelms, T., Yu, H., Achilles, H. D., and Salz, R., "Exploiting heterogeneous multicore-processor systems for high-performance network processing," *IBM J. Res. Dev.*, vol. 54, pp. 13–26, Jan. 2010.

[57] Franke, H., Xenidis, J., Basso, C., Bass, B. M., Woodward, S. S., Brown, J. D., and Johnson, C. L., "Introduction to the wire-speed processor and architecture," *IBM J. Res. Dev.*, vol. 54, pp. 27–37, Jan. 2010.

[58] Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C. J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A., Provos, N., Rafique, M. Z., Rajab, M. A., Rossow, C., Thomas, K., Paxson, V., Savage, S., and Voelker, G. M., "Manufacturing compromise: The emergence of exploit-as-a-service," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, (New York, NY, USA), pp. 821–832, ACM, 2012.

[59] Gu, G., Perdisci, R., Zhang, J., and Lee, W., "Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th conference on Security symposium*, SS'08, (Berkeley, CA, USA), pp. 139–154, USENIX Association, 2008.

[60] Gu, G., Zhang, J., and Lee, W., "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[61] Guo, D., Liao, G., Bhuyan, L. N., and Liu, B., "An adaptive hash-based multilayer scheduler for l7-filter on a highly threaded hierarchical multi-core server," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '09, (New York, NY, USA), pp. 50–59, ACM, 2009.

[62] Guo, D., Liao, G., Bhuyan, L. N., Liu, B., and Ding, J. J., "A scalable multithreaded l7-filter design for multi-core servers," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, (New York, NY, USA), pp. 60–68, ACM, 2008.

[63] Guo, F., Ferrie, P., and Chiueh, T.-C., "A study of the packer problem and its solutions," in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, RAID '08, (Berlin, Heidelberg), pp. 98–115, Springer-Verlag, 2008.

[64] Invernizzi, L., Benvenuti, S., Cova, M., Comparetti, P. M., Kruegel, C., and Vigna, G., "Evilseed: A guided approach to finding malicious web pages," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, IEEE Computer Society, 2012.

[65] Invernizzi, L., Lee, S.-J., Miskovic, S., Mellia, M., Torres, R., Kruegel, C., Saha, S., and Vigna, G., "Nazca: Detecting malware distribution in large-scale networks," 2014.

[66] Ivaturi, K. and Janczewski, L., "A taxonomy for social engineering attacks," in *International Conference on Information Resources Management*, 2011.

[67] Jacob, G., Hund, R., Kruegel, C., and Holz, T., "Jackstraws: picking command and control connections from bot traffic," in *Proceedings of the 20th USENIX conference on Security*, SEC'11, (Berkeley, CA, USA), pp. 29–29, USENIX Association, 2011.

[68] Jain, R. and Routhier, S., "Packet trains–measurements and a new model for computer network traffic," *Selected Areas in Communications, IEEE Journal on*, vol. 4, pp. 986–995, Sep 1986.

[69] Jang, J., Brumley, D., and Venkataraman, S., "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 309–320, ACM, 2011.

[70] Jones, M., "Protecting privacy with referrers," 2010. `https://www.facebook.com/notes/facebook-engineering/protecting-privacy-with-referrers/392382738919`.

[71] Kandula, S., Katabi, D., Sinha, S., and Berger, A., "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 51–62, Mar. 2007.

[72] Kapravelos, A., Shoshitaishvili, Y., Cova, M., Kruegel, C., and Vigna, G., "Revolver: An automated approach to the detection of evasive web-based malware," in *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, (Berkeley, CA, USA), USENIX Association, 2013.

[73] Kim, D. W., Yan, P., and Zhang, J., "Detecting fake anti-virus software distribution webpages," *Comput. Secur.*, vol. 49, pp. 95–106, Mar. 2015.

[74] Kim, H.-A. and Karp, B., "Autograph: toward automated, distributed worm signature detection," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2004.

