UNDERSTANDING AND DETECTING MALICIOUS CYBER INFRASTRUCTURES

A Dissertation

by

JIALONG ZHANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2016

Major Subject: Computer Engineering

ABSTRACT

Malware (e.g., trojans, bots, and spyware) is still a pervasive threat on the Internet. It is able to infect computer systems to further launch a variety of malicious activities such as sending spam, stealing sensitive information and launching distributed denial-of-service (DDoS) attacks. In order to continue malevolent activities without being detected and to improve the efficiency of malicious activities, cyber-criminals tend to build malicious cyber infrastructures to communicate with their malware and to exploit benign users. In these infrastructures, multiple servers are set to be efficient and anonymous in (i) malware distribution (using redirectors and exploit servers), (ii) control (using C&C servers), (iii) monetization (using payment servers), and (iv) robustness against server takedowns (using multiple backups for each type of server).

The most straightforward way to counteract the malware threat is to detect malware directly on infected hosts. However, it is difficult since packing and obfuscation techniques are frequently used by malware to evade state-of-the-art anti-virus tools. Therefore, an alternate solution is to detect and disrupt the malicious cyber infrastructures used by malware. In this dissertation, we take an important step in this direction and focus on identifying malicious servers behind those malicious cyber infrastructures. We present a comprehensive inferring framework to infer servers involved in malicious cyber infrastructure based on the three roles of those servers: compromised server, malicious server accessed through redirection and malicious server accessed through directly connecting. We characterize these three roles from four novel perspectives and demonstrate our detection technologies in four systems: PoisonAmplifier [113], SMASH [112], VisHunter [111] and NeighbourWatcher [110]. PoisonAmplifier focuses on compromised servers. It explores the fact that cybercriminals tend to use compromised servers to trick benign users during the attacking process. Therefore, it is designed to proactively find more compromised servers. SMASH focuses on malicious servers accessed through directly connecting. It explores the fact that multiple backups are usually used in malicious cyber infrastructures to avoid server takedowns. Therefore, it leverages the correlation among malicious servers to infer a group of malicious servers. VisHunter focuses on the redirections from compromised servers to malicious servers. It explores the fact that cybercriminals usually conceal their core malicious servers. Therefore, it is designed to detect those "invisible" malicious servers. NeighbourWatcher focuses on all general malicious servers promoted by spammers. It explores the observation that spammers intend to promote some servers (e.g., phishing servers) on the special websites (e.g., forum and wikis) to trick benign users and to improve the reputation of their malicious servers. In short, we build a comprehensive inferring framework to identify servers involved in malicious cyber infrastructures from four novel perspectives and implement different inference techniques in different systems that complement each other.

Our inferring framework has been evaluated in live networks and/or real-world network traces. The evaluation results show that it can accurately detect malicious servers involved in malicious cyber infrastructures with a very low false positive rate.

We found the three roles of malicious servers we proposed can characterize most of servers involved in malicious cyber infrastructures, and the four principles we developed for the detection are invariable across different malicious cyber infrastructures. We believe our experience and lessons are of great benefit to the future malicious cyber infrastructure study and detection.

ACKNOWLEDGEMENTS

First of all, I would like to express my thanks and appreciation to my advisor, Prof. Guofei Gu for his continuous guidance and support throughout my Ph.D. study. Without his inspiration, I can not complete this research. I am grateful as well to the rest of my dissertation committee members, Prof. Riccardo Bettati, Prof. James Caverlee and Prof. Narasimha Reddy for generously giving their time and expertise to better my work.

I have also been very fortunate to be a member of a great security research team in our lab at Texas A&M university. I wish to thank Chao Yang, Zhaoyan Xu, Seungwon Shin, Lei Xu, and Guangliang Yang for their valuable assistance on my research. It is my pleasure to work with them over the past few years. They are not only intriguing helpers for my research but also great friends in my life.

Furthermore, I would like to thank my collaborators, Xin Hu, Jiyong Jang, Ting Wang, Marc Ph. Stoecklin, Sabyasachi Saha, Sung-Ju Lee, Marco Mellia for mentoring me during my summer internship. They also provide useful data and valuable comments for my research.

I also gratefully acknowledge the funding sources that made my Ph.D. work possible. My work was supported by the National Science Foundation (NSF) under Grant no. CNS-1314823, CNS-1218929, and CNS-0954096, and the Air Force Office of Scientific Research (AFOSR) under Grant No. FA-9550-13-1-0077.

Last but not the least, I take this opportunity to thank my parents. Through the good times and bad times, they always stand behind me and devote their love. I cannot be more thankful to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Malware is still a pervasive threat to current Internet. CNN [33] recently reported that nearly 1 million new malware threats are released every day, and a variety of malware has been developed such as trojans, bots, and spyware. These malware controlled by cybercriminals is able to infect computer systems to further launch different malicious activities such as sending spam, stealing sensitive information and launching denial of services attacks [73] to either facilitate cyber crimes or gain illegal profits.

To control malware efficiently and anonymously, cybercriminals usually set up a variety of malicious servers, e.g., exploit servers, C&C servers, and phishing servers. Based on a recent threat report from Websense [52], malicious websites have increased by nearly 600% worldwide since 2012. Different malicious servers, such as redirectors, exploit servers, C&C servers, and payment servers, often join forces to leverage their diverse functionalities and to create more efficient and anonymous malicious cyber infrastructures for malware distribution, control, and monetization.

In this chapter, we first introduce the malicious cyber infrastructure and explain why it is important for malware detection. We also present an abstract model to characterize the current malicious cyber infrastructures. We then outline the research challenges for the malicious cyber infrastructure detection, and clarify the goals we want to achieve in our solution. Next, we provide an overview of our solution: an inference framework to detect servers involved in malicious cyber infrastructures from different perspectives. Finally, we present the contributions of this dissertation and the organization of remaining chapters.

1.1 Malicious Cyber Infrastructure

The most direct way to counteract malware is to detect it on the infected hosts. However, it becomes more and more difficult. Since malware continues to evolve and accelerate in frequency and complexity, current security measures are too slow to evolve in addressing malware risks[30]. In addition, packing and obfuscation techniques are frequently used by malware to evade signature-based detection. Typical techniques include substituting equivalent code instructions, changing their ordering, and packing (which encrypts or compresses the original malware program into random-looking data and decrypts the content when the packed malware is executed). What makes them successful is that malware authors have the resources (e.g., state-of-art security tools and time) to test version after version, and to incrementally enhance their tactics up to the point where malware can infect a system and go undetected. Recent research [70] shows that some malware authors will first upload their new variants of malware into public online anti-virus services before releasing that malware to the public. For example, the malware authors first upload their malware on VirusTotal website[51], where more than 40 commercial anti-virus software is used to scan the uploaded malware. If VirusTotal detects the uploaded malware, malware authors will further obfuscate their malware until it can successfully bypass all those anti-virus tools installed on VirusTotal. In this way, they can ensure that current state-of-the-art anti-virus tools can not detect them. Therefore, detecting malware directly on infected hosts is challenging.

Thus, to counteract malware, another alternate solution is to detect and disrupt malicious cyber infrastructures, which are used by cybercriminals to control malware and to launch malicious activities. Malicious cyber infrastructures essentially comprise multiple servers with diverse functionalities. In a malicious cyber infrastructure, multiple servers are usually set to be efficient and anonymous in malware distribution (using redirectors and exploit servers), control (using C&C servers) , monetization (using payment servers), and robustness against server takedowns (using multiple backups for each type of servers). Such combination with diverse malicious servers makes the malicious cyber infrastructure become more powerful and robustness. Thus, great efforts are needed to understand and mitigate malicious cyber infrastructures.

Usually, there are mainly two ways for a user to visit malicious infrastructures. A user may first connect to a compromised server (① in Figure 1.2), which will then redirect the user to an exploit server (Type A malicious server in Figure 1.2) through one or multiple redirections. After the user's machine got exploited and infected with malware, the user's machine may directly visit (② in Figure 1.2) other post-infection malicious servers (Type B malicious server in Figure 1.2) . Based on this observation, in this dissertation, we model the malicious cyber infrastructure based on how the servers under malicious infrastructures are accessed.

Figure 1.2 shows the model of our defined malicious cyber infrastructures. In this way, we category all malicious servers with diverse functionalities into three roles: compromised servers, Type A malicious server [1], and Type B malicious server [2]. Such model can represent most of the malicious cyber infrastructures, and these three roles can represent all kinds of servers with diverse functionalities. For example, exploit server is usually reached through one or multiple redirections from compromised servers, therefore, it belongs to Type A malicious server in our model. C&C server is usually accessed through directly connecting, it belongs to Type B malicious server in our model.

[1]Malicious servers are accessed through redirections.
[2]Malicious servers are accessed through directly visiting.

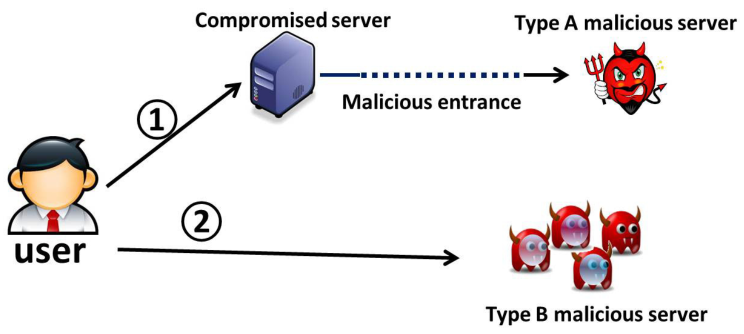


Figure 1.1: Malicious cyber infrastructure

1.2 Research Challenges and Our Goals

To counter malicious activities and disrupt the malicious cyber infrastructures, we first need to detect the servers with above three roles in malicious cyber infrastructures. There are several properties make such detection become more complicated.

- Malicious cyber infrastructures are very similar to benign web infrastructures. Both of them use HTTP protocol, which is always the majority traffic in the current network. In addition, malicious cyber infrastructures tend to utilize compromised servers as their entrances, which are actually benign servers and share similar features (e.g., benign web content and benign registration information) with other benign servers. Therefore, some domain name based detection systems may not be effective.

- Servers involved in malicious cyber infrastructures are dynamically evolving. For example, cybercriminals can easily add new malicious servers or compromised servers more quickly than a user updates his blacklists. Thus, static and signatures-based approaches may not be effective.

- Malicious activities are usually multi-phased processes, incorporating multiple servers with same or different functions. Therefore, each server only needs to involve in part of the whole malicious activities. Thus, looking at only one specific stage or perspective, as many existing solutions do, may not be effective.

- The number of infected clients involved in malicious activities are usually small in certain networks. Therefore, such malicious activities are usually stealthy and less frequently triggered. As a result, detection systems require multiple infections will be less effective.

Because of these challenges, existing techniques such as traditional anti-virus tools cannot sufficiently handle the malicious cyber infrastructures. In Chapter 2, we provide a detailed overview of various related work (e.g., compromised server detection, malicious server detection, malicious redirection detection and malicious cyber infrastructure detection) and further explain why these existing solutions are not adequate. In this dissertation, we propose our solution for malicious cyber infrastructure detection. When designing the solution, we have the following goals in mind:

- Our solution should be based on sound principles that can capture the fundamental invariants of malicious cyber infrastructures rather than symptoms (e.g., special template in HTTP traffic)

- Our solution should provide complementary techniques and cover multiple stages, dimensions, and perspectives.

- Our solution should be general. By general, we mean the solution could detect general malicious servers rather than a specific type of malicious servers.

- Our solution should be easily deployed with less requirement for the network traffic. By less requirement, we mean that our system should not require the large diverse user base or multiple infections in the network traffic.

- Our solution should provide practical prototype systems that can work in the real-world network. By practical, we mean that our system can accurately detect real-world malicious servers.

1.3 Solution Overview

We propose an inference framework to effectively detect servers behind the malicious cyber infrastructures. Within this framework, we study and detect malicious cyber infrastructures based on the three roles of servers and their correlations. We explore four novel principles to characterize these three roles and build four detection systems: PoisonAmplifier [113], SMASH [112], VisHunter [111], and NeighbourWatcher [110]. PoisonAmplifier and NeighbourWatcher can be deployed as online services to proactively find compromised servers and spammer promoted servers. SMASH and VisHunter can be deployed at the network edge router to monitor network traffic, they will generate alerts when servers in malicious cyber infrastructures are detected.

Figure 1.2 illustrates our inference framework and four systems. Among these systems, PoisonAmplifier focuses on compromised servers detection, SMASH focuses on Type A and Type B malicious server detection. VisHunter mainly focuses on redirections from compromised servers to Type A malicious server and NeighbourWatcher focuses on all spammer promoted servers, which may include compromised servers, Type A and Type B malicious server. In addition, while SMASH and VisHunter are passive monitoring systems, PoisonAmplifier and NeighbourWatcher use active strategies to proactively find more compromised servers

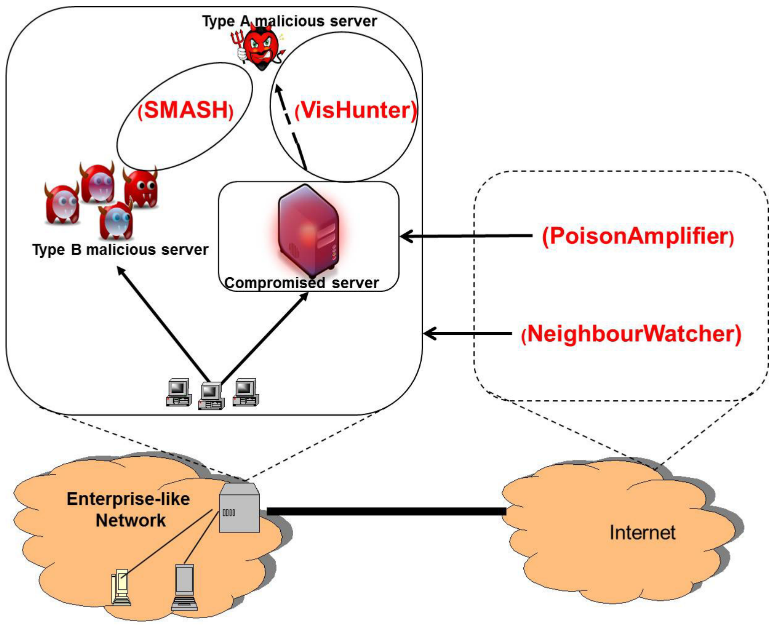


Figure 1.2: Our malicious cyber infrastructure detection framework

and malicious servers without relying on specific network traffic.

PoisonAmplifier focuses on compromised servers, which are usually used as "stepping stone" servers to redirect the traffic to malicious servers during the search poisoning attack. Most existing work [79, 104–106] on search poisoning attacks focus on detection, however, this dissertation exploits search poisoning attacks to proactively find more compromised servers. Through studying the search poisoning attack, PoisonAmplifier can automatically discover compromised websites that are utilized by attackers to launch search poisoning attacks. Particularly, starting from a small seed set of known compromised websites utilized in search poisoning attacks, PoisonAmplifier can automatically find more compromised websites by analyzing special terms and links on poisoned webpages, and exploring compromised websites' vulnerabilities.

SMASH focus on Type A and Type B malicious server involved in malicious cyber infrastructures. It leverages an insight that cybercriminals are increasingly using multiple malicious servers in their malicious activities. Therefore, instead of focusing on detecting individual malicious domains [60, 63], it identifies a group of closely related servers that are potentially involved in the same malware campaign. Specifically, it utilizes an unsupervised framework to infer malware associated server herds (ASHs) by systematically mining the relationships among servers from multiple dimensions.

VisHunter mainly focus on redirections from compromised servers to malicious servers (we define such redirections as the entrances to malicious cyber infrastructures). However, its propagation component can also find more post-infection servers[3]. VisHunter uses the fact that cybercriminals make efforts to conceal their core servers (e.g., C&C servers, exploit servers, and drop-zone servers) in the malicious cyber infrastructures in order to continue their malevolent activities without being detected. We characterize such deliberate invisibility of those concealed malicious servers by using a new property named server visibility. Base on this insight, we conduct the first large-scale measurement study investigating the visibility of both malicious and benign servers and identify a set of distinct features of malicious web infrastructures from their locations, structures, roles, and relationship. Based on the insights obtained from the study, VisHunter identifies malicious redirections from visible servers to invisible servers at the entryway of malicious cyber infrastructures through a trained classifier and detects post-infection servers through a novel graph-based inference algorithm.

PoisonAmplifier, SMASH and VisHunter have some limitations. They can only find servers with part of three roles. In addition, SMASH and VisHunter can only passively find malicious servers. Thus, to complement these systems, we pro-

[3]Post-infection servers are the servers directly connected by the infected clients after infection

pose NEIGHBOURWATCHER, which can pro-actively find spammer promoted servers without relying on any specific network traffic. While existing work [80, 89] relies on the content of postings, we utilize the spamming infrastructure for the promoted server detection, which is much more robust for evasion. We leverage the observation that cybercriminals usually have a limited number of promoting harbors (i.e., normal websites allowing users to leave comments such as forums, wikis, guestbooks) and they want to make full utilization of these harbors to promote their malicious servers. Therefore, NEIGHBOURWATCHER exploits spammers spamming infrastructure information to infer cybercriminals' promoted servers.

Our inference framework and four detection systems meet design goals mentioned above. We provide a brief explanation here and leave the details for the remaining chapters:

First, each system captures some perspectives of the invariants of malicious cyber infrastructures, i.e., POISONAMPLIFIER explores attackers' attacking pattern for compromised server detection, VISHUNTER explores the hidden pattern of malicious cyber infrastructure, SMASH captures the backup pattern of malicious cyber infrastructure and NEIGHBOURWATCHER captures the promoting pattern. We believe these principles can potentially capture the invariants of future malicious cyber infrastructures.

Second, each system covers a different part of malicious cyber infrastructures. For example, POISONAMPLIFIER focuses on compromised servers and SMASH focused on Type A and Type B malicious server. In addition, some techniques by themselves can cover multi-dimensions. For instance, SMASH correlated malicious servers from multi-dimensions. Furthermore, each system may have its own limitations and coverage. However, they can complement each other to enlarge the detection coverage from multiple different perspectives.

Third, most of our systems are general. In design, they target generate malicious cyber infrastructures and can detect a variety of malicious servers with diverse functionalities. In other words, they are not restricted to a very specific malicious cyber infrastructure or a specific type of malicious servers.

Fourth, most of our systems can be easily deployed without any special requirement to the network traffic. Comparing to system [99] which requires a large diverse user base for malicious web pages detection, our system SMASH and VisHunter can detect malicious servers involved in malicious activities even only few clients get infected.

Finally, our systems are practical and can work in real word. All of our system can detect malicious servers missed by existing state-of-the-art anti-virus tools. This will be demonstrated in the following chapters.