[75] Kolter, J. Z. and Maloof, M. A., "Learning to detect and classify malicious executables in the wild," *J. Mach. Learn. Res.*, vol. 7, pp. 2721–2744, Dec. 2006.

[76] Krombholz, K., Hobel, H., Huber, M., and Weippl, E., "Advanced social engineering attacks," *J. Inf. Secur. Appl.*, vol. 22, pp. 113–122, June 2015.

[77] Laribee, L., "Development of methodical social engineering taxonomy project," 2006. MSc, Naval Postgraduate School, Monterey, California.

[78] Lee, S. and Kim, J., "Warningbird: A near real-time detection system for suspicious urls in twitter stream," *IEEE Trans. Dependable Secur. Comput.*, vol. 10, May 2013.

[79] Li, C., Peng, G., Gopalan, K., and cker Chiueh, T., "Performance guarantee for cluster-based internet services," in *Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on*, pp. 327–332, Dec 2002.

[80] Li, Z., Alrwais, S., Xie, Y., Yu, F., and Wang, X., "Finding the linchpins of the dark web: A study on topologically dedicated hosts on malicious web infrastructures," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, 2013.

[81] Li, Z., Zhang, K., Xie, Y., Yu, F., and Wang, X., "Knowing your enemy: Understanding and detecting malicious web advertising," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, (New York, NY, USA), ACM, 2012.

[82] Lu, L., Perdisci, R., and Lee, W., "Surf: Detecting and measuring search poisoning," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, (New York, NY, USA), ACM, 2011.

[83] Lu, L., Yegneswaran, V., Porras, P., and Lee, W., "Blade: An attack-agnostic approach for preventing drive-by malware infections," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, (New York, NY, USA), ACM, 2010.

[84] Martin, R., Menth, M., and Hemmkeppler, M., "Accuracy and dynamics of hash-based load balancing algorithms for multipath internet routing," in *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pp. 1–10, Oct 2006.

[85] Mavrommatis, P., Ballard, L., Provos, N., Inc, G., and Zhao, X., "The nocebo effect on the web: An analysis of fake anti-virus distribution," in *In USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET*, 2010.

[86] Mekky, H., Torres, R., Zhang, Z.-L., Saha, S., and Nucci, A., "Detecting malicious http redirections using trees of user browsing activity," in *INFOCOM, 2014 Proceedings IEEE*, 2014.

[87] min Wang, Y., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., and King, S., "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," in *In NDSS*, 2006.

[88] Mitnick, K. D. and Simon, W. L., *The Art of Deception: Controlling the Human Element of Security*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 2002.

[89] Moshchuk, E., Bragin, T., Gribble, S. D., and Levy, H. M., "A crawler-based study of spyware on the web," 2006.

[90] Mouton, F., Malan, M., Leenen, L., and Venter, H., "Social engineering attack framework," in *Information Security for South Africa (ISSA), 2014*, pp. 1–9, Aug 2014.

[91] Mouton, F., Leenen, L., Malan, M., and Venter, H., "Towards an ontological model defining the social engineering domain," in *ICT and Society* (Kimppa, K., Whitehouse, D., Kuusela, T., and Phahlamohlaka, J., eds.), vol. 431 of *IFIP Advances in Information and Communication Technology*, pp. 266–279, Springer Berlin Heidelberg, 2014.

[92] Nappa, A., Rafique, M. Z., and Caballero, J., "Driving in the cloud: An analysis of drive-by download operations and abuse reporting," in *Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'13, (Berlin, Heidelberg), Springer-Verlag, 2013.

[93] Nazario, J., "Phoneyc: A virtual client honeypot," in *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'09, (Berkeley, CA, USA), USENIX Association, 2009.

[94] Neasbitt, C., Perdisci, R., Li, K., and Nelms, T., "Clickminer: Towards forensic reconstruction of user-browser interactions from network traces," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer & Communications Security*, CCS '14, (New York, NY, USA), ACM, 2014.