1.4 Thesis Contribution and Organization

In this dissertation, we make the following main contributions:

1. We characterize the malicious cyber infrastructure in three roles and propose an inference system to detect servers involved in malicious cyber infrastructures from different novel perspectives, i.e., PoisonAmplifier explore attackers' attacking pattern to find more compromised servers and VisHunter explores the invisibility of malicious cyber infrastructures to detect malicious servers.

2. We provide four practical detection prototype systems (i.e., PoisonAmplifier, SMASH, VisHunter, and NeighbourWatcher) to detect malicious cyber infrastructures. These systems are evaluated on real-world network traffic and/or online public dataset. They are shown to accurately detect different malicious servers with a low false positive rate.

3. We provide a relatively sound principle to capture some fundamental patterns

of malicious cyber infrastructures. And we believe the general principles behind these systems could also be used to detect future advanced malicious servers and infrastructures, and can be applicable to the malicious infrastructure study over other platforms such as social networks and mobile app markets.

The remainder of this thesis is organized as follows. In the next chapter, this dissertation introduces related work and explains why the existing work is not adequate to detect malicious cyber infrastructures. Chapter 3 presents an approach to pro-actively discover more compromised web servers. This dissertation further describes a novel system to infer a group of malicious servers involved in the same malicious cyber infrastructures by characterizing their relationships in Chapter 4, and a system to detect post-infection servers and entrances to malicious cyber infrastructures in Chapter 5. Chapter 6 presents a system to pro-actively discover spammer promoted servers. Chapter 7 summarizes the lessons learned from our systems and demonstrates how to apply lessons learned from malicious cyber infrastructures in this dissertation to study the malicious infrastructures over other platforms such as malicious social network infrastructures and malicious app infrastructures. Finally, Chapter 8 concludes the thesis and describes directions for the future work.

2. RELATED WORK

In the previous chapter, we identified the research challenges for the malicious cyber infrastructure detection. In this chapter, we will answer the question why existing techniques are not sufficient for the malicious cyber infrastructure detection.

2.1 Compromised Server Detection

Existing systems to detect compromised servers fall into two main categories. The first category focuses on analyzing web content to determine the maliciousness of a website. For example, compromised servers can be detected by analyzing changes between the base version and a modified version of web contents [65] and JavaScript libraries [87]. Others utilized instrumented browsers [95] or JavaScript engines [68] to automatically visit suspicious websites, and examine the run-time system or browser behaviors for the signs of drive-by download attacks. The second category investigates evasion behaviors that attackers use to hide their malicious activities. For example, [102, 107] used web search cloaking to detect compromised servers. All of these existing work passively detected compromised web servers.

In a recent concurrent study, EvilSeed [77] proposed a framework that can actively find compromised servers with the help of search engines. It searched the web for pages that are likely malicious by starting from a small set of malicious pages. However, EvilSeed used generic signatures in its SEO gadget to detect compromised servers while our system PoisonAmplifier extracted the content that the attackers intend to promote. Therefore, PoisonAmplifier can find more search poisoning compromised websites more efficiently and effectively than EvilSeed, e.g., the hit rate of EvilSeed is 0.93% in its SEO gadget compared with 6.87% hit rate in PoisonAmplifier. We consider our system PoisonAmplifier as a good complement to

EvilSeed.

2.2 Malicious Server Detection

Detecting malicious domains has been widely studied from different angles. Many schemes detected malicious domains from the DNS point of view. In [60, 63], the authors used different features (e.g., the number of distinct TLDs, number of distinct malware samples that contacted the domain, changes in the number of requests to a domain) to evaluate the reputation of each single domain *in isolation*. However, these methods need initial malicious domains as seeds to train their systems. In addition, since they are focuses on the single domain detection, they miss the connections among malicious servers and can not capture the complete picture of malicious campaigns. In [61], Antonkakis et. al. focused only on malicious domains generated by DGA malware, and such method can not be applied to general malicious domains. Kopis [60] can be used to detect general malicious domains. However, it needs to monitor DNS traffic at the upper DNS hierarchy, which dramatically limits its application.

Another line of research detects malicious domains by extracting signatures from malware traffic and applying generated signatures to live network to detect malware traffic [90, 93]. Perdisci et al. [93] proposed a system that clusters malware samples requesting similar URLs and generates structure signatures from them. The generated signatures can be used to detect infected hosts on live networks. Nelms et. al. [90] improved upon [93] by generating an "adaptivetemplate." Their system automatically built C&C templates from known malware traffic and self-tuned to apply the templates to different environments. These work requires malware seeds to generate the templates, therefore, they can not detect evolved malware. In addition, some of the templates they generated could be easily evaded by attackers. Our system

SMASH is a completely unsupervised system that does not need malicious traffic seeds to train features or build templates/signatures and our system VisHunter captures the fundamental pattern of malicious servers, named server visibility, which can not be easily evaded by attackers without incurring any significant costs.

Gu et al. [71, 72] proposed anomaly-based botnet detection systems that look for similar network behaviors across client hosts. A set of bots that share similar anomaly patterns are detected as botnets. Yen et al. [109] detected malware by aggregating traffic that shares the same external destinations or similar payload, and involve internal hosts with similar OS platforms. The intuition behind these work is that hosts infected with the same bot malware usually have common C&C communication patterns. Therefore, they inferred the infected clients by analyzing the relationship among clients. Different from these work, our four systems focus on malicious servers; we study the relations among servers because server-side infrastructure is more robust and stable. While malware can easily randomize client-side traffic patterns (e.g., injecting random content in their packets, sending requests to random benign websites), they inevitably need to contact their malicious servers to fulfill their desired functions. In addition, client-based approaches usually require multiple infections of clients in a network. We believe server-side based detection system is an excellent complementary to the existing client-side based detection systems.

2.3 Malicious Redirection Detection

Most malicious redirections are actually from compromised servers to malicious servers. Leontiadis et al. [82] conducted the first measurement study on a search poisoning attack and found that some high-ranking websites were compromised to dynamically redirect users to online pharmacies. Later, Lu et al.[88] detected malicious redirection chains in a search poisoning attack using a group of features (e.g., poisoning resistance) specific to search poisoning activities. Lee et al. [81] identified malicious redirections on Twitter using the tweet features, such as appearing frequencies and the correlation of redirection chains in tweets. Wang et al. [102] proposed an approach to indirectly detect malicious redirections based on the cloaking techniques used by attackers. More similar to our work is that of Stringhini et al. [99] which detected general malicious servers using the features extracted from interactions between a crowd of web users' browsers with websites. However, an immediate limitation of the system is its requirements for a large and diverse user base which may limit its applicability in practice.

Our system VisHunter differs from the previous work in that we designed 12 features (8 of them are newly proposed) from visibility perspective to characterize the fundamental differences between benign and malicious redirections, which are more robust against manipulation. In addition, our system does not require large diverse user base and we can detect malicious entrances when there are only a few clients accessing them.

2.4 Malicious Cyber Infrastructure Detection

Most of the existing research on malicious web infrastructure study only focused on specific attack channels associated with malicious web infrastructures. Anderson et al. [59] studied the Internet infrastructure used to host and support scams in terms of its lifetime, stability, and so on. Li et al. [86] focused on malicious web advertising and built a system to inspect advertisement delivery processes to detect malicious advertising activities. Recently, Li et al. [85] conducted a study on general malicious web infrastructures based on the redirection topology and detected 12 times more malicious servers. However, their system required initial malicious seeds for bootstrapping and require a large number of redirections. In addition, it can not cover malicious servers accessed through directly visiting. In our inference framework, we detect servers in malicious cyber infrastructure based on their access pattern (e.g., directly connecting or connected through redirections). Thus, our framework is not restricted to a specify attack channel. In addition, for each role in malicious cyber infrastructures, we characterize it from different novel perspectives, which are complementary to each other to provide a relatively complete view of malicious cyber infrastructures.

3. DISCOVERING COMPROMISED WEBSITES THROUGH REVERSING SEARCH POISONING ATTACKS*

We have introduced our proposed model of malicious cyber infrastructures and explained why previous work cannot sufficiently to detect servers behind malicious cyber infrastructures. In this chapter, we present a novel and efficient approach, PoisonAmplifier, to find compromised websites on the Internet that are utilized by attackers to launch search poisoning attacks.

Search poisoning attacks, as one particular type of "black hat" Search Engine Optimization (SEO) manipulation techniques, inject malicious scripts into compromised websites and mislead victims to malicious websites by taking advantages of users' trust on search results from popular search engines. By launching search poisoning attacks, attackers can achieve their malicious goals such as spreading spam, distributing malware (e.g., fake AntiVirus tools), and selling illicit pharmaceutical products [36]. In April 2011, many search terms (e.g., those related to the royal wedding between Britain Prince William and Catherine Middleton) were poisoned with Fake AntiVirus links [39]. These links misled victims to install fake Security Shield AntiVirus software. In 2011, one research group from Carnegie Mellon University also reported substantial manipulation of search results to promote unauthorized pharmacies by attackers through launching search poisoning attacks [8]. Essentially, search poisoning attacks compromise benign websites by injecting malicious scripts either into existing benign webpages or into newly created malicious pages. Then,

*Part of this chapter is reprinted with permission from "PoisonAmplifier: A Guided Approach of Discovering Compromised Websites through Reversing Search Poisoning Attack" Jialong Zhang, Chao Yang, Zhaoyan Xu and Guofei Gu. In *Proceedings of the 15st International Symposium On Research in Attacks, Intrusions and Defenses (RAID'12)*, Copyright © 2012 by SpringerVerlag Berlin Heidelberg.

these scripts usually make the compromised websites respond with different web content to users that visit via or not via particular search engines. Specifically, once compromised websites recognize that the requests are referred from specific search engines, the compromised websites may lead the users to malicious websites through multiple additional redirection hops. However, if the compromised websites recognize that the requests are directly from users, the compromised websites will return benign content rather than malicious content. Thus, this kind of cloaking makes the attack very stealthy and difficult to be noticed. In addition, the good reputation of these (compromised) websites (e.g., many are reputable ".edu" domains) essentially help boost the search engine ranks and access opportunities of malicious webpages. Because of this, it would be helpful to discover as many of those compromised websites as possible.

Most current state-of-the-art approaches to find such compromised websites merely utilize pre-selected key terms such as "Google Trends [20]", "Twitter Trending Topics [47]" or specific "spam words" to search on popular search engines. However, the number of newly compromised websites discovered by using this kind of approaches is highly restricted to those pre-selected key terms. First, the limited number of the pre-selected terms will restrict the number of compromised websites that could be found. Second, since these terms usually belong to some specific semantic topics, it will be hard to find more compromised websites in different categories. In addition, since many pre-selected key terms (e.g., Google Trends) are also widely used in benign websites, such approaches will also search out many benign websites leading to low efficiency.

In this chapter, we propose a novel and efficient approach, PoisonAmplifier, to find compromised websites on the Internet that are utilized by attackers to launch search poisoning attacks. Specifically, PoisonAmplifier consists of five major components: Seed Collector, Promote Content Extractor, Term Amplifier, Link Amplifier, and Vulnerability Amplifier. Seed Collector initially collects a small seed set of compromised websites by searching a small number of terms on popular search engines. Then, for each known compromised website, Promote Content Extractor will extract "promoted web content", which is promoted by compromised website *exclusively* to search engine bots, but not seen by normal users. This web content is normally promoted by attackers, and always has a close semantic meaning to the content of the final malicious website (e.g., a website selling illicit pharmaceutical products). Through extracting specific query terms from "promoted web content", Term Amplifier will find more compromised websites by searching those query terms instead of simply using pre-selected key terms. The intuition behind designing this component is that attackers tend to provide similar key terms for search engine bots to index the webpages. For each compromised website, Link Amplifier first extracts two types of links: inner-links and outer-links. Inner-links refer to those links/URLs in the promoted web content of the compromised website. Outer-links refer to those links/URLs in the web content of other websites, which also have links to known compromised websites. Then, Link Amplifier finds more compromised websites by searching those inner-links and outer-links. The intuition is that links in the promoted content tend to link to other compromised websites. Also, the websites linking to known compromised websites may also link to other (unknown) compromised websites. Vulnerability Amplifier will find more compromised websites, which have the similar system or software vulnerabilities to existing known compromised websites. The intuition is that attackers tend to exploit similar vulnerabilities to compromise websites for search poisoning attacks. Through implementing a prototype system, PoisonAmplifier, our approach can find around 75,000 compromised websites by starting from 252 known comprised websites within first 7 days and continue to find 827 new compromised websites everyday on average thereafter. In addition, our approach can achieve a high Amplifying Rate[1], much higher than existing work [83, 88].

3.1 Problem Statement

We next introduce the research scope of this work. As we mentioned before, our research goal is to proactively find more compromised web servers. However, a web server could be compromised by different attackers and could be used as different functions. In this work, we focus on proactively finding one specific type of compromised servers which are utilized in Search Poisoning Attack as “stepping stone” servers. In the Search Poisoning Attack, this kind of compromised servers usually responds with malicious content to the users referred via search engines, while responds with non-malicious content to the direct visiting users.

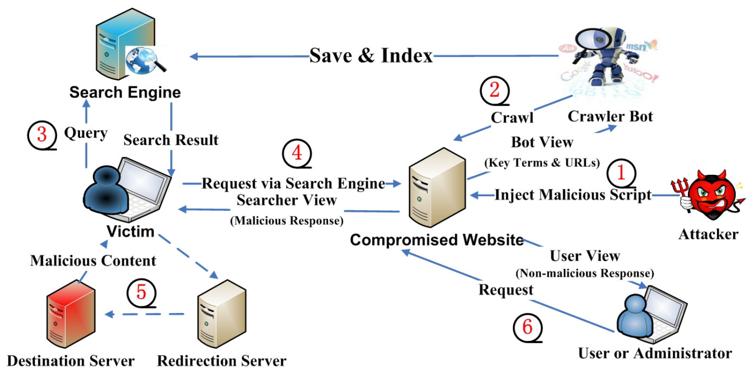


Figure 3.1: The workflow of search poisoning attacks.

Figure 3.1 shows the workflow of search poisoning attacks. To launch such search poisoning attacks, an attacker typically needs to first compromise a website by ex-

[1]It is the ratio of the number of newly discovered compromised websites to the number of seed compromised websites.

ploiting the website's system/software vulnerabilities, and then injects malicious scripts (PHP or Javascript) into the compromised website (labeled as ① in Figure 3.1). The core of such search poisoning attack is the ability for the compromised website to utilize injected malicious scripts to recognize different origins of the requests. Specifically, once the compromised website finds that the requests originate from crawler bots such as Google Bots, the website responds with web content containing special keywords and URLs injected by attackers. These special keywords and URLs are essentially what attackers desire to promote exclusively to the search engine crawler bots and hope to be indexed by the search engines(②). Then, if a user queries those keywords on search engines (③) and sends requests to the compromised website by clicking on the search results, the user will be the desired target victim because he shows interest in finding this website. In this case, the compromised server will provide malicious content to the user (④). The malicious response could directly be malicious web content or redirect the user to malicious websites through multiple redirection hops (⑤). However, if the request originates from direct users (not via specific search engines), attackers will not intent to expose the malicious content. This cloaking technique can make such attack very stealthy. In this case, the compromised website will return non-malicious content (⑥).

In our work, we define the web content responded by the compromised website (after redirection if it has) to the crawler bot as "Bot View", to users via the search engine as "Searcher View", and to users *not* via the search engine as "User View". We apply a similar technique used in [83, 101] to collect the ground truth on whether a website is compromised by search poisoning attack or not, i.e., whether its Searcher View and User View are different. More precisely, we *conservatively* consider the two views (Searcher/User) are different only when their final domain names (after redirection if there is any) are different [83]. In this way, we can reduce false positives (due to dynamics in benign websites) and increase our confidence. [2]

3.2 System Design

3.2.1 Intuitions

Our design of PoisonAmplifier is based on the following four major intuitions:

Intuition 1: Attackers tend to use a similar set of keywords in multiple compromised websites (in the Bot View) to increase the visibility to desired users through search engines. Attackers usually artificially construct the content of Bot View, which will be indexed by search engines, to increase the chance of making compromised websites be searched out by users through search engines. More specifically, similar to keyword stuffing [26], a common way of achieving this goal is to put popular keywords (those words are frequently searched by users on search engines such as Google Trends) into the Bot View. In this way, different compromised websites may share the similar popular keywords to draw attention from victims. However, since many popular benign websites may also use these popular keywords and thus occupy high search ranks, it is difficult to guarantee high search ranks for those compromised websites that may be not very popular. As a supplement, another way is to buy some "distinguishable keywords" from specific websites [25]. These keywords may be not so popular as those popular terms. However, they tend to have low competition in search engines, i.e., they are not widely contained in the websites and can be effectively used to search out target websites. Thus, through promoting these words in the Bot View, the compromised websites could be easily searched out when users query these keywords in search engines. Thus, attackers may use these "distinguishable keywords" in multiple compromised websites to increase

[2]Note that we may have very few false negatives using this conservative comparison. However that is not a problem for us because our goal is *not* on the *precise* detection of *all* compromised websites, but on the high efficiency in finding more compromised websites.

their search ranks.

In addition, since some attackers desire to return malicious content to their target victims rather than arbitrary users, they tend to put specific keywords into the Bot View of compromised websites, which have close semantic meanings to the content of promoted websites. For example, some attackers tend to post pharmacy-related words into the Bot View, because they will finally mislead victims who are interested in buying pharmaceutical products to malicious websites selling illicit pharmaceutical products. In this way, different attackers who promote similar malicious content may spontaneously use similar keywords in the Bot View.

Based on this intuition, once we obtain those specific keywords injected by attackers into the Bot View of known compromised websites, we can search these keywords in search engines to find more compromised websites.

Intuition 2: Attackers tend to insert links of compromised websites in the Bot View to promote other compromised websites. To increase the chance of leading victims to malicious websites, attackers usually use multiple compromised websites to deliver malicious content. Thus, to increase the search ranks of those compromised websites to search engines, or to help newly created webpages on compromised websites be indexed by search engines, attackers tend to link their compromised websites with each other by inserting links of other compromised websites into the Bot View.

Based on this intuition, we can find more compromised websites by searching the injected links in the Bot View.

Intuition 3: Attackers tend to post links to compromised websites in benign third-party websites to trick victims into directly clicking or to promote these compromised websites. Through posting compromised websites into third-party websites such as forums, blogs, or guestbooks, attackers can easily trick visitors into directly clicking these compromised websites when users review content in third-party websites. Also, third-party websites with high reputations are usually indexed by search engines frequently, which can help compromised websites be indexed by search engines quickly, and even inherit some reputations from these third-party websites. Furthermore, attackers with different malicious goals may spontaneously promote links of compromised websites into the same popular third-party websites, either because these third-party websites are easy to be indexed by search engines, easy for automatically posting, or they do not have sanitation mechanisms. With these benefits, attackers intend to keep promoting compromised websites on these third-party websites to maximize their profits, which also gives us a chance to keep monitoring newly compromised websites.