[95] Nelms, T. and Ahamad, M., "Packet scheduling for deep packet inspection on multi-core architectures," in *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, pp. 1–11, Oct 2010.

[96] Nelms, T., Perdisci, R., and Ahamad, M., "Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates," in *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, (Berkeley, CA, USA), USENIX Association, 2013.

[97] Nelms, T., Perdisci, R., Antonakakis, M., and Ahamad, M., "Webwitness: Investigating, categorizing, and mitigating malware download paths," in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, (Berkeley, CA, USA), pp. 1025–1040, USENIX Association, 2015.

[98] Newsome, J., Karp, B., and Song, D., "Polygraph: Automatically generating signatures for polymorphic worms," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, (Washington, DC, USA), pp. 226–241, IEEE Computer Society, 2005.

[99] Oberheide, J., Cooke, E., and Jahanian, F., "Cloudav: N-version antivirus in the network cloud," in *Proceedings of the 17th Conference on Security Symposium*, SS'08, (Berkeley, CA, USA), pp. 91–106, USENIX Association, 2008.

[100] Paxson, V., Sommer, R., and Weaver, N., "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," in *Sarnoff Symposium, 2007 IEEE*, pp. 1–7, April 2007.

[101] Paxson, V., "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, pp. 2435–2463, Dec. 1999.

[102] Perdisci, R., Ariu, D., and Giacinto, G., "Scalable fine-grained behavioral clustering of http-based malware," *Computer Networks*, vol. 57, no. 2, pp. 487 – 500, 2013. Botnet Activity: Analysis, Detection and Shutdown.

[103] Perdisci, R., Lanzi, A., and Lee, W., "Classification of packed executables for accurate computer virus detection," *Pattern Recogn. Lett.*

[104] Perdisci, R., Lee, W., and Feamster, N., "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, (Berkeley, CA, USA), pp. 26–26, USENIX Association, 2010.

[105] Platt, J., "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[106] Power, R. and Forte, D., "Social engineering: attacks have evolved, but countermeasures have not," *Computer Fraud and Security*, vol. 2006, no. 10, pp. 17 – 20, 2006.

[107] Provos, N., Mavrommatis, P., Rajab, M. A., and Monrose, F., "All your iframes point to us," in *Proceedings of the 17th Conference on Security Symposium*, SS'08, (Berkeley, CA, USA), USENIX Association, 2008.

[108] Provos, N., McNamee, D., Mavrommatis, P., Wang, K., and Modadugu, N., "The ghost in the browser analysis of web-based malware," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, 2007.

[109] Rafique, M. Z. and Caballero, J., "Firma: Malware clustering and network signature generation with mixed network behaviors," in *Proceedings of the 16th international conference on Research in Attacks, Intrusions, and Defenses*, RAID'13, Springer-Verlag, 2013. To be published. Research conducted concurrently and independently of ExecScent.

[110] Rajab, M. A., Ballard, L., Lutz, N., Mavrommatis, P., and Provos, N., "Camp: Content-agnostic malware protection," in *Proceedings of Annual Network and Distributed System Security Symposium, NDSS (February 2013)*, Citeseer, 2013.

[111] Rajab, M. A., Ballard, L., Mavrommatis, P., Provos, N., and Zhao, X., "The nocebo effect on the web: An analysis of fake anti-virus distribution," in *3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'10, (Berkeley, CA, USA), USENIX Association, 2010.

[112] Ratanaworabhan, P., Livshits, B., and Zorn, B., "Nozzle: A defense against heap-spraying code injection attacks," in *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, (Berkeley, CA, USA), USENIX Association, 2009.

[113] Rieck, K., Krueger, T., and Dewald, A., "Cujo: Efficient detection and prevention of drive-by-download attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, (New York, NY, USA), ACM, 2010.