Based on this intuition, we can find more compromised websites by searching the links in the web content of third-party websites, which have already been exploited to post links linking to known compromised websites.

Intuition 4: Attackers tend to compromise multiple websites by exploiting similar vulnerabilities. Once attackers compromise some specific websites by exploiting their system/software vulnerabilities to launch search poisoning attacks, they tend to use similar tricks (e.g., Google dork) or tools to compromise other websites with similar vulnerabilities to launch search poisoning attacks.

Based on this intuition, once we know the vulnerabilities signatures exploited by attackers to some compromised websites, we can find more compromised websites by searching websites with similar vulnerabilities.

3.2.2 System Overview

We next introduce the system overview of PoisonAmplifier, based on four intuitions described in Section 3.2.1. As illustrated in Figure 3.2, PoisonAmplifier mainly contains five components: Seed Collector, Promoted Content Extractor, Term Amplifier, Link Amplifier, Vulnerability Amplifier.

- Similar to other existing work [88, 104], the goal of **Seed Collector** is to collect a seed set of compromised websites by searching initial key terms (e.g., Google Trends) in popular search engines.
- For each compromised website, **Promoted Content Extractor** will first work as a search engine bot to crawl the website's Bot View, and then work as a real user to obtain the websites' User View. Then, Promoted Content Extractor will extract those content that exists in the website's Bot View but does *not* exist in the website's User View. This content, defined as "promoted content", is essentially what attackers desire to promote into search engines.
- After extracting the promoted content, **Term Amplifier** extracts special key terms by analyzing the promoted content and querying these key terms in search engines to find more compromised websites.
- **Link Amplifier** extracts URLs in the promoted content. Link Amplifier will also extract URLs contained in the web content of third-party websites, which have already been posted links to known compromised websites. Then, Link Amplifier will analyze these URLs to find more compromised websites.
- By analyzing system/software vulnerabilities of those seed compromised websites and newly found compromised websites through using Term Amplifier and Link Amplifier, **Vulnerability Amplifier** finds more compromised websites by searching other websites with similar vulnerabilities.

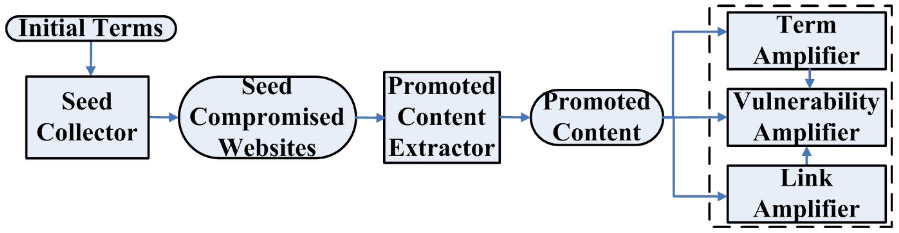


Figure 3.2: The system architecture of PoisonAmplifier.

3.2.3 Seed Collector

As illustrated in Figure 3.3, Seed Collector mainly uses the following four steps to collect seed compromised websites: (1) it first uses Google Trends [20], Twitter trends[47], and our customized key terms as initial key terms to search on search engines. (2) For each term, it will extract the links of the top M search results showed in the search engine[3]. (3) For each link, Seed Collector crawls its Searcher View and User View through utilizing HttpClient-3.x package[24][4] to set different HTTP header parameters and values. Specifically, to crawl the Searcher View of the website linked by each search result, we send HTTP requests with customized Http Referrer (http://www.google.com/?q="term") to simulate a user to visit the website through searching Google. To crawl the User View, we send HTTP requests with customized values of UserAgent in the HTTP header (e.g., UserAgent: Mozilla/5.0 (Windows NT 6.1), AppleWebKit/535.2 (KHTML, like Gecko), Chrome/15.0.874.121, Safari/535.2) to simulate a user to directly visit the website. For both User View and Searcher View, the seed collector follows their redirection chains and gets their final destination URL. (4) For each link, if its final destination domains between User View and Searcher View are different, we consider that this

[3]In our experiment, we choose $M = 200$.

[4]This package can handle HTTP 3xx redirection and provide flexible HTTP header configuration functions

website is compromised and output it as a compromised website.

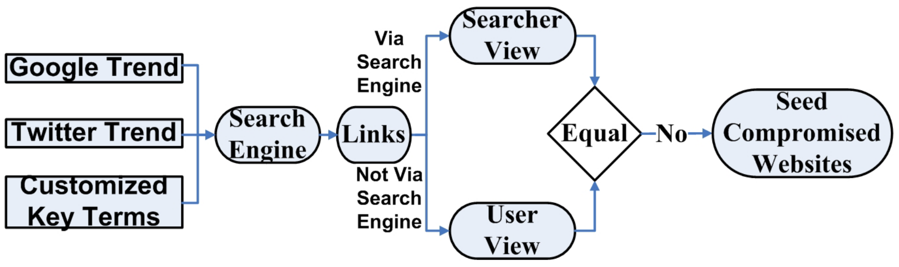


Figure 3.3: Flow of seed collector.

3.2.4 Promoted Content Extractor

As described in Section 6.1.1, the essence of the search poisoning attack is to recognize different request origins and provide different web content to crawler bots (Bot View), to users via search engines (Searcher View), and to users not via search engines (User View). Attackers tend to inject specific content into Bot View to increase the chances of their compromised websites to be searched out in search engines. They may also tend to inject malicious content that is related to the final promoted destination malicious websites. This content is usually different from normal web content, and can not be seen by users without using search engines.

The goal of the Promoted Content Extractor is to extract that injected content in the Bot View of known compromised webpages, which may also be contained in other compromised websites. Note that Bot View may also contain normal content that is not injected by attackers and will be displayed in the User View. To be more effective, PoisonAmplifier only extracts and analyzes the content that is in the Bot View but is *not* in the User View, i.e., the content will be indexed by crawler bots, but not be seen by users directly visiting the websites. As illustrated in Figure 3.4, for each compromised website, Promoted Content Extractor crawls its Bot View and User View through sending crafted requests from crawler bots and users without using search engines, respectively. Specifically, to crawl the Bot View, we send a request with customized value of UserAgent in the HTTP header (e.g., UserAgent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)) to mimic a Google bot visit. Promoted Content Extractor crawls the User View in the same way as Seed Collector. Then, Promoted Content Extractor extracts HTML content that appears in the Bot View but not in the User View. Then, it further filters web content that is used for displaying in the web browsers such as HTML Tags and CSS codes, and also removes dynamic web function related codes such as Javascript, which are not unique enough to help further amplification. Finally, it outputs extracted "Promoted Content" after filtering.

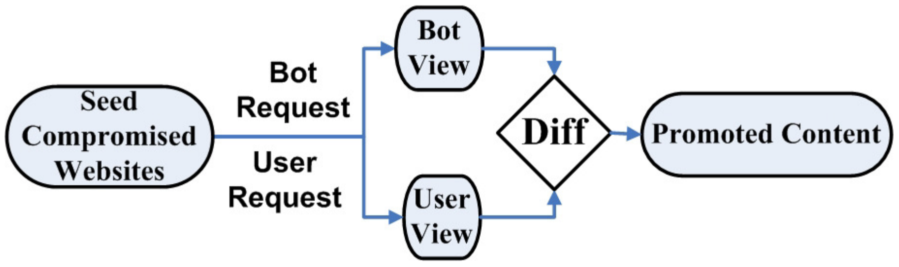


Figure 3.4: Flow of promoted content extractor.

It is worth noting that some legitimate websites with dynamic server-side codes can also return different content or even redirect to different websites for every request no matter where the visit is from (User View or Bot View), which may lead to false positives in our extracted promoted content. To decrease this kind of false positives, we crawl the User View twice within a short time period. In this case, if the two

User Views are different, we will conservatively consider that this website is *not* compromised (even its User View and Searcher View may be different) and discard it for promoted content extraction.

3.2.5 Term Amplifier

Based on Intuition 1 in Section 3.2.1, the goal of Term Amplifier is to find more compromised websites through searching specific terms extracted from promoted content.

It is worth noting that if we use less distinguishable content as query terms to search, we can obtain a higher recall number (more compromised websites could be returned) but a lower accuracy (top search results are less likely to be compromised websites), and vice versa. In addition, in order to obtain a higher accuracy, it is practical to focus on analyzing replied search results with top search ranks rather than analyzing all search results. Thus, the essential part of Term Amplifier is how to extract effective query terms from promoted content, through searching which we can obtain as many compromised websites as possible with a high accuracy. One option is to use each word/phrase in the content as one query term. However, in this way, some terms may be so general that most returned websites are benign, leading to a low accuracy. Another option is to use the "n-gram" algorithm [32] ($n \geq 2$). In this way, some terms may be so distinguishable that many compromised websites will be missed, leading to a low recall number.

In our work, we design an algorithm, named "distinguishable n-gram", to extract query terms. As illustrated in Figure 3.5, Term Amplifier first tokenizes the promoted content into a sequence of phrases $\{P_i|i = 1, 2, \ldots, N\}$ by using the tokenizer of any non-Alphanumeric character except "blank", such as "comma", "semicolon", and "question mark". Then, for each phrase P_i, Term Amplifier will *exactly search* it

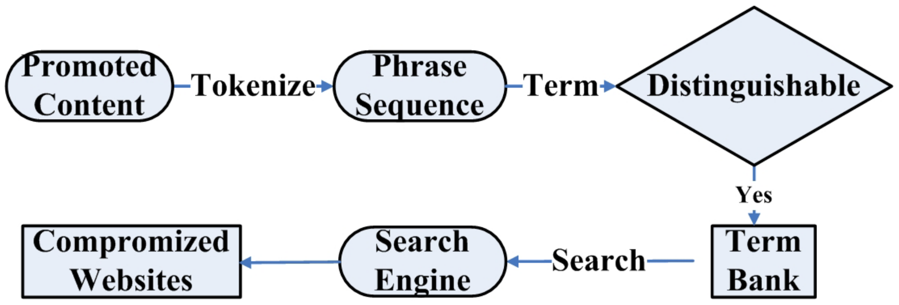


Figure 3.5: Flow of term amplifier.

on the search engine. If the number of returned search results SN_i is lower than a threshold T_D[5], we consider P_i as a "distinguishable" term and directly add it into a term set, named TermBank. Otherwise, Term Amplifier combines the phrases of P_i and P_{i+1} as a new query term to search. If this new term is "distinguishable", we add it into TermBank; otherwise, Term Amplifier combines the phrases of P_i, P_{i+1} and P_{i+2} as a new term to search. This process will continue until the number of phrase in the new term is equal to n. If the new term with n phrases is still not "distinguishable", the algorithm will discard the phrase P_i. In this way, TermBank comprises all the distinguishable terms extracted from the promoted content. The detailed description of "distinguishable n-gram" algorithm is shown in Algorithm 1.

After building TermBank, similar to Seed Collector, Term Amplifier uses each query term in TermBank to search in the search engine and identifies compromised webpages through comparing their Searcher Views and User Views.

3.2.6 Link Amplifier

Based on Intuition 2 and Intuition 3 in Section 3.2.1, Link Amplifier first extracts two types of links: inner-links and outer-links. Inner-links refer to those links/URLs

[5]T_D can be tuned with the consideration of the tradeoff between the accuracy and the recall number. In our preliminary experiment, we choose $T_D = 1,000,000$.

Algorithm 1 : Distinguishable n-gram Algorithm

```
Tokenize promoted content into phrases $\{P_i | i = 1, 2, \ldots, N\}$
for $i := 1$ to $N$ do
    for $j := 0$ to $n - 1$ do
        Search “$P_i \ P_{i+1} \ \ldots \ P_{i+j}$” on the search engine to get $SN_i$
        if $SN_i \leq T_D$ then
            Add ‘$P_i \ P_{i+1} \ \ldots \ P_{i+j}$” into TermBank
            CONTINUE
        end if
    end for
end for
Return TermBank
```

in the promoted web content of the compromised websites (as illustrated in the left part of Figure 3.6). Outer-links refer to those links/URLs in the web content of third-party websites, which have been posted with URLs linking to known compromised websites (as illustrated in the right part of Figure 3.6). We utilize Google dork [21] to locate the outer-links. For example, if one compromised website “seed.com” is obtained through searching seed term “seedTerm”, then we obtain those websites through searching “*intext:seed.com*” on Google. Then we crawl all the websites in search results, which usually are benign third-party websites such as blogs, forums, and some company guestbooks, and extract postings that contain ”seed.com” and other scam links from these third-party websites. Then, similar to Term Amplifier, for each inner-link and outer-link, Link Amplifier crawls its Searcher View and User View, and considers the linking website as compromised website if the Searcher View and User View are different.

We acknowledge that since most of those third-party websites are benign and they may also be posted with many benign links, this may lead to a relatively low accuracy for outer-links. However, there are still several benefits for studying outer-links: (1) Through analyzing those outer-links, we can find more categories of compromised

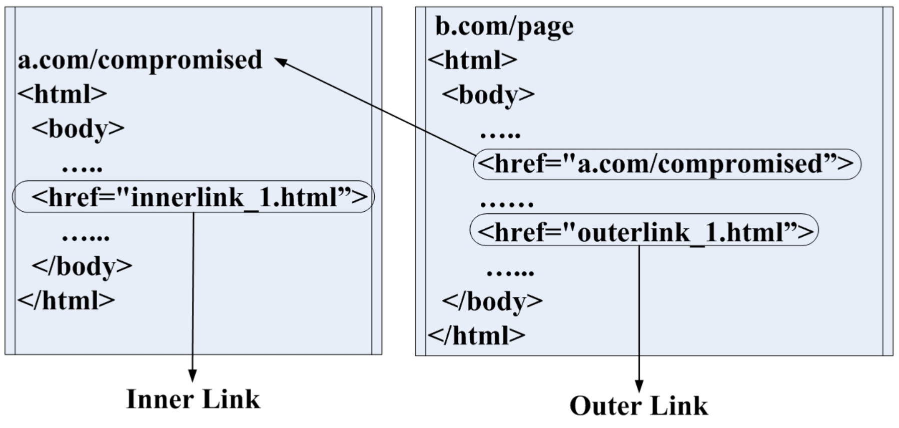
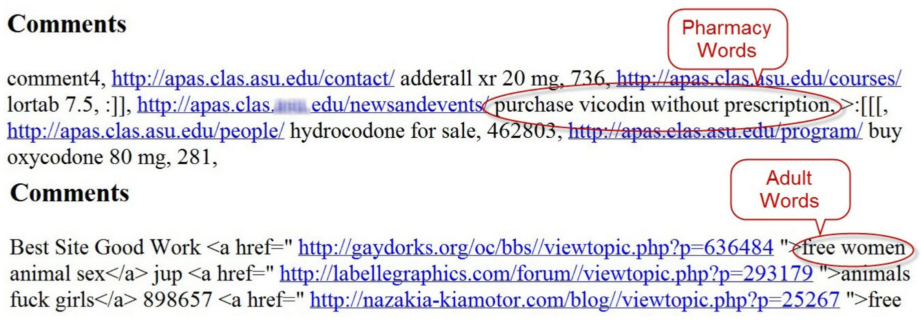


Figure 3.6: The illustration of inner-links and outer-links.

websites because different attackers may explore the same third-party websites to post outer-links, or the same attackers will promote different categories outer-links to evade detection mechanisms on these third-party websites. As an example, Figure 3.7 shows a benign forum webpage. We first find this forum webpage through searching a compromised website "*http://apas.clas.asu.edu/contact/*", a known pharmacy target compromised website. Then, through analyzing the content of the forum webpage, we can also find other compromised websites with "Adult" content such as "*http://nazakia-kiamotor.com/blog/viewtopic.php?p=25267*". (2) To keep promoting new/existing compromised websites, attackers intend to recycle these third-party websites, which may be easy to be spammed on, or easily be indexed by search engines. Thus, we can keep finding newly compromised websites in time by periodically analyzing outer-links. Furthermore, we can somehow increase the accuracy of Link Amplifier through focusing on only those third-party websites that have posted scam terms, such as "*intext:seed.com intext:seedTerm*". This is because that this kind of websites are more likely to be used to promote malicious content by attackers than other websites. Thus, the links posted on such websites are more suspicious.



Comments

comment4, http://apas.clas.asu.edu/contact/ adderall xr 20 mg, 736, http://apas.clas.asu.edu/courses/ lortab 7.5, :]], http://apas.clas.[redacted].edu/newsandevents purchase vicodin without prescription, >:[[[, http://apas.clas.asu.edu/people/ hydrocodone for sale, 462803, http://apas.clas.asu.edu/program/ buy oxycodone 80 mg, 281,

Comments

Best Site Good Work <a href=" http://gaydorks.org/oc/bbs//viewtopic.php?p=636484 ">free women animal sex</a> jup <a href=" http://labellegraphics.com/forum//viewtopic.php?p=293179 ">animals fuck girls</a> 898657 <a href=" http://nazakia-kiamotor.com/blog//viewtopic.php?p=25267 ">free

Figure 3.7: Example of outer-link.

3.2.7 Vulnerability Amplifier

Once an attacker compromises a website to launch search poisoning attack by exploiting specific system/software vulnerabilities of the websites, it is very likely that he will use the same vulnerability to compromise more websites. For example, once some attackers know about the vulnerabilities of some specific version of "WordPress" [55] and successfully use some existing tools to compromise some websites with the specific version of WordPress, they may try to find other vulnerable websites that are also implemented with that version of WordPress. One possible simple way of finding those vulnerable websites could be to search keywords such as "powered by WordPress" on search engines.

Based on Intuition 4 in Section 3.2.1, Vulnerability Amplifier essentially mimics the way of attackers to find those compromised websites. Specifically, Vulnerability Amplifier first collects compromised websites by using Term Amplifier and Link Amplifier. Then, it will analyze possible system/software vulnerabilities of those compromised websites and extract the web content signature of the websites that utilize the vulnerable software. In our preliminary work, we only focus on analyzing the vulnerabilities of one specific software WordPress[6], which is a very popular target

[6]Even though we only analyze the vulnerabilities of one specific software in this work, our

for attackers [1] recently. For example, one vulnerability of "Timthumb.php" in the WordPress themes allows attackers to upload and execute arbitrary PHP scripts. Vulnerability Amplifier will find compromised websites through searching those websites that use WordPress and contain at least one scam word. Since the URLs of the websites developed by WordPress typically contain a string of "wp-content", we can find those websites through searching Google Dork "*inurl:wp-content intext:scamWord*". After visiting each of such websites, Vulnerability Amplifier examines whether it is compromised or not by comparing its Searcher View and User View.

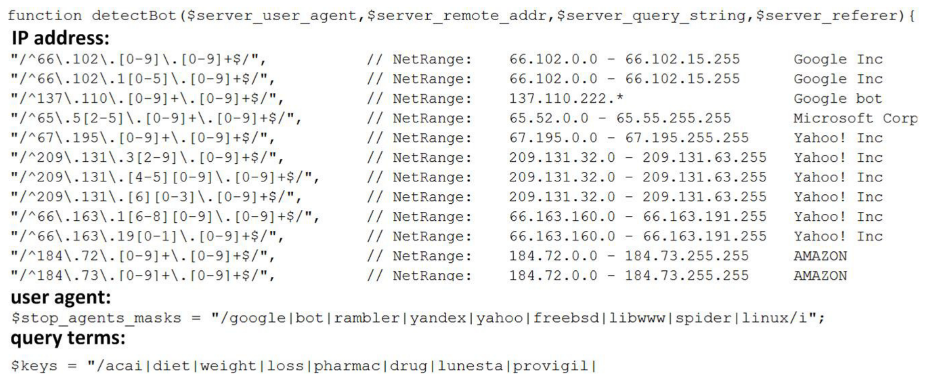
3.3 Evaluation

In this section, we first give a real case of search poisoning attack observed in the wild. Then we evaluate PoisonAmplifier in two stages. For the first stage, since the goal of our work is not to comprehensively detect all compromised websites on the Internet, instead, we attempt to infer more compromised websites more efficiently starting from a given small seed set, thus we evaluate PoisonAmplifier regarding its effectiveness, efficiency, and accuracy with first 7 days' data. We also check the "discovery diversity" among different components in terms of finding exclusively compromised websites, i.e, how different the discovered compromised websites by different components are. In addition, we examine how existing Google security diagnostic tools in labeling malicious/compromised websites work on our found compromised websites. In the second stage, we extend time to 1 month to verify if PoisonAmplifier can constantly find newly compromised websites.

approach can easily include other types of system/software vulnerabilities, which is our future work.

3.3.1 Case Study

The prevalence of search poisoning attacks makes a variety of famous universities victims. During our study, we have cooperated with the network administrators of a compromised website in a university (after we notified them about the compromise). Thus, we collected the malicious scripts uploaded by attackers and have a chance to further analyze them. In this case, attackers explored a WordPress vulnerability to upload new spam pages. Then, if a user visits these pages from search engines, it will redirect the user to a pharmacy-related website. And if the user directly inputs the URL in a browser, it will return an HTML 404 (page does not exist) error. Figure 3.8 shows a snip of the malicious scripts uploaded by the attackers. To further understand how such attacks proceed, we analyze the malicious scripts from the following two perspectives:



```
function detectBot($server_user_agent,$server_remote_addr,$server_query_string,$server_referer){
IP address:
"/^66\.102\.[0-9]\.[0-9]+$/",           // NetRange:    66.102.0.0 - 66.102.15.255      Google Inc
"/^66\.102\.1[0-5]\.[0-9]+$/",          // NetRange:    66.102.0.0 - 66.102.15.255      Google Inc
"/^137\.110\.[0-9]+\.[0-9]+$/",         // NetRange:    137.110.222.*                   Google bot
"/^65\.5[2-5]\.[0-9]+\.[0-9]+$/",       // NetRange:    65.52.0.0 - 65.55.255.255       Microsoft Corp
"/^67\.195\.[0-9]+\.[0-9]+$/",          // NetRange:    67.195.0.0 - 67.195.255.255     Yahoo! Inc
"/^209\.131\.3[2-9]\.[0-9]+$/",         // NetRange:    209.131.32.0 - 209.131.63.255   Yahoo! Inc
"/^209\.131\.[4-5][0-9]\.[0-9]+$/",     // NetRange:    209.131.32.0 - 209.131.63.255   Yahoo! Inc
"/^209\.131\.[6][0-3]\.[0-9]+$/",       // NetRange:    209.131.32.0 - 209.131.63.255   Yahoo! Inc
"/^66\.163\.1[6-8][0-9]\.[0-9]+$/",     // NetRange:    66.163.160.0 - 66.163.191.255   Yahoo! Inc
"/^66\.163\.19[0-1]\.[0-9]+$/",         // NetRange:    66.163.160.0 - 66.163.191.255   Yahoo! Inc
"/^184\.72\.[0-9]+\.[0-9]+$/",          // NetRange:    184.72.0.0 - 184.73.255.255     AMAZON
"/^184\.73\.[0-9]+\.[0-9]+$/",          // NetRange:    184.72.0.0 - 184.73.255.255     AMAZON
user agent:
$stop_agents_masks = "/google|bot|rambler|yandex|yahoo|freebsd|libwww|spider|linux/i";
query terms:
$keys = "/acai|diet|weight|loss|pharmac|drug|lunesta|provigil|
```

Figure 3.8: Malicious script.

Search Bot Detection: To decide if the request comes from a search bot, the script defines a function "detectBot($ server_user_agent,$ server_remote_addr, $ server_query_string, $server_referer)", This function uses UserAgent extracted from

HttpHeader and the IP address of the requestor as inputs[7], and returns a boolean value to indicate whether the request comes from a search bot or not. Specifically, if the IP address of the requestor comes from Google Inc, Microsoft Corp, Yahoo, and McAfee, etc, or if the request with UserAgent containing keywords such as bot, google, yahoo in “$stop_agent_masks” in Figure 3.8, the function will return true to indicate a bot request. In this case, we can simulate search engine bots by manipulating bot UserAgent in HttpHeader.

Redirection Policy: After recognizing the requestor, the scripts will return the original content plus malicious content to a bot requestor. If the requestor is from human, the script further checks the referer and query string. Specifically, if the referer contains keywords “google”, the script will redirect users to a pharmacy-related website through a redirection chain. However, if the referer contains other search engine keywords such as Bing, aol.com, and ask.com, the script will need to further check that if the query string also contains target terms, such as key terms in “$keys” shown in Figure 3.8. If so, it will still redirect users to a pharmacy-related website. Otherwise, it will do nothing and return “not exist” pages. In this case, we can see that the script has a loose redirection policy for requests coming from Google while needs additional query string satisfaction for requests from other search engines. In our system, we simply simulate the search view by manipulating the referer as “google” and the query string with extracted terms. With this setting, it is possible that we can detect only a subset of all compromised websites. Thus, not surprisingly, the result here is only a low bound of actually compromised websites in the wild.

[7]Although the function defines 4 parameters, “server_query_string” and “server_referer” are not used in this function.

3.3.2 Evaluation Dataset

As mentioned in Section 5.4.2, the seed term set consists of three categories: Google Trends, Twitter Trends, and our customized keywords. For the Google Trend Topics, we crawled 20 Google Trend keywords each day for a week. In this way, we collected 103 unique Google Trends topics. For the Twitter Trends, we collected top 10 hottest Twitter trends each day for a week. In this way, we collected 64 unique Twitter Trends topics. For the customized key terms, we chose one specific category of scam words – pharmacy-related words[8]. Specifically, we chose 5 pharmacy-related words from existing work [103], which provides several categories of scam words. We also manually selected another 13 pharmacy-related words from several pharmacy websites. Table 3.1 lists all 18 pharmacy words used in our study.

Table 3.1: 18 seed pharmacy words

kamagra	diflucan	levitra	phentermine	propecia	lasix
viagra	amoxil	xanax	cialis	flagyl	propeciatramadol
zithromax	clomid	Viagra super active	cialis super active	cipro	pharmacy without prescription

Then, for each of 18 pharmacy words, we obtained another 9 Google Suggest words through Google Suggest API [22]. In this way, we finally collected 165 unique pharmacy words. Table 3.2 summarizes the total number and the unique number of seed terms for each category.

Then, for each of these 332 unique seed terms, we searched it on "Google.com" and collected the top 200 search results[9]. Then, for each search result, we use

[8]In our preliminary experiment, we only use pharmacy-related words. However, our approach is also applicable to other categories of words such as "adult words" or "casino words".

[9]In our experiment, we only focus on the search poisoning attacks on Google. However, our approach can be similarly extended to other search engines such as "yahoo.com" and "baidu.com".

Table 3.2: The number of seed terms for three different categories

Category	# of terms	# of unique terms
Google Trend	140	103
Twitter Trend	70	64
Pharmacy	180	165
Total	390	332

the similar strategy as in [83] to determine whether a website is compromised by examining whether the domain of its Searcher View and User View are different. In this way, we finally obtained 252 unique seed compromised websites through using those 332 seed terms. We denote this dataset as S_I, which is used in Stage I. After one week's amplification process, we denote amplified terms and compromised websites from Stage I as S_{II}, which is the input for Stage II to recursively run PoisonAmplifier for 1 month.

3.3.3 Evaluation Results

3.3.3.1 Effectiveness

To evaluate the effectiveness of our approach, we check how many new compromised URLs/domains can be found through amplifying dataset S_I. We use the metric,"Amplifying Rate (AR)", to measure the effectiveness, which is the ratio of the number of newly found compromised URLs/domains to the number of seed compromised URLs/domains. Thus, a higher value of AR implies that the approach is more effective in finding compromised URLs/domains.

Table 3.3 shows the number of newly found compromised URLs/domains for each component. We can see that Term Amplifier has the highest AR of 323.03 for URLs and 78.71 for domains, which confirms that Term Amplifier can be very effective in

Also, the number of 200 can be tuned according to different experiment settings.

discovering compromised websites. And Inner-link Amplifier has the lowest AR of 0.51 for domains, because attackers usually compromise multiple URLs in the same domain, and intend to use inner-links to promote each other. Thus, most inner-links belong to the same domains. In addition, even though Inner-link Amplifier and Outer-link Amplifier have relatively lower ARs than Term Amplifier, they can still discover over 10 times more compromised URLs from the seeds. Actually, the reason why Term Amplifier can obtain a higher AR is mainly because we can extract many more query terms than inner-links and outer-links from the promoted content. In this way, we can search out much more websites that contain the query terms from search engines. Furthermore, even though we only focus on analyzing one specific software in our Vulnerability Amplifier, we can still discover over 4 times more compromised URLs from the seeds.

Overall, starting from only 252 seed compromised websites, these four strategies can totally discover 74,671 unique compromised URLs and 17,258 unique compromised domains (on average 4.32 malicious URLs per domain), and achieve an overall high amplifying rate of 296 for URLs. The distribution information of these compromised websites in terms of their Top Level Domain(TLD) is shown in Figure 6.12(a).

Table 3.3: The effectiveness of PoisonAmplifier.

Component	**# Seed CW[1]**		**# UniqueCW[1]**		**Amplifying Rate**	
	URLs	Domains	URLs	Domains	URLs	Domains
TermAmplifier	252	212	69,684	16,688	323.03	78.71
Inner-linkAmplifier	252	212	2,468	109	10.63	0.51
Outer-linkAmplifier	252	212	2,401	660	10.34	3.11
VulnerabilityAmplifier	252	212	482	360	4.49	1.70
Total (Unique)	252	212	74,671	17,258	296.31	81.41

[1] Compromised Websites.

3.3.3.2 Efficiency

To evaluate the efficiency of our approach, we essentially examine whether the websites visited by PoisonAmplifier are more likely to be compromised websites or not. To measure the efficiency, we use another metric, named "Hit Rate (HR)", which is the number of newly found compromised websites to the total number of websites visited by PoisonAmplifier. Thus, a higher Hit Rate implies that our approach is more efficient because it means our approach can find more compromised websites by visiting fewer websites. Next, we evaluate the efficiency of individual amplification component, as well as the efficiency of different types of query keywords.

Component Efficiency. Table 3.4 shows the number of visited websites, the number of newly found compromised websites, and the values of hit rate for each component.

Table 3.4: The efficiency of different components.

Component	# Visited Websites	# CW[1]	Hit Rate
TermAmplifier	684,540	69,684	10.18%
Inner-linkAmplifier	3,097	2,468	79.69%
Outer-linkAmplifier	353,475	2,401	0.68%
VulnerabilityAmplifier	45,348	482	1.06%
Total	1,086,496	74,671	6.87%

[1] Compromised Websites.

From this table, we can see that Inner-link Amplifier can achieve the highest hit rate of 79.69%. This confirms that attackers tend to promote links of compromised websites to the search engine bots. The hit rate of Term Amplifier is around 10%, which is lower than that of Inner-link Amplifier. However, Term Amplifier can discover much more compromised websites than that of Inner-link Amplifier in terms of overall quantity because we essentially extract significantly more terms than inner-links to search on search engines. The hit rate of Outer-link Amplifier is relatively low, which is mainly because most of those outer-links are benign or do not have redirections. However, through using Outer-link Amplifier, we can find new types of scam terms promoted by different attackers. This is very useful to increase the diversity of the seed terms and to find more types of compromised websites. Vulnerability Amplifier also has a relatively low hit rate, because most top ranked websites with "WordPress" are benign. However, similar to Outer-link Amplifier, Vulnerability Amplifier also provides a method to find more (new) types of scam words and compromised websites.

Term Efficiency. We also analyze the term efficiency in finding compromised websites, i.e., which kinds of terms can be used to efficiently search out compromised (rather than normal) websites. Specifically, we compare three types of terms: seed terms (those 332 seed terms used in the Seed Collector), promoted phrases (the sequence of phrases obtained through tokenizing promoted content), and distinguishable terms (all the terms in TermBank obtained by utilizing "Distinguishable n-gram Algorithm"). Essentially, we use these three types of terms to search on Google to find compromised websites by utilizing Term Amplifier.

As seen in Figure 3.9, among these three types of terms, our extracted distinguishable terms can achieve the highest hit rate. Specifically, around 60% of distinguishable terms' hit rates are less than 0.2, while around 80% of promote phrases and 90% of seed terms have such values. This implies that using distinguishable terms is more effective to find compromised websites. In addition, over 60% of seed terms' hit rates are nearly zero, which shows that the current pre-selected terms are not very efficient compared to our new terms extracted from promoted content.

To find what specific terms are most efficient, we further analyze the terms with

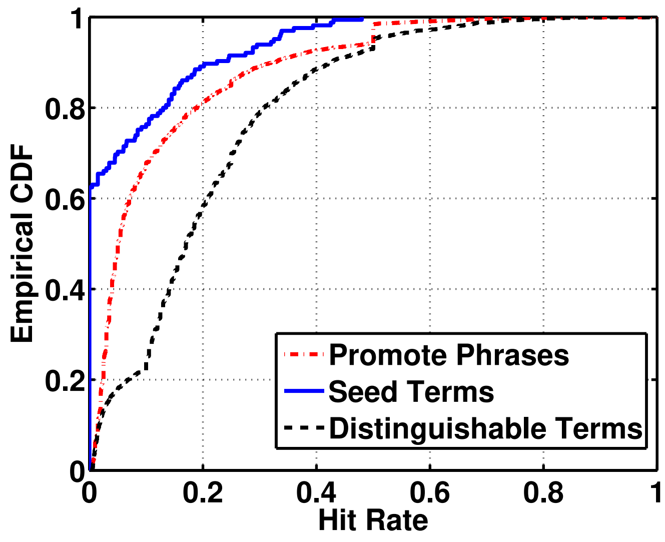


Figure 3.9: Hit rate distribution.

the top five hit rates in TermBank. As seen in Table 3.5, we can see that all these five terms' hit rates are higher than 79%. In addition, we also find that three of these five terms have the same semantic meanings of sub-phrase as "No Prescription Needed". That may be due to the reason that attackers frequently use such phrases to allure victims, because many kinds of pharmacy drugs need the prescription to buy in reality. The other two terms contain the names of two popular drugs: "Diflucan" (an anti-fungal medicine) and "Nimetazepam" (working specifically on the central nervous system).

Table 3.5: Terms with top five hit rates

Term	**Hit Rate**
Order Diflucan from United States pharmacy	90%=180/200
Buy Online No Prescription Needed	87%=174/200
Buy Cheap Lexapro Online No Prescription	85%=170/200
Online buy Atenolol without a prescription	83%=166/200
Nimetazepam interactions	79.5%=159/200

3.3.3.3 Diversity Among Different Components

In this part, we analyze the diversity of different components in terms of newly found compromised websites and campaigns.

To analyze the diversity of newly found compromised websites among different components, we essentially examine how many newly found compromised websites of each component are exclusive, which can not be found by other components. The intuition is that if one component can find the compromised websites that can not be found by another component, then these components are very complementary and they can be combined together to be effective in discovering compromised websites. To measure the diversity, we use a metric, named "Exclusive Ratio (ER)", which is the ratio of the number of compromised websites that are only found by this component to the total number of compromised websites found by this component.

As seen in Table 6.5, we can find that all four components can obtain high exclusive ratios, higher than 88%. This observation shows that all these four components are complementary and it makes perfect sense to combine them together to achieve high effectiveness in discovering new compromised websites. Also, we can find that Term Amplifier's exclusive ratio is over 99%. That is mainly because Term Amplifier can find more compromised websites through visiting more websites.

Table 3.6: Exclusive ratio of different components

Component	TermAmplifier	Inner-linkAmplifier	Outer-linkAmplifier	VulnerabilityAmplifier
Exclusive Ratio	99.56%	96.11%	89.09%	88.77%

We further examine how good each component is in terms of finding more compromised websites in *different* campaigns. We define a campaign as a set of compromised websites that share the same final or intermediate domains in their redirection chains. The intuition here is that if some components can efficiently find compromised websites in certain campaigns, and others are mainly for finding new compromised websites in different campaigns, then these components can greatly complement each other. Following this intuition, we design a metric, named "campaign ratio", which is the ratio of the number of found campaigns over the number of compromised websites. A higher ratio will indicate the component is good at finding new compromised websites in different campaigns rather than mainly finding compromised websites in the same set of campaigns.

Table 3.7: Campaign ratio of different components

Component	TermAmplifier	Inner-linkAmplifier	Outer-linkAmplifier	VulnerabilityAmplifier
Campaign Ratio	0.33%	2.88%	6.50%	7.47%

From Table 3.7, we can see that Term Amplifier and Inner-link Amplifier have relatively lower ratio than Outer-link Amplifier and Vulnerability Amplifier, which indicates that Outer-link Amplifier and Vulnerability Amplifier are better in finding compromised websites in new/diverse campaigns while Term Amplifier and Inner-link Amplifier are better in finding compromised websites in certain campaigns. Thus, these four components can work together to find more compromised websites in both known and unknown campaigns. We also find that the largest campaign in our dataset is a pharmacy related one, which contains 439 compromised websites.

3.3.3.4 Comparison With Existing Work

In this experiment, we test the performance of the pre-selected term method, which is widely used in existing work, with our dataset. Then we further compare the performance of PoisonAmplifier with two existing work: Leontiadis et al. [83] and Lu et al. [88]. They represent current methods of collecting websites compromised by search poisoning attacks, although their goals are not trying to find more compromised websites. Table 3.8 shows the comparison results.

Table 3.8: The comparison of effectiveness with existing work

Research Work	# Seed Terms	# Visited Websites	# CW[1]	Hit Rate	Time
Leontiadis et al. [83]	218	3,767,040	63,000	1.67%	9 months
Lu et al. [88]	140	500,000	1,084	0.2%	1 months
Pre-selected terms	332	64,400	252	0.39%	7 days
PoisonAmplifier	332	1,086,496	74,671	6.87%	7 days

[1] Compromised Websites

From this table, we can see that compared with pre-selected terms method, with the same number of seed terms and same time period, our work can find much more compromised websites (290 times than pre-selected terms method). This observation shows that pre-selected terms currently lead to a low hit rate, which has also been verified by [77]. To further compare with [83] (only collect compromised websites related to illicit pharmaceutical products) and [88] (collect compromised websites related to general spam), PoisonAmplifier can find more general compromised websites with highest hit rate (4 times than [83] and 34 times than [88]) within a significantly shorter period (only 7 days). Thus, our approach is more efficient and effective in discovering/collecting compromised websites, because our approach does not highly rely on the pre-selected terms, which are used by both existing work.

3.3.3.5 Comparison With Google Security Diagnostic Tools

We conduct another experiment to further evaluate the effectiveness of our approach. We want to test whether our newly found compromised websites are also detected in a timely fashion by Google's two state-of-the-art security diagnostic tools: Google Safe Browsing (GSB) [18] and Google Security Alert [15]. GSB is a widely used URL blacklist to label phishing websites and malware infected websites. Google Security Alert is another security tool, which labels compromised websites through showing the message "This site maybe compromised" within Google search results.

We first check how many compromised websites found by each component are labeled as "phishing" or "malware infected". As seen in Table 3.9, we found that GSB labels only 547 websites as "malware infected" and zero as "phishing" after examining $74,671$ newly found compromised websites. We next check how Google

Table 3.9: Labeling results by using GSB

Component	**# CW[1]**	**# Phishing**	**# Malware Infected**
TermAmplifier	69,684	0	536
Inner-linkAmplifier	2,468	0	2
Outer-linkAmplifier	2,401	0	3
VulnerabilityAmplifier	482	0	6
Total (Unique)	74,671	0	547

[1] Compromised Websites

Security Alert works on our newly found compromised websites. Specifically, we sampled 500 websites (which were randomly selected from those $74,671$ compromised websites) and manually checked them in Google. None of them were labeled as compromised.

Through the above experiments, we can find that most of our newly found compromised websites have not been detected by current Google security diagnostic tools. Although we do not argue that our approach is more effective than those two Google security tools, this observation shows that our approach can be effectively utilized to discover many new compromised websites that Google has not yet detected.

3.3.3.6 Accuracy

We collect the ground truth through comparing the difference between Searcher View and User View, which proves to be a conservative and effective approach to identify search poisoning attacks [83, 101]. To further gain more confidence, we have conducted a manual verification on 600 randomly sampled URLs from all labeled compromised websites. And all of them were manually verified as indeed compromised websites.

3.3.3.7 Constancy

To evaluate the constancy of our approach, we essentially examine whether PoisonAmplifier can continue to find new compromised websites over time. Figure 6.12(b) is the distribution of new crawled compromised websites in Stage II. We can see that during the first several days, our system can find more new compromised websites because Term Amplifier inherits a large number of terms from data S_I. With these terms, our system can efficiently find other compromised websites sharing similar terms. After that, the daily newly found compromised websites decrease quickly due to the exhaustion of terms. However, Link Amplifier and Vulnerability Amplifier can keep finding new terms and compromised websites everyday because the attackers keep promoting and attacking everyday. In this case, our system can still constantly find new compromised websites everyday leading to 26,483 newly found compromised websites during 1 month's recursive amplification process.

3.4 Discussion

In this section, we discuss possible limitations of PoisonAmplifier.

We first acknowledge that since we mainly utilize pharmacy-related keywords as

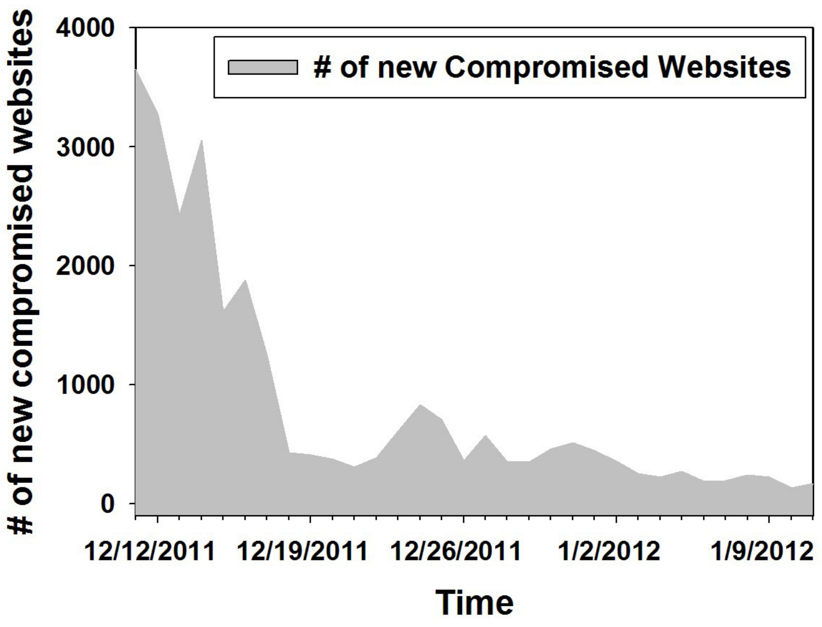


Figure 3.10: Daily new found compromised websites.

initial terms in our evaluation, this method may generate some bias. We use illicit pharmaceutical products as a specific case study to evaluate our approach mainly because it is a typical target of search poisoning attacks. However, our approach can be easily applied to other scenarios such as fake AntiVirus or web scams through changing customized keywords. In addition, through our evaluation, we can also observe that even though we use pharmacy-related keywords as initial customized keywords, those newly found compromised websites could be injected with content related to other scenarios. Thus, PoisonAmplifier can discover a broader range of compromised websites, instead of being restricted to only those used to promote illicit pharmaceutical products by attackers.

We also acknowledge that since it is still difficult for Promoted Content Extractor to accurately filter all dynamic content, this may decrease the performance (in terms of hit rate) of our approach. However, visiting websites multiple times can somehow relieve this kind of problem. In addition, we indeed manually checked several hundred randomly sampled compromised websites and we have not found such kind of false positives so far. Also, our Distinguishable n-gram Algorithm can filter some general terms (generate by dynamic content) and reduce their impacts.

In addition, we realize that once attackers know about the principle of our approach, they may try to evade our approach through providing non-malicious content to our Bot View with the utilization of IP-based cloaking techniques. For example, they may refuse to deliver malicious content if they find the IP address from our crafted Google bot crawler does not match known addresses of Google. However, as an alternative technique of Bot View by manipulating Http Referer, we can use the cache results of search engines such as Google cache as Bot View. In such way, we can obtain the Bot View of those compromised websites, as long as attackers want to make their content be crawled and indexed by popular search engines to launch search poisoning attacks. Besides, attackers may also try to decrease the effectiveness of our approach through inserting noisy content into their injected content. However, if the noisy content is general, our system will drop them based on our "Distinguishable n-gram Algorithm". Otherwise, we can still consider these noisy data as "real" promoted content as long as they are shared in multiple compromised websites.

4. SYSTEMATIC MINING ASSOCIATED SERVER HERDS INVOLVED IN MALICIOUS CYBER INFRASTRUCTURES*

We have described our first detection system,PoisonAmplifier. It only focuses on compromised servers detection. In this chapter, we present a new system, SMASH, which will focus on detecting malicious servers behind malicious cyber infrastructures.

SMASH leverages an insight that cyber criminals are increasingly using a dynamic malicious infrastructure with *multiple* servers to be efficient and anonymous in (i) malware distribution (using redirectors and exploit servers), (ii) control (using C&C servers), (iii) monetization (using payment servers), and (iv) being robust against server takedown (using multiple backups for each type of servers). As a result, in each malware campaign, there are multiple malware servers used as well as common benign servers they attacked. As illustrated in Figure 4.1, those servers are correlated, e.g., they share similar client sets. This is typically not true for benign servers because different (independent) servers usually have different sets of clients.[1]

This insight comes from an inquisition that benign servers usually serve different benign users whose behaviors might be diverse while malicious servers are set up for certain malicious clients. Not only connected by a similar set of clients, but also if these servers are the same type (e.g., both are exploit or C&C servers), they are likely to receive requests targeting the same/similar URI files (e.g., vulnerable files or exploit scripts) from malware clients. For benign servers, each server usually has lots

*Part of this chapter is reprinted with permission from "Systematic mining of associated server herds for malware campaign discovery" Jialong Zhang, Sabyasachi Saha, Guofei Gu, Sung-Ju Lee, and Marco Mellia. In *Proceedings of the 35rd International Conference on Distributed Computing Systems (ICDCS'15)*, Copyright © 2015 by IEEE

[1]Even in the case of load balancing or Content Distribution Networks where multiple benign servers are used, these servers are likely to serve different set of clients (e.g., based on their locations).

of scripts/pages and different users likely visit different pages for different purposes. On the other hand, as malicious servers are set up for certain purpose (e.g., C&C, malware downloading), it only uses certain scripts/pages to handle all their bots' requests. In addition, we observe many other correlations among malware servers in the same campaign. For example, they may have the same IP address although with many different domain names (i.e., domain-fluxing, as shown in Figure 4.1(a)), or their domains are registered by the same organization at a similar time.

Based on the above insights, instead of focusing on detecting *individual* malicious domains, we propose a complementary approach to identify a *group* of closely related servers that are involved in the same malware campaign, which we term as *Associated Server Herd (ASH)*. Our scheme, SMASH (Systematic Mining of Associated Server Herds), is designed to be deployed at enterprise or ISP networks to automatically detect malicious servers that communicate with their bot/malware armies. It uses an unsupervised community detection technique to characterize the relationship among the servers from multiple dimensions, *e.g.* if they are contacted by common clients, if the same or similar files are accessed/downloaded from them, or if they have the same Whois information, etc. Our data mining based approach exposes that often servers involved in an attack retain some similarity at multiple dimensions and we can detect such groups by combining them. Therefore, SMASH is not a real-time detection system, however, it can be run everyday to detect daily malicious activities in a large ISP/Enterprise networks or be run on a large network traffic trace to dig out previously unknown malicious activities.

SMASH detects malicious campaigns by correlating ASHs generated from multiple dimensions. Although each dimension itself might not be sufficient to distinguish malicious servers from benign servers, the combination of these dimensions can generate ASHs involved in malicious campaigns. The suspicious score of correlated ASHs is based on different combinations. The more close relationship an ASH has, the higher the probability the servers in it are involved in malicious activities

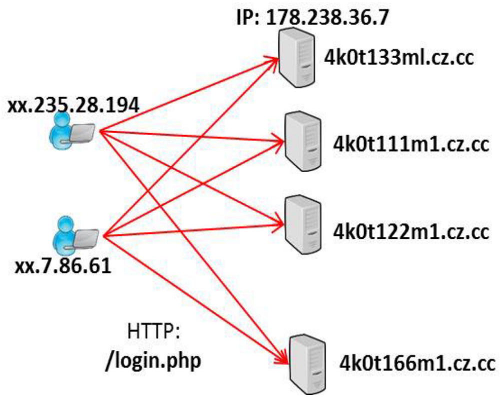
4.1 Motivation and System Goals

In this section, we first use two case studies to illustrate our motivation and then present our system goals.

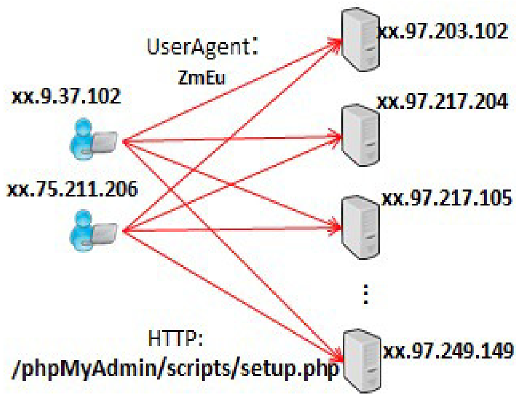
4.1.1 Case Studies

Figure 4.1 shows the example of two types of malicious activities. In the communication activity (Figure 4.1(a)), there are two clients sending HTTP requests to multiple C&C domains. Malware often uses such domain fluxing to evade detection, leading to sharing the same IP address. In addition, since these C&C servers use the same communication protocol, they utilize the same script "login.php" to handle the requests from their bots. These multiples C&C domains form a malicious communication campaign. In the attacking activity (Figure 4.1(b)), which is a ZmEu scanning campaign, there are two clients/bots that kept scanning seven benign servers targeting on "setup.php" script, which has a known code injection vulnerability. In this case, those two clients/bots scan the default path of phpMyAdmin for the exploitation leading to sharing the same file "setup.php". Those seven targeted servers form a malicious attacking campaign.

We can see that both types of malicious activities share very similar server-side properties: those servers are correlated by sharing similar clients and the URI path (e.g., "login.php" for the communication activity and "setup.php" for the attacking activity).



(a) Communication activity



(b) Attacking activity

Figure 4.1: Malicious campaigns examples.

4.1.2 Goals

Motivated by the above example, we proposed a novel framework to detect a group of closed related servers that are potentially involved in the same malware campaigns. Different from existing work that relies on signatures [90], client side behavior patterns [71], or supervised learning of individual server reputation [63], our framework utilizes an *unsupervised* approach that focuses on *server side* communication patterns and does *not* rely on signatures. Therefore, our approach can detect zero-day malware campaigns. In addition, since we study the group behaviors of malware servers rather than a single server, we can provide more complete views of malicious campaigns than existing work focusing on single server detection.

4.2 System Design

4.2.1 System Overview

The primary goal of SMASH is to detect suspicious correlated servers that are involved in malicious activities by passively looking at the network-wide HTTP communications. Such malicious activities include launching HTTP attacks on benign servers and communicating with malicious servers through the HTTP channel. Instead of detecting each server in isolation, we study the *different relationship* among all the servers involved in similar activities. Those servers involved in the same malicious activity are inferred as a malicious campaign by SMASH.

Figure 6.8 depicts the architecture of SMASH. The system takes HTTP network traffic as input, and has five components: traffic preprocessing, ASH mining, ASH correlation, pruning, and malicious campaign inference.

4.2.2 Preprocessing

The goal of preprocessing is to reduce the traffic that needs to be processed by SMASH. We explore two steps to reduce the number of input servers to SMASH. First, we assume that domains with the same second-level domain belong to the same organization.[2] For example, a.xyx.com.cn and b.xyz.com.cn both belong to xyz.com.cn, thus there is no need to differentiate them. Some CDN/Cloud servers will be also aggregated as one server in this case. For example, all the Facebook CDN servers will be aggregated as “fbcdn.net”. Amazon cloud servers will be aggregated as “amazonaws.com”. The aggregation of all domains based on their second-level domains leads to 60% reduction of all servers.

We further remove most benign servers based on their popularity.[3] To measure the “popularity” of servers, we utilize the concept of inverse document frequency (IDF), which is a measure of whether the term is common across all documents in information retrieval. In our case, we try to remove common servers across all the clients’ requests. We define the popularity of a server as the number of clients that

[2]There are some exceptions such as cloud servers, dynamic DNS. We will discuss them in Section 6.5.

[3]We acknowledge that we may miss some compromised popular domains. However, we argue that this represents a necessary tradeoff between performance and accuracy. In reality, most popular servers have resources and incentives to secure their websites and thus have a lower possibility to be compromised than less popular ones.

communicated with the server. The more clients the server is connected to, the more popular the server is.

We select an IDF threshold of 200 based on the popularity distribution of IDS confirmed malicious servers, which filters very popular servers but still keeps 99% of the servers. After the preprocessing process, we reduced 58.6% of traffic in our dataset.

4.2.3 Associated Server Herd Mining

The goal of ASH mining is to find closely related servers that are involved in the same malware campaign. We define one main dimension and three secondary dimensions to characterize the relationship among the servers, and systematically mine ASHs. ASH generated from each dimension itself might not be sufficient to distinguish the malicious group of servers from benign servers, but ASHs associated with the combination of these dimensions are more likely to generate server groups involved in malicious campaigns.

To find the correlated servers, a simple way is to assign each server with a feature vector and perform clustering on this multi-dimension feature vector. However, the dimensions are different in nature and it is inefficient to combine them to evaluate similarity. Also, as we show in Section 4.4.3, it is hard to assign a unique weight for each dimension because different malicious campaigns rely on different combinations of those dimensions. We observe that malicious servers in the same malicious campaign usually share a very similar (if not the same) set of malware clients while those malicious servers are usually not connected by benign clients. Thus, we use client similarity as the main dimension, which is much more robust against manipulation from attackers than other dimensions, and can reliably group the servers.

We see that although client similarity alone may not directly distinguish malicious servers from benign servers, it separates benign server groups from malicious server groups. Thus, the main dimension must be satisfied for all campaigns.

Each secondary dimension characterizes the relationship among different servers from a certain perspective. We evaluate how their combinations can be used to infer malicious campaigns in Section 4.4.3. Note that we envision SMASH, as an extensible system, can easily incorporate new dimensions. For example, to keep our system lightweight, we have not included all payload downloaded from each host. However, this can be an interesting dimension to consider and can be easily added as another dimension.

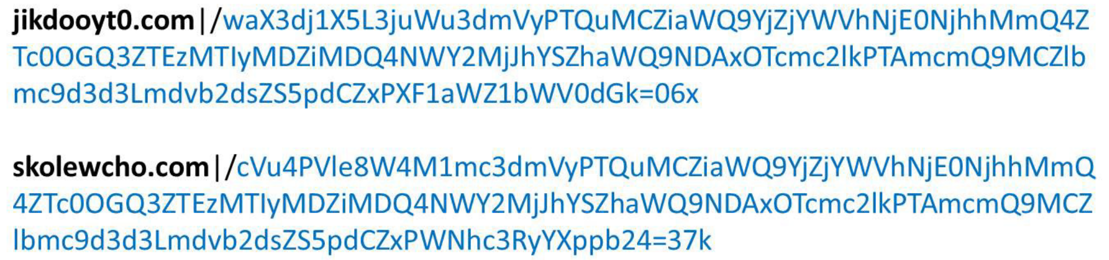
4.2.3.1 Main Dimension

We use *client similarity* as the main dimension. Client similarity between two servers depends on the common set of clients contacting them. We define the client similarity between servers S_i and S_j as:

$$Client(S_i, S_j) = \frac{|C_{s_i} \bigcap C_{s_j}|}{|C_{s_i}|} * \frac{|C_{s_j} \bigcap C_{s_i}|}{|C_{s_j}|} \tag{4.1}$$

where C_{S_i} denotes the set of clients contacting the server S_i. The ratio $\frac{|C_{S_i} \bigcap C_{S_j}|}{|C_{S_i}|}$ represents the importance of the common clients for server S_i. The intuition here is that if two servers with many clients are similar, there will be a large overlap between their clients. Thus, two servers are similar when their common clients are important to both servers.

Since malicious servers are usually not connected by benign clients while infected clients are usually connected to the same set of suspicious servers, two servers sharing similar sets of client connections should belong to the same ASH. Specifically, we build a communication graph $G = (V, E)$, where V denotes all the servers and each edge $(i, j) \in E$ denotes that servers i and j share a set of clients. The weight of the



```
jikdooyt0.com|/waX3dj1X5L3juWu3dmVyPTQuMCZiaWQ9YjZjYWVhNjE0NjhhMmQ4Z
Tc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZhaWQ9NDAxOTcmc2lkPTAmcmQ9MCZlb
mc9d3d3Lmdvb2dsZS5pdCZxPXF1aWZ1bWV0dGk=06x

skolewcho.com|/cVu4PVle8W4M1mc3dmVyPTQuMCZiaWQ9YjZjYWVhNjE0NjhhMmQ
4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZhaWQ9NDAxOTcmc2lkPTAmcmQ9MCZ
lbmc9d3d3Lmdvb2dsZS5pdCZxPWNhc3RyYXppb24=37k
```

Figure 4.2: Obfuscated filenames.

edges reflects the strength of similarity between the two servers in terms of client similarity.

To extract ASH from G, we adopt a graph based clustering algorithm [64] that is designed to efficiently uncover communities in large networks. It uses modularity to measure the quality of extracted community, which is a scalar value between -1 and 1, and represents the density of the links inside the community as compared with the links between communities. It automatically finds high modularity partitions of large networks in short time. The nodes that are still connected to each other after this process form ASHs of the main dimension.

4.2.3.2 Secondary Dimensions

We present our current secondary dimensions.

URI File Similarity: We study the relationship among servers based on URI files as servers in the same malicious activities might share similar/same URI files. For example, web attacks target certain vulnerable files, and thus different targeted servers share the same destination files. Different C&C servers in the same campaign may use the same scripts to handle the requests from the infected clients, and hence they might also share the same files. We extract all the URI files of the servers by checking the HTTP requests. Here we focus on URI files rather than the whole URI path because in attacking activities, some benign servers share the same vulnerable file but have different paths due to the different configurations on each web server.

We define a URI file as the substring of a URI starting from the last '/' until the end before the question mark, which usually is the file or script used for handling clients' requests. As shown in Figure 4.2, sometimes attackers use obfuscated filenames for different malicious servers that are involved in the same malicious campaign. We define URI file similarity between the two files as follows. If the length of the filename is shorter than or equal to *len*, we define the similarity function of files f_i and f_j as:

$$sim(f_i, f_j) = \begin{cases} 1 & if f_i = f_j, \qquad (4.2) \\ 0 & otherwise. \qquad (4.3) \end{cases}$$

Thus, two files are similar if they are exactly the same, since short filenames are usually not obfuscated. However, if the length of a filename is longer than len[4], we define the similarity function as:

$$sim(f_i, f_j) = \begin{cases} 1 & if \cos(\theta) > 0.8 \qquad (4.4) \\ 0 & otherwise \qquad (4.5) \end{cases}$$

where

$$\cos(\theta) = \frac{CharSet_{f_i} \cdot Charset_{f_j}}{\|CharSet_{f_i}\| \cdot \|CharSet_{f_j}\|}. \qquad (4.6)$$

where $CharSet_{f_i}$ is the character distribution vector of file name f_i. Thus, for long filenames, we check the characters frequency distribution ($CharSet$ in Eq. (6.3)) of the filenames. Two filenames are similar as long as their names have similar character distributions. For the exact same filenames, the similarity score is 1. While our

[4]*len* is empirically set to 25 in this chapter, which is based on the length distribution of the filenames whose servers have been labeled by IDS.

similarity function works well in our evaluation, it can be replaced by any similarity functions such as Levenshtein distance and Hamming Distance.

We now define the file similarity between the two servers S_i and S_j as

$$File(S_i, S_j) = \frac{\sum_m max_n(sim(f_{S_{im}}, f_{S_{jn}}))}{\sum_m 1} * \frac{\sum_n max_m(sim(f_{S_{jn}}, f_{S_{im}}))}{\sum_n 1} \tag{4.7}$$

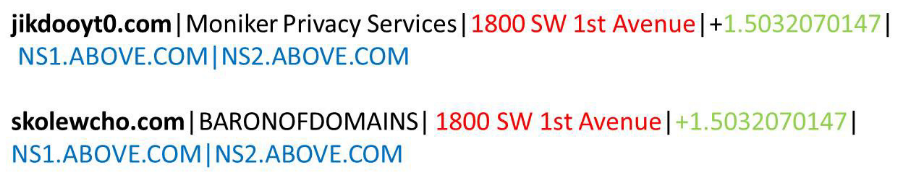
where $f_{S_{im}}$ is the m-th file from server S_i. Similar to client similarity, the first term of the right hand side of Eq. (4.7) reflects the importance of similar files to server S_i, and the second term of the right hand side of the equation reflects the importance of similar files to server S_j. Thus, if two servers share enough similar files, they might be involved in the same activities, and should be in the same ASH.

IP Address Set Similarity: We investigate the relationship among the servers based on their IP addresses because malicious domains may share a similar set of IP addresses. For example, malicious servers may use fast flux to evade domain based detection, and thus multiple domains may share the same IP address. In our dataset, skolewcho.com, switcho81.com, jikdooty0.com and swltch081.com all used the same IP address. Similar to client similarity, we define IP address set similarity as:

$$IP(S_i, S_j) = \frac{|I_{S_i} \bigcap I_{S_j}|}{|I_{S_i}|} * \frac{|I_{S_j} \bigcap I_{S_i}|}{|I_{S_j}|} \tag{4.8}$$

where I_{S_i} is the set of IPs that server S_i is associated with. Thus, if two servers share similar IP addresses, they might be involved in the same activities and should be in the same ASH.

Whois Similarity: We study the relationship among servers based on their whois information as malicious servers may be registered with similar information, such as register name, home address, email address, phone number, and name servers. Figure 4.3 shows the whois information of two example malicious servers. Although


jikdooyt0.com|Moniker Privacy Services|1800 SW 1st Avenue|+1.5032070147|
NS1.ABOVE.COM|NS2.ABOVE.COM

skolewcho.com|BARONOFDOMAINS| 1800 SW 1st Avenue|+1.5032070147|
NS1.ABOVE.COM|NS2.ABOVE.COM

Figure 4.3: Whois similarity.

they have different registrants, they share the same home address, phone number, and name servers. We use the whois similarity to measure the relationship among servers, which is defined as the number of shared fields between the two servers over the union of fields. We require that the two servers share at least two above mentioned fields to be considered as associated servers to avoid the case that two servers only share the domain name registration proxy.

4.2.3.3 ASH Generation

After studying the similarity among servers from different multiple dimensions, we build similarity graphs for different dimensions, and use the same graph based community detection algorithm mentioned in Section 4.2.3.1 to generate ASHs. The nodes connected to each other after this process form ASHs for each dimension.

4.2.4 Associated Server Herd Correlation

Once we obtain the ASHs from different dimensions, we perform ASH correlation. The goal of multi-dimension correlation is to distinguish malicious ASHs from benign ASHs. To achieve this, we consider ASHs in different dimensions and extract their common associated servers to form new ASHs. Ideally, the more dimensions an ASH belongs to, the more likely it is involved in malicious activities. The intersection of ASHs between the main dimension and secondary dimensions forms the new suspicious ASHs.

For example, $(ASH_j^d \bigcap ASH_i^m)$ forms a new ASH combining ASH_i^m from the main dimension m and ASH_j^d from a secondary dimension d. We compute the suspicious score for each server in the new ASH as follows:

$$S(S_i) = \sum_{d \in Sec\_Dimensions} w_d(C_{S_i}^d) w_m(C_{S_i}^m) \Phi(|C_{S_i}^d \bigcap C_{S_i}^m|) \tag{4.9}$$

where $\Phi(x) = \frac{1}{2}(1 + erf(\frac{x-\mu}{\gamma}))$, $erf(\cdot)$ is the "S" shaped Gaussian error function, and μ and γ are user specified parameters.[5] $C_{S_i}^m$ is an ASH from dimension m that includes server S_i and $w_d(C_{S_i}^d)$ represents the ASH density. Density is measured as the number of edges $|e|$ in one group over the number of edges in the fully connected graph with $|v|$ vertices in that group $(2 * |e|/(|v| * (|v| - 1))$. The intuition here is that the denser a group is, the more likely it belongs to a malicious group, as benign servers are less likely to be well connected. When we obtain the suspicious score for each server in the newly formed ASH, the servers whose scores are below the threshold *thresh* are removed. In addition, the groups with only one server left are also removed because that server can not be associated with others. We discuss the selection of *thresh* value in Section 4.4.

In Eq. (4.9), $\Phi(ASH_i^m \bigcap ASH_j^d)$ measures the suspiciousness of the newly formed ASH, created with the servers common in two ASHs formed based on dimensions m and d. It is based on the size of the ASH and promotes the ASHs with a large number of servers. A smaller value of $|ASH_i^m \bigcap ASH_j^d|$ means that there are only a few servers in the ASH, and we have less confidence in its maliciousness. Hence we need to cross check with more dimensions to make a decision.

The "S" shaped $\Phi()$ normalizes $\Phi(ASH_i^m \bigcap ASH_j^d)$ into a value between 0 and 1. After the normalization, a group with less than four servers receives a low score,

[5]We empirically set $\mu = 4$ and $\gamma = 5.5$. Choice of μ promotes the clusters with size larger than 4. γ determines the desired steepness of the curve.

and need to be cross checked with more dimensions to accumulate higher suspicious scores. For each dimension, the highest score is 1. In this case, the correlation score reflects how the ASHs are formed. Suspiciousness score $S(.)$ of each server then accumulates scores from all ASHs it belongs to. The higher the score, the more suspicious the server is. If a server has suspiciousness score below *thresh*, it gets removed from all the ASHs. After the removal, the ASHs (created by combining multiple dimensions) are left with only servers with high scores. For example, a score higher than 1.0 means that the server is inferred through one main dimension and at least two secondary dimensions. We then call the ASHs with high scoring servers as suspicious.

4.2.5 Pruning

After the correlation, we define and prune two types of noisy ASHs: (i) Redirection Group and (ii) Referrer Group. For the redirection group, some servers are associated with each other because they belong to the same redirection chain. Hence they share exactly the same sets of clients, IP addresses, and sometimes URI files. For the referrer group, some servers are associated with each other because they are referred by the same landing server (e.g, landing websites are embedded with other websites).

To remove these noisy servers without eliminating malicious servers, we use their landing servers to replace all the servers in the same redirection chain instead of simply dropping those groups, if all the servers in the chain share same IP addresses, URI files or Whois information. Similar to the redirection group, we also use landing servers to replace all the referred servers. The intuition here is that for the redirection and the referrer groups, if a client visits the landing server, it *automatically* visits other servers in the redirection chain or the embedded servers. We, therefore, use only the landing server to represent those servers.

We collect the redirection chains by sending an HTTP request to each server in the ASHs, and obtain referrer information by extracting the HTTP "referrer" field from the input network traffic. After the pruning process, if there still exist more than one server in the ASH, we keep that group as a candidate malicious ASH.

4.2.6 Malicious Campaign Inference

ASH correlation process typically captures specific malicious activities, but not the whole malicious activities. For example, bots first download encrypted files from some servers and then connect to other C&C servers. In this case, ASH correlation process might separate these two processes into two different herds, making it difficult to analyze the file downloading activities. Towards this end, we apply a refinement step in which we rebuild the original attack campaign based on the client similarity. Two malicious ASHs are merged when their servers are in the same herd for the main dimension, i.e., they share a very similar set of clients. The intuition is that the main dimension captures the group connection behaviors of malicious activities, and the infected clients that connect to different files or IPs could still belong to the same malicious campaign.

4.3 Evaluation Data

We now describe the details of the network dataset and the ground truth used to evaluate SMASH.

4.3.1 Network Trace

To evaluate our system, we experiment with real network traffic traces collected at the edges of a large ISP. We monitored all the incoming and outgoing traffic in the network. The monitored users were mostly residential, and connected to the

Table 4.1: ISP network traffic statistics.

	$Data_{2011day}$	$Data_{2012day}$	$Data_{2012week}$
# of clients	14,649	18,354	28,285
# of HTTP requests	28,544,473	40,522,026	168,726,091
# of Servers	92,517	117,507	354,578
# of URI Files	1,521,249	2,936,082	12,698,176

Internet via high-end ADSL links. Our traces are PCAP files and for every TCP connection and UDP flow, we collected the first 5000 bytes, including the IP addresses and domain names of the destination servers. Table 6.4 presents the information of the ISP traffic we collected at different times: one day data from October 2011 ($Data_{2011day}$), one-day data from August 2012 ($Data_{2012day}$), and one-week data from October 2012 ($Data_{2012week}$). We choose data from different periods to evaluate the performance of SMASH over time.

4.3.2 Ground Truth

To estimate the false positives and negatives of our inference results, we use the following data sources as the ground truth.

4.3.2.1 Intrusion Detection System (IDS)

We used a well-known commercial IDS with signatures to label malicious flows with corresponding threat identifiers. Note that since IDS signatures are constantly updated, we used two versions of the same IDS, one from early 2012 and the other from June 2013. We run all the collected network traces through both IDS versions and generate two ground truth datasets: the servers (IDS_{2012}) labeled by the 2012 IDS signatures and the servers (IDS_{2013}) labeled by the 2013 signatures but **not** in IDS_{2012}.

Table 4.2: Number of malicious campaigns.

	$Data_{2011day}$				$Data_{2012day}$			
Infer Thresh.	0.5	**0.8**	1.0	1.5	0.5	**0.8**	1.0	1.5
SMASH	34	**17**	11	6	38	**19**	12	2
IDS_2012_total	1	**1**	1	0	0	**0**	0	0
IDS_2013_total	0	**0**	0	0	0	**0**	0	0
IDS_2012_partial	4	**3**	3	0	0	**0**	0	0
IDS_2013_partial	1	**0**	0	0	2	**0**	0	0
Blacklist_partial	16	**10**	5	6	13	**12**	7	1
Suspicious	4	**0**	0	0	9	**2**	1	0
False Positives	8	**3**	2	0	14	**5**	4	1
FP (Updated)	4	**1**	1	0	7	**1**	1	0

4.3.2.2 Online Blacklist

We also check our inferred results with popular blacklists, including Malware Domain Block List [27], Malware Domain List [28], Phishtank [37], SpyEye Tracker [43], ZeuS Tracker [58] and online services such as Virustotal [51], Web of Trust (WOT) [56] and WhatIsMyIPAddress [4]. If a server is listed as malicious by any of these blacklists, except WhatISMyIPAddress, we confirm it as a malicious server. As for WhatIsMyIPAddress, which integrates results from 78 blacklist services, we require a malicious report from at least two blacklists to confirm as a malicious server.

4.4 Evaluation Results

4.4.1 Inference Results

4.4.1.1 Number of Malicious Campaigns

We first evaluate inference results in terms of malicious campaigns.[6] Table 4.2 reports the number of malicious campaigns the SMASH inferred with different *thresh* we described in Section 4.2.4. For those inferred campaigns, we verify them with our ground truth. If all the servers of a campaign are confirmed by the IDS, we term it as

[6]Due to the page limit, here we only discuss the campaigns that have at least two involved clients.

"IDS_2012/2013_total." If only a subset of the servers in a campaign is confirmed by the IDS, we term it as "IDS_2012/2013_partial." If none of the servers of a campaign are confirmed by IDS but confirmed by online blacklist, we term it as "Blacklist." For the campaigns that can not be confirmed by either IDS or Blacklist, we further check the HTTP request status code of those servers from the network traffic, and send the HTTP requests to verify the existence of those servers.[7] If at least half of the servers in a campaign have an error code in their network traffic or do not exist any more, we consider this as a "suspicious" campaign. All other campaigns are considered as false positives. Note that there may exist malicious campaigns that are labeled as false positives because we do not have enough information to confirm them. Thus, the false positives here should be an upper bound for our system.

For $Data_{2011day}$ with threshold 0.8, SMASH infers 17 malicious campaigns. Among these, one campaign has all the servers confirmed by 2012 IDS signatures. There are three campaigns where some of their servers are confirmed by 2012 IDS signatures. There are ten campaigns that have their servers partially detected by blacklists. Three campaigns are false positives. If we reduce the threshold to 0.5, SMASH identifies 34 malicious campaigns but the false positives increase to eight. On the other hand, if we increase the threshold to 1.0 and then to 1.5, SMASH detects 11 and 6 campaigns, but with two and zero false positives, respectively. Similar results are observed from $Data_{2012day}$.

Further analyzing the false positives, we discovered two major categories of false positives: Torrent and TeamViewer [45], a remote online collaboration tool. For the Torrent category, several P2P clients connect to a large number of torrent servers by only requesting "scrape.php" files. Thus, they share at least the same filename and

[7]We only check the existence of those domains. Our intuition is that malicious domains usually have a short lifetime, and thus might have expired while benign domains usually have a longer lifetime.

sometimes the same IP addresses. For TeamViewer, it has a large pool of servers that are used by their clients to retrieve their ID, which leads to sharing the same path name among those servers. By removing the false positives of these two "noisy" campaigns, we have very few false positives as shown in the last row (FP Updated) of Table 4.2.

4.4.1.2 Number of Servers in Malicious Campaigns

Table 4.3 shows the inference results of the number of servers involved in malicious activities. Similar to the malicious campaign, if a server is confirmed by the 2012 IDS signatures, we term it as IDS_2012, and if a server is confirmed by the 2013 IDS signatures but **not** by the 2012 IDS signatures, we term it as IDS_2013. For those servers that are not confirmed by either IDS signatures but confirmed by the blacklist, we term it as "Blacklist." All the servers in "suspicious" attack campaigns (as described in Section 4.4.1.1) are inferred as "suspicious". For the remaining servers, we compare them with IDS and Blacklist confirmed servers in terms of the requested path, User-Agent, and parameter patterns. Servers confirmed through this way are termed as "New Servers", which are previously undetected servers. All other servers are false positives.

For $Data_{2011day}$ with threshold 0.8, SMASH infers 3,156 servers that are involved in malicious campaigns. Among these servers, only 20 are labeled by IDS signatures and 401 are confirmed by the blacklist. Our system can infer 2,701 more servers which is nearly 7 times the servers detected by IDS and blacklists combined. There are 34 false positives and only 16 after excluding the P2P and TeamViewer cases. We can also see that we generate fewer false positives with higher thresholds. For threshold 1.5, there were no false positives for either $Data_{2011day}$ or $Data_{2012day}$. However, this comes at a price of missing many attack campaigns. We, therefore, select 0.8 as the

Table 4.3: Number of servers in malicious activities.

	$Data_{2011day}$				$Data_{2012day}$			
Infer Thresh.	0.5	**0.8**	1.0	1.5	0.5	**0.8**	1.0	1.5
SMASH	3,222	**3,156**	3,039	845	407	**287**	150	9
IDS_2012	20	**19**	19	0	0	**0**	0	0
IDS_2013	2	**1**	1	0	3	**0**	0	0
Blacklist	413	**401**	389	74	67	**55**	29	2
New Servers	2,713	**2,701**	2,626	771	171	**152**	91	0
Suspicious	13	**0**	0	0	27	**5**	2	0
False Positives	61	**34**	4	0	139	**75**	28	7
FP (Updated)	22	**16**	2	0	32	**5**	2	0

threshold, where we detect many attack campaigns while the highest false positive rate is only 0.064%. After removing noise, the largest false positive rate is 0.017%.

We see that SMASH discovers many malicious servers that are not discovered by IDS and blacklist. In Table 4.2 with threshold 0.8 for $Data_{2011day}$, 13 clusters are partially detected by IDS or blacklist. Among these clusters, IDS detects only 20 servers and blacklists detect 401 servers. On the other hand, SMASH inferred 2,701 servers that are new, previously unknown malicious servers. Those servers either share the similar pattern with IDS confirmed servers in terms of User-Agent, parameter patterns and URI files, etc or are detected by other researchers based on the Google search results. This indicates that about 86.5% of these malicious servers could not be detected by simply relying on IDS or blacklists.

Furthermore, we see from Table 4.3 that without any training or signature updating, SMASH infers malicious severs that are detected by new IDS signatures but missed by old IDS signatures. This shows that SMASH can detect zero-day malicious campaigns before IDS signatures get updated.

Finally, we measure the malware servers detected by the IDS but missed by SMASH, i.e., false negatives. To get the ground truth of malware server groups from IDS labels, we group the IDS-labeled malicious servers based on the IDS threat

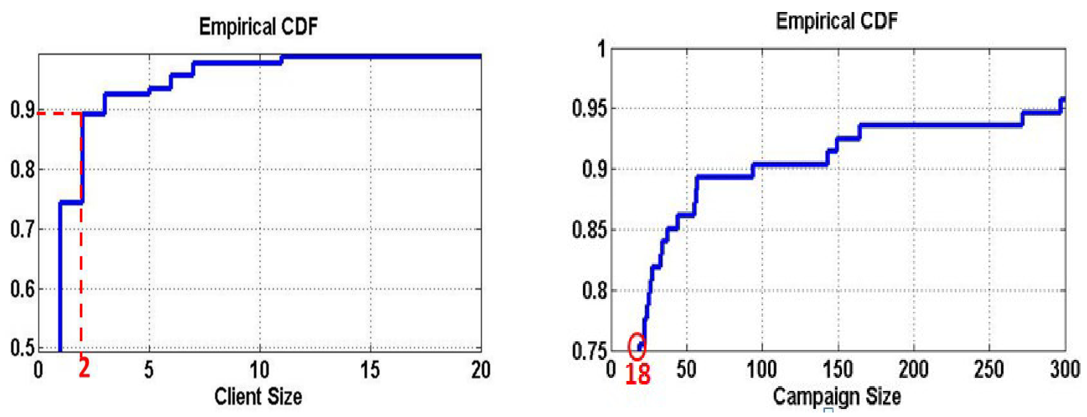


Figure 4.4: Distribution of the client and campaign sizes.

identifier, assuming all the servers in the same threat identifier belong to the same malicious campaign. We have a total of 26 missed malware servers for $Data_{2011day}$ and 27 for $Data_{2012day}$.

There are two major types of false negatives. First, there are 40 malicious servers (in the Cycbot, Fake AV, and Tidserv threat labels) that do not share any secondary dimension, thus are missed by our system. However, most of those servers share the same URI parameters pattern. Thus, if we extend our URI file dimension to consider the parameter pattern, we could detect these threats. Second, several false negatives are caused by our pruning/filtering process because these servers share the same referrer. SMASH has room for improvement and nevertheless, it can be a great complementary tool to existing approaches.

4.4.1.3 Activity Scale of Malware Campaigns

We measure the scale of malware campaigns by observing the number of clients and servers involved in each malicious campaign. Figure 4.4 presents the distribution of the campaign size and the client size. We see that about 75% of the attack campaigns have the size smaller than 18, which indicates that most attack campaigns

Table 4.4: Attack categories.

Activity	Category	# of Servers
Communication	C&C	30
	Web exploit	1
	Phishing	5
	Drop zone	2
	Other malicious servers	1,120
Attacking	Web scanner	23
	iFrame injection	14

do not connect to a large number of malicious servers. However, campaigns with a size larger than 18 are usually attacking campaigns, which attack a large number of benign servers (e.g, web scanning and iFrame injection attacks). As for the number of involved clients, 75% of attack campaigns have only one infected client. This result suggests that most client-side clustering systems [71, 72] might be ineffective because they need to correlate among multiple infected clients in the same network.

4.4.2 Attack Diversity & Persistency

To demonstrate that SMASH is not limited to only certain types of malicious campaigns, we evaluate SMASH from two different perspectives: attack categories and persistence of servers involved in malicious activities.

4.4.2.1 SMASH *infers diverse malicious campaigns*

Our inference results include the attack campaigns related to malicious communication activities and web attacking activities. Table 6.5 categorizes part of our inferred servers involved in malicious campaigns based on IDS labels and Online Blacklists. The servers belonging to communication activities are typically malicious servers, such as those involved in botnet activities and web exploits. The servers belonging to attacking activities are usually benign websites that are targeted by

Table 4.5: Number of malicious campaigns during $Data_{2012week}$.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
SMASH	31	36	51	40	34	47	51
IDS_2013_total	1	1	1	0	1	1	1
IDS_2013_partial	3	5	7	4	3	8	8
Blacklist	14	19	28	19	16	18	25
Suspicious	4	5	3	4	3	4	6
False Positives	9	6	12	13	11	16	11
FP (Updated)	3	3	11	9	6	12	9

Table 4.6: Number of servers involved in malicious activities during $Data_{2012week}$.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
SMASH	1023	1246	1481	1157	911	1286	1301
IDS_2013	7	15	27	11	7	13	13
Blacklist	371	645	726	348	253	354	698
New Servers	467	398	586	668	443	737	497
Suspicious	13	19	8	18	10	8	21
False Positives	165	169	134	122	198	174	72
FP (Updated)	82	14	130	36	24	89	52

malware, such as web scanning and iFrame injection.

4.4.2.2 SMASH *infers both persistent and agile malicious campaigns*

Persistent malicious campaigns are a set of servers that continue to communicate with infected clients for multiple days. On the other hand, agile malicious campaigns are a set of newly identified servers that are communicated by known infected clients. To study the evolution of persistent and agile malicious campaigns, we test our system with one-week data from 2012, $Data_{2012week}$. Tables 4.5 and 4.6 show the inference results.[8] We consider the first day of the week data as the benchmark. Figure 4.5 shows the results of each day, where there are 1,014 servers and 27 clients involved in malicious activities at the benchmark day. We see that SMASH infers both persistent (Old_Server in Figure 4.5) and agile malicious campaigns

[8]For the campaign with one client, we use threshold 1.0 while for the campaign with more than one client, we use threshold 0.8.

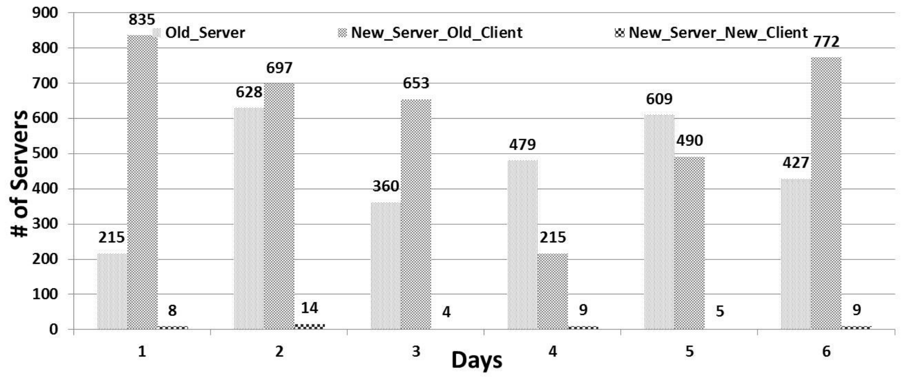


Figure 4.5: Persistent vs dynamic campaigns.

(New_Server_Old_Client). It can also infer new campaigns (New_Server_New_Client in Figure 4.5). In addition, we observe that most servers belong to agile malicious campaigns. This result suggests that malware may change their servers/domains every day to evade existing domain-based detection.

4.4.3 Effectiveness of the Main and Secondary Dimensions

4.4.3.1 Main Dimension

24,964 servers in $Data_{2011day}$ and 33,603 servers in $Data_{2012day}$ are dropped after the main dimension processing because they can not be correlated with other servers in client similarity. For those remaining servers, we further investigate the relationship among those servers in the same ASH.[9] To do this, we manually study 50 randomly chosen campaigns from each day. 60% of ASHs are "referrer groups," in which all servers in the same group are referred by the same server. Such groups can be further filtered by the pruning process. 10% of ASHs are "redirection groups," in which all servers in the same groups belong to a redirection chain. Such groups can be also filtered by the pruning process. 8% of ASHs are "similar content groups,"

[9]Here, we ignore ASH with only one client, as all the servers in this case are correlated together only because they are visited by one client.

in which all servers share very similar content. We further analyze those servers and most of them belong to adult web servers. 18% of ASHs are "unknown groups," in which we can not directly find any relationship among those servers. However, they are visited by similar sets of clients. Most of these servers are different companies selling different products or services. The remaining 4% of ASHs belong to malicious ASHs. None of these servers is detected by the IDS while SMASH did.

4.4.3.2 Secondary Dimensions

In SMASH, the ASHs need not satisfy all secondary dimensions. Hence, we measure the effectiveness of each secondary dimension. Figure 4.6 is the decomposition of inferred servers. We see that 15.05% of the servers satisfy all secondary dimensions and there is no false positive for these herds. URI File dimension is the most effective secondary dimension, which by itself contributes to 53.71% of the detected servers. Although Whois and IP Address Set dimensions individually are not very effective, these two dimensions can help the URI File dimension to confirm more suspicious ASHs. For example, 14.16% are inferred through the combination of IP Address Set and URI File dimensions, and 17.01% are inferred through URI File and Whois dimension combination.

4.4.4 Attack Campaign Case Study

We investigate with case studies the advantages of SMASH over detecting each malicious server in isolation.

4.4.4.1 Capturing the Insight of Malicious Activities

ASHs help us understand the malicious campaign in a holistic fashion. Table 4.7 shows the Bagle botnet [3], which is a mass-mailing computer worm campaign that SMASH inferred. In this campaign, the bot first goes to some servers to download an

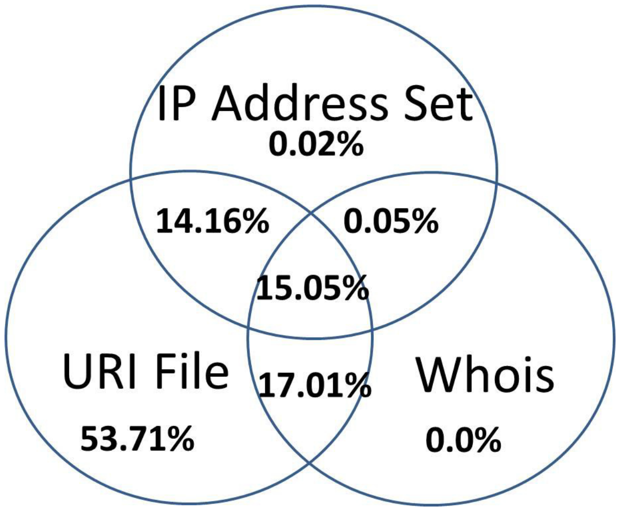


Figure 4.6: Effectiveness of secondary dimensions.

Table 4.7: Bagle botnet.

Categories	Servers	URI	UserAgent	Parameters
C&C Domain	novitacolori.it	/images/news.php	Internet Exploder	p=16435&id=21799517&e=0
	beachrugbyfestival.com	/images/news.php	Internet Exploder	p=16435&id=21799517&e=0
	beautywoman.sk	/images/news.php	Internet Exploder	p=16435&id=21799517&e=0
	...	...	...	...
Downloading	lajuve.org	/images/file.txt	Mozilla/4.0 ...	na
	shayestegansch.com	/images/file.txt	Mozilla/4.0 ...	na
	www.bigdaybreaker.com	/images/file.txt	Mozilla/4.0 ...	na
	...	...	...	...

encrypted file “file.txt” and then connects to C&C servers by requesting “new.php” with the same parameter pattern “p=[]&id=[]&e=[]”. There are 94 servers involved in this campaign and they can be clustered in two categories; 40 downloading servers and 54 C&C servers. None of the downloading servers was detected by the IDS or blacklists. Only three C&C servers were detected by VirusTotal. Without the holistic approach of SMASH, we would not have captured the downloading servers and many additional C&C servers.

Table 4.8 shows a Sality botnet [41] campaign also inferred by SMASH. There are 12 servers involved in this campaign. All have been labeled by the IDS but only eight have been labeled by the blacklists. Again, we cluster the servers in this

Table 4.8: Sality botnet.

Categories	Servers	URI	UserAgent	Parameters
C&C Domain	kukutrustnet777.info	/	KUKU v5.05exp =22667130988	22adcdc=72726968
	kjwre9fqwieluoi.info	/	KUKU v5.05exp =22667130988	e65564=135856260
Downloading	merc-connect.com	/images/mainf.gif	KUKU v5.05exp =22667130988	8fff57=84933135
	meta-kit.com	/images/logos.gif	KUKU v5.05exp =22667130988	4f152d=10365530
	fashionenigma.com	/images/logos.gif	KUKU v5.05exp =22667130988	6f2483=58270744
	...	...	...	...

campaign into C&C servers and downloading servers. Two C&C servers are inferred because they share the same set of IP addresses, the same filename "/" and the same registration information; thus they form a strong ASH. Downloading servers form different ASHs based on the shared filenames. Different from the Bagle botnet, only downloading servers are benign websites compromised by the attackers. Thus, they do not share IP addresses or Whois information. The Sality botnet campaign is inferred by merging these ASHs. The bots might first go to the compromised servers to download additional malware through requesting ".gif" files. They then go to C&C servers to get further instructions.

Based on above two examples, most of the downloading servers and some C&C servers from the Bagle botnet are compromised websites. Therefore, domain-reputation-based systems [63] or similarity-based detection systems [61] would not detect such malicious servers.

4.4.4.2 Finding More Malicious Activities

Table 4.9 shows a web injection attack campaign, where infected hosts inject malicious iFrames to benign websites. SMASH inferred 600 benign servers suffering from such attacks while the IDS labeled only four of such attacks, missing more than 99% of the servers. All of these inferred servers are queried by the same set of clients with the same file "sm3.php" under different paths. The servers not labeled by the IDS share the same UserAgent "-" in their HTTP requests to the IDS labeled

Table 4.9: iFrame injection attack.

Server	URI	UserAgent
smileenh????.co.uk	/images/sm3.php	'-'
dorsets????.org	/images/sm3.php	'-'
calu????.it	/images/sm3.php	'-'
zi??.nl	/wp-content/uploads/sm3.php	'-'
...	...	...

Note: for the privacy protection reason, we use ? to mask part of the detected domains.

Table 4.10: Zeus botnet.

Zeus C&C Server	URI
4k0t155m.cz.cc	/login.php
4k0t177m.cz.cc	/login.php
4k0t144m.cz.cc	/login.php
4k0t166m.cz.cc	/login.php
4k0t111m.cz.cc	/login.php
...	...

servers. This confirms that they belong to the same attack campaign. Note that most of the URIs have the path "wp-content," which indicates that those servers are installed with WordPress web application. This attack campaign explores WordPress vulnerability to upload a malicious script "sm3.php."

Table 4.10 is the Zeus botnet [57] that SMASH also inferred. This campaign includes eight C&C servers of Zeus. 2012 IDS signatures labeled none of these domains while the blacklists detected only one domain. However, 2013 IDS signatures detected all of these domains. This shows that, as SMASH does not need to update signatures, it can detect zero-day attack campaigns. This campaign seems to be using a DGA-based algorithm to generate similar domain names. All these domains are requested by the same set of clients, and share the same IP addresses and filename "login.php." We searched these domains in Google, and only "4k0t111m.cz.cc" is confirmed as "Zeus tracker."

4.5 Discussion

4.5.1 Overhead

SMASH is designed to monitor the traffic at the edge of a network. Thus, it can be deployed at enterprise or ISP networks. The most expensive computation part of SMASH is on the similarity calculation, whose complexity is N^2 where N is the number of servers, as we need pairwise similarity among different servers. However, the complexity of similarity calculation can be significantly reduced by using Bloom filters [5] or sparse matrix multiplication [67].

4.5.2 Limitations

SMASH assumes that cyber criminals use multiple servers to conduct their malicious activities. Thus, if an attacker uses only a single server to conduct malicious activities (which is now very rare), SMASH can not detect it. In addition, since we use second-level domain for the inference, we might miss the malicious servers using dynamic DNS or hosted on third-party cloud servers. However, those services could incur the significant financial cost to the attackers.

4.5.3 Evasions

SMASH relies on the correlation between the main and secondary dimensions. Thus, an attacker who gains the knowledge of SMASH might try to mislead our system by manipulating their relationships as follows:

4.5.3.1 Evading the Main Dimension:

To mislead the main dimension, an attacker can make their bots visit many benign domains with the same URI file.[10] In this case, our main dimension might generate ASHs that include both benign and malicious servers. However, since our client similarity looks at the similarity among all the client sets, it is difficult for an attacker to assure that there are no other benign clients that visit those benign domains. Even when an attacker can use some benign servers to mislead our system, their malicious servers are still included in ASHs that SMASH inferred, which can be further filtered through other heuristics. For example, benign domains might not have such URI files, which may return an error code. In addition, an attacker can also let different bots communicate with different servers to prevent us from generating ASHs. However, this would be a very costly method for attackers, as the more bots they have, the more servers they need to register.

4.5.3.2 Evading Secondary Dimensions:

Compared with the main dimension, secondary dimensions can be relatively easily changed. However, the process is very inconvenient and costly for the attackers. For example, to evade the IP dimension, an attacker can fast flux the IP addresses of their servers, which is very expensive yet easily detected [75]. To evade the URI File dimension, an attacker can assign different names for different servers. However, it makes their connections less scalable; for the attacking campaigns, it usually targets the vulnerabilities of certain files, and thus an attacker can not change such filenames. Although an attacker may successfully evade one of the secondary dimensions, it is non-trivial to simultaneously evade all dimension.

[10]There is a very low possibility that the benign domains share similar IP addresses and Whois information with the malicious servers.

5. CHARACTERIZING AND DETECTING MALICIOUS ENTRANCES TO MALICIOUS CYBER INFRASTRUCTURES*

Previously, we have described PoisonAmplifier and SMASH. PoisonAmplifier only focuses on compromised servers that are utilized in Search Poisoning attacks. Therefore, it is not applicable to detect the general compromised servers. SMASH can discover the relationship among different malicious servers to infer a group of malicious servers. However, it requires multiple malicious servers involved in. In this chapter, we introduce a new system VisHunter, which mainly focuses on detecting entrances (e.g., redirections from compromised servers to malicious servers) to malicious cyber infrastructures. Based on the detected malicious entrances, its propagation component can further find more malicious servers under the malicious cyber infrastructures. Comparing to PoisonAmplifier and VisHunter, VisHunter can cover all types (e.g., compromised server, Type A and B malicious server) of servers we defined in our model without requiring multiple malicious servers involved in.

VisHunter studies malicious web infrastructures from a novel perspective, i.e., *visibility*. In a nutshell, we define the visibility[1] of a server as whether the server is visible to benign users, for instance, through search engines. To obtain a comprehensive understanding of whether invisible malicious web infrastructures are indeed popular, we investigate 100,000 benign servers and nearly 45,000 malicious servers collected from both *public blacklists* and *real-world enterprise networks*. Our key

*Part of this chapter is reprinted with permission from "Hunting for Invisibility: Characterizing and Detecting Malicious Web Infrastructures through Server Visibility Analysis" Jialong Zhang, Xin Hu, Jiyong Jang, Ting Wang, Guofei Gu and Marc P. Stoecklinby. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM'16)*, Copyright © 2016 by IEEE.

[1]An in-depth discussion on visibility will be in Section 5.2.

findings include: 1) Unlike legitimate servers, a large number of malicious servers tend to be invisible, especially for some categories such as C&C servers and exploit servers. C&C servers used to communicate with bot-infected machines are almost always invisible to benign users. This is because bot masters naturally try to minimize the exposure of their C&C servers to keep their malicious activities under the radar. 2) Server visibility alone is not sufficient to precisely pinpoint malicious web infrastructures because a small number of benign servers are also invisible. Therefore, we further examine 41,190 redirections collected from a large enterprise network and observe that the "entrance" into malicious web infrastructures, i.e., redirection from visible servers to invisible servers, is significantly different from that of benign web infrastructures.

Motivated by these findings, we design a lightweight yet effective system, VisHunter, to detect the malicious web infrastructure, which includes compromised servers, malicious entrances, and post-infection servers. VisHunter uses a trained classifier to identify the "entrance" to the malicious web infrastructures based on 12 features including several novel visibility-related features and applies a graph-based propagation algorithm to find more compromised servers and post-infection servers starting from detected malicious entrances. Our evaluation with one month traffic from a large enterprise network shows that VisHunter achieves an average true positive rate of 96.2% at a false positive rate of 0.9% for the malicious entrance detection, and identifies 6x more malicious servers under the malicious cyber infrastructure based on the detected malicious entrances.

5.1 Research Goal and DataSet

5.1.1 Research Goal

Our research goal is to provide the first empirical analysis on the visibility of malicious cyber infrastructures. Through analyzing the visibility pattern of both benign servers and malicious servers (in Section 5.3), we identify a set of distinct features of malicious servers involved in malicious cyber infrastructures from their locations, structures, roles, and relationship. Based on these features, we aim to design an efficient system to detect the servers involved in malicious cyber infrastructures from the visibility perspective, which could be complementary to existing work [76, 78].

5.1.2 DataSet

Our dataset consists of both benign servers and malicious servers extracted from public blacklists and the real enterprise network traffic.

5.1.2.1 Public Blacklists

This dataset consists of domains from two blacklists: Malware Domain List [29] and DNS-BH Malware Domain Blocklist [9], both of which have been widely used as ground truth in existing work [62]. We collected 43,768 malicious SLDs from 2009 to 2014 from [29] and 10,967 SLDs from 2011 to 2014 from [9], and performed visibility check on Oct 2014. Results show that around 80% of them were invisible. One caveat here is that malicious domains may have already expired leading to search engines may not index them anymore. To avoid such bias, we selected only the domains that were blacklisted in 2013 and 2014, and verified their existence by checking if the domains could be successfully resolved to IP addresses. As a result, this dataset contains 875 malicious SLDs from Malware Domain List (`M_PB1`) and 5,739 malicious SLDs from DNS-BH (`M_PB2`).

5.1.2.2 Enterprise Traffic

To avoid possible bias caused by public blacklists, we also captured real network traffic from a large enterprise network from June 16 to June 20, 2014, from which we extracted 462,226 unique servers. Among them, 1,782 servers (`M_Enter`) were detected as malicious by an internal intrusion detection system (IDS). For those that were not detected by the IDS, we randomly chose 100,000 benign servers that did not share any clients with malicious servers in `M_Enter` and labeled them as `B_Enter`. We performed the viability check on these servers at the same time we captured them. We also collected one month traffic of an institute in June 2013. Due to privacy and storage constraints, we stored only the metadata and the header information of all the web requests and responses, which included the request URL paths, HTTP response codes, host names, user agents, referrers, cookies, and so on. Notice that without the content of the web pages, we were not able to directly determine certain types of redirections, e.g., JavaScript or iFrame based redirections. Therefore, we focused on HTTP header redirections in this dataset. In total, we extracted 41,190 redirections(`R_Enter`) and performed the viability check on this data on June 2014.

Table 6.4 summarize the data collection results.

Table 5.1: Data collection

	M_PB1	M_PB2	M_Enter	B_Enter	R_Enter
# of servers	875	5,739	1,782	100,000	165,957
# of redirections	N/A	N/A	N/A	N/A	41,190

5.2 Server Visibility

The concept of visibility used in this work hinges on how a normal user locates a server. For popular or frequently visiting websites, a user may directly type the domain names into the address bar or follow bookmarks to access the web servers. We consider these servers visible to normal users. For other (less popular) websites, a user may locate them using search engines, and access the websites through search results. We also consider these web servers visible if the server itself (not only the domain name) is indexed by search engines. Intuitively, benign servers are more likely to be *visible* to normal users because their owners are often motivated to promote their websites in search engines to increase their user base. On the contrary, malicious servers, especially the ones in the core malicious infrastructure (e.g., exploit servers), are less likely to be indexed by search engines because attackers try to minimize their exposure and avoid being detected. Therefore, we determine server visibility as follows (cf. Fig. 5.1).

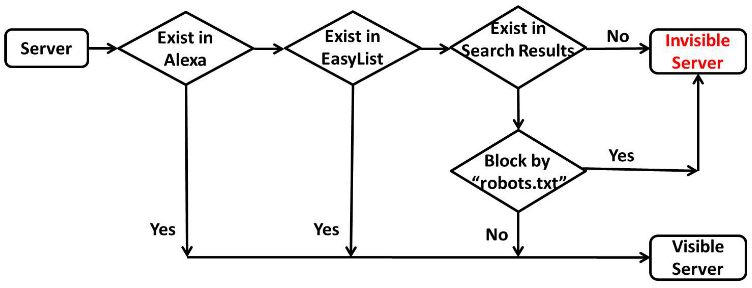


Figure 5.1: Determination of server visibility

1. For a domain name, we extract its second-level domain (SLD) based on [46]. Since an SLD is commonly associated with the organization that registers the

domain, we assume that a domain has the same visibility as its SLD. In the remaining of this chapter, the term domain denotes its SLD unless stated otherwise.

2. If a domain is well-known (e.g., `google.com`), we consider it visible. We utilize two public whitelists to determine well-known domains: top 1 million domains from Alexa [2], and domains from EasyList [10]. Alexa provides a global popularity ranking of domains based on their collected web traffic. EasyList provides a list of well-known Ad-network domains and trackers. Notes that domains in whitelists only reflect the popularity of them, it does not mean that they are benign domains.

3. For other domains, we query them in search engines [2] and obtain top 100 [3] search results. If the queried domain appears in the search results, meaning that it was crawled and indexed by the search engines, we further examine the indexed content of the domain. If the site owner blocks the access to the content (e.g., Google displays a message under the domain name "A description for this result is not available because of this site's robots.txt"), we consider the domain invisible; otherwise, it is considered visible. Note that since multiple search results may be returned for a domain, for the domain to be classified as invisible, none of them should have available content. This is to avoid misclassifying legitimate cases where a site owner may use robots.txt to prevent search engines from crawling their sensitive webpages (e.g., admin interface).

4. If the queried domain does not appear in the top 100 search results, we consider

[2]We use Google search engine in the current implementation. However, other search engines could be used to reduce possible bias of search engine results.

[3]100 is the maximum number of search results per page. Since we directly search the domain, the pages on that domain are usually returned in the first.

it invisible.

5. All IP addresses are considered as invisible.

Noting that the definition of server visibility depends on search engine results that may change dynamically along with time, we evaluate the server visibility over time in section 5.5.2.

5.3 Measurement Study of Server Visibility

Our hypothesis is that legitimate servers are more likely to be visible because their owners have the incentives to promote their products or services. However, certain categories of malicious servers (e.g., exploit servers), tend to remain invisible for several reasons. First, from the cyber criminals' perspective, they may only want to allure their targeted victims to reach the core malicious servers in order to minimize the exposure to security analysts because previous work has shown that it is easy to pinpoint malicious servers using search engines [76]. Second, from the search engines' perspective, it may not be straightforward to crawl and index malicious servers. Some malicious servers are intentionally isolated from the World Wide Web without any hyper-links pointing to them. Other malicious servers may be ephemeral and dynamically changing such as domains generated by domain generation algorithms (DGAs). Third, search engines employ their own algorithms (e.g., PageRank) to index and rank servers. In general, they prefer to return to their users with high reputation websites. Hence, malicious websites may not be indexed by search engines or not be returned to the users.

To validate our hypothesis, we conduct a comprehensive measurement study on the visibility of both benign and malicious servers using real-world datasets.

5.3.1 *Server Visibility Study*

5.3.1.1 *Visibility of Malicious Servers*

We first determined the visibility of malicious servers in `M_PB1`, `M_PB2`, and `M_Enter` using the process outlined in Section 5.2. To better understand how visibility correlates with specific malicious types, we measured the visibility distribution of malicious servers within each attack category. Based on the description of malicious functionalities in `M_PB1` and `M_PB2`, we classified them into 63 and 109 categories respectively.

We then calculated *invisibility ratio*, defined as the number of invisible servers over the total number of servers in a category. The CDF of the invisibility ratio distribution across all the categories is illustrated Figure 5.2. Only around 10% of categories from `M_PB1` had the ratios lower than 30%, meaning that the servers in those categories were likely to be visible. For `M_PB2`, around 35% of categories had low ratios.

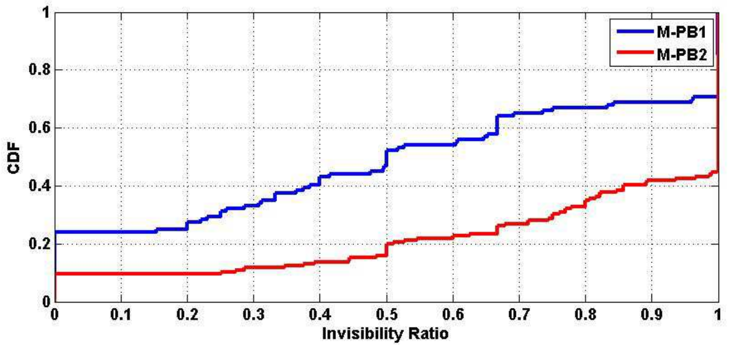


Figure 5.2: Invisibility ratio distribution across all categories

Table 5.2 lists the top 5 largest *visible* malicious categories in terms of their sizes.

Most visible servers in `M_PB1` were compromised servers and had labels, such as "leads to", "iFrame", and "JavaScript". `M_PB2`, on the other hand, had different distribution of malicious servers with more social-engineering type of attacks like phishing and rogue software. These servers, by their nature, were designed to be easily accessible to unsuspecting users, and therefore were more likely to be visible. Overall, less than half of servers, 356 (44.49%) in `M_PB1` and 363 (31.33%) in `M_PB2` belonged to these categories.

Table 5.2: Top 5 largest visible malicious categories

M_PB1			M_PB2		
Category	# of servers	Invisibility ratio	Category	# of servers	Invisibility ratio
leads	246	25.20%	unsafe	159	22.01%
spyware	21	28.57%	highrisk	65	23.07%
iFrame	17	29.41%	fake	27	25.93%
JavaScript	14	21.43%	phishing	13	15.38%
compromised	5	0%	rogue	53	0.25%

Table 5.3: Top 5 largest invisible malicious categories

M_PB1			M_PB2		
Category	# of servers	Invisibility ratio	Category	# of servers	Invisibility ratio
blackhole	31	90.32%	malware	652	65.18%
-	21	52.386%	malspam	215	64.65%
drive-by	5	66.66%	zeus	81	60.49%
java	5	100%	fake_flash	34	73.53%
fake	4	100%	putter_panda	27	96.29%

As a comparison, Table 5.3 lists the top 5 largest *invisible* malicious categories with their sizes. We can see that malware and exploit servers, such as blackhole, Zeus and drive-by-download, were among the most common invisible malicious server types. For example, more than 90% of blackhole servers and more than 96% of "putter panda" servers, which were C&C servers used in a cyber espionage campaign, were hidden from search engine crawlers.

Noting that the invisibility ratio for the largest invisible malicious category is not 100%. This is because that some of them are actually compromised servers rather than malicious servers. For example, `northerningredients.com` was visible according to our visibility definition, and it was labeled as malicious by both `Malciousdomain.com` and VirusTotal. Further investigation on its Whois information and its web contents revealed that the food company's domain was registered in 2006 with an expiration date in 2019. Such a long history made us believe that it was a compromised legitimate domain instead of a malicious server set up by cyber criminals. In addition, other visible malicious servers usually shared certain patterns in their contents or URLs, which could be leveraged to efficiently detect a group of similar servers using existing work such as EvilSeed [76] and PoisonAmplifier [78].

We further checked the visibility of `M_Enter` from the enterprise network. 67.51% of them were invisible, and 13.99% of them belonged to the top 1 million Alexa web list, indicating that they were likely compromised or abused.

5.3.1.2 Visibility of Benign Servers

Next, we checked the visibility of benign servers in `B_Enter`. As expected, only very small portion of them, i.e., 6,626 (6.63%) were invisible. To better understand the underlying causes for their invisibility, we manually analyzed a set of randomly selected 100 servers. We found that most of the benign invisible servers were: 1) new servers which had not been indexed; 2) service providers which actively blocked crawlers or chose not to be indexed by search engines. As a result, we can leverage these characteristics to distinguish them from their malicious invisible servers, which we will elaborate in Section 5.4.2.

Lessons learned: As demonstrated above, there exist significant and consistent differences between certain malicious servers and legitimate servers in terms of their visibility status. These findings suggest that visibility could be an effective feature for malicious server identification. However, we also note that visibility alone is not sufficient, as many legitimate servers, although few percentages, may not be directly accessible to users through search engines for various reasons. Therefore, we explore several new redirection-based features to augment visibility and minimize false positives.

5.3.2 Visibility Study on Redirections

Based on the visibility of each server, we clustered the redirections in `R_Enter` into four categories: visible to invisible (1,063 (2.58%)), visible to visible (36,727 (89.17%)), invisible to invisible (1,559 (3.78%)) and invisible to visible (1,841 (4.47%)). Unsurprisingly, the majority of redirections were among visible servers, which were mostly benign redirections with a few from one compromised server to another.

Many existing systems detect malicious redirections using the characteristics of the full redirection chains [88, 99], such as chain lengths, geolocations of landing and final servers, and so on. Unfortunately, such features can be easily manipulated when extracting them on the whole redirection chain. For example, attackers can change the length of the redirection chain by appending more/fewer compromised or malicious servers. Alternatively, attackers can also add arbitrary inner domain redirections since they can partially control compromised servers and fully control their own malicious servers.

In contrast, only the transition from visible to invisible servers is more resilient to manipulation whenever attackers want to lure unsuspecting users to their malicious infrastructures. Therefore, we only focus on such kind of redirection and inspect it from several different perspectives.

To collect ground truth, we used VirusTotal to label 1,063 redirections from visible servers to invisible servers. Specifically, if an invisible server was labeled as malicious by at least two anti-virus vendors, we considered the redirection as malicious. In this way, we finally collected 27 malicious redirections. To collect benign redirection cases, we checked the Whois history of invisible servers for the remaining redirections. We removed redirections whose invisible server had a lifetime (the expiration date minus the creation date) less than or equal to one year, which have a high chance to be malicious but not yet been labeled by VirusTotal. As a result, we collected 683 benign redirections.

Below we itemize our observations and lessons learned from these redirections

5.3.2.1 Location Attributes

Typically attackers can not control the place where the compromised benign servers are located so that visible compromised servers and invisible malicious servers would be located at different places. In VisHunter, we characterize the location difference using IP addresses, Whois information, and autonomous systems numbers (ASNs). Specifically, we consider a redirection to be made between different locations if its visible and invisible servers do not locate under the same IP subnet (/24), do not share the same Whois information, and do not have the same ASNs. Otherwise, the two servers are likely to be co-located. In our ground truth dataset, all but one malicious redirections had different locations. On the other hand, 188 (27.53%) of benign redirections had co-located visible and invisible servers. In fact, those visible servers usually hosted the home page of the company's website while those invisible servers provided internal or non-public services. These invisible servers may actively block search engine crawlers and/or there is no way for the crawlers to find them.

5.3.2.2 Structure Attributes

To capitalize on their resources, attackers often compromise a large number of servers and point them to a small set of their core malicious servers. Thus, we assume there should be an authority structure in the malicious redirections: multiple visible servers redirect to a few authority invisible servers. To characterize this hub/authority structure, we propose a metric "in-out ratio", defined as the degree of the invisible server over the degree of the visible server. A ratio lower than 1.0 means that multiple