[114] Roesch, M., "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, 1999.

[115] Santorelli, S., "Developing botnets - an analysis of recent activity," 2010. `http://www.team-cymru.com/ReadingRoom/Whitepapers/2010/developing-botnets.pdf`.

[116] Shi, W. and Kencl, L., "Sequence-preserving adaptive load balancers," in *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, ANCS '06, (New York, NY, USA), pp. 143–152, ACM, 2006.

[117] Shi, W., MacGregor, M. H., and Gburzynski, P., "Load balancing for parallel forwarding," *IEEE/ACM Trans. Netw.*, vol. 13, pp. 790–801, Aug. 2005.

[118] Shi, W., MacGregor, M. H., and Gburzynski, P., "A scalable load balancer for forwarding internet traffic: Exploiting flow-level burstiness," in *Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems*, ANCS '05, (New York, NY, USA), pp. 145–152, ACM, 2005.

[119] Singh, S., Estan, C., Varghese, G., and Savage, S., "Automated worm fingerprinting," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2004.

[120] Squillante, M. and Lazowska, E., "Using processor-cache affinity information in shared-memory multiprocessor scheduling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, pp. 131–143, Feb 1993.

[121] Stokes, J. W., Andersen, R., Seifert, C., and Chellapilla, K., "Webcop: Locating neighborhoods of malware on the web," in *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'10, USENIX Association, 2010.

[122] Stone-Gross, B., "Pushdo downloader variant generating fake HTTP requests," 2012. `http://www.secureworks.com/cyber-threat-intelligence/threats/Pushdo_Downloader_Variant_Generating_Fake_HTTP_Requests/`.

[123] Stone-gross, B., Abman, R., Kemmerer, R. A., Kruegel, C., Steigerwald, D. G., and Vigna, G., "The underground economy of fake antivirus software," in *In Proc. (online) WEIS 2011*, 2011.

[124] Stringhini, G., Kruegel, C., and Vigna, G., "Shady paths: Leveraging surfing crowds to detect malicious web pages," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, ACM, 2013.

[125] Townsend, K., "R&d: The art of social engineering," *Infosecurity*, vol. 7, no. 4, pp. 32–35, 2010.

[126] Vadrevu, P., Rahbarinia, B., Perdisci, R., Li, K., and Antonakakis, M., "Measuring and detecting malware downloads in live network traffic," in *Computer Security ESORICS 2013* (Crampton, J., Jajodia, S., and Mayes, K., eds.), vol. 8134 of *Lecture Notes in Computer Science*, pp. 556–573, Springer Berlin Heidelberg, 2013.

[127] Whaley, B., "Toward a general theory of deception," 1982. Miliary Deception and Strategic Surprise.

[128] Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., and Kirda, E., "Automatically generating models for botnet detection," in *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, (Berlin, Heidelberg), pp. 232–249, Springer-Verlag, 2009.

[129] Xie, G., Iliofotou, M., Karagiannis, T., Faloutsos, M., and Jin, Y., "Resurf: Reconstructing web-surfing activity from network traffic," in *IFIP Networking Conference, 2013*, pp. 1–9, May 2013.

[130] Xie, Y., Yu, F., Achan, K., Panigrahy, R., Hulten, G., and Osipkov, I., "Spamming botnets: signatures and characteristics," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, (New York, NY, USA), pp. 171–182, ACM, 2008.

[131] Yu, H., Franke, H., Biran, G., Golander, A., Nelms, T., and Bass, B. M., "Stateful hardware decompression in networking environment," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, (New York, NY, USA), pp. 141–150, ACM, 2008.

[132] Zhang, J., Seifert, C., Stokes, J. W., and Lee, W., "Arrow: Generating signatures to detect drive-by downloads," in *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, (New York, NY, USA), ACM, 2011.

[133] Zhou, Y. and Jiang, X., "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 95–109, IEEE, 2012.

[134] Zipf, G. K., *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949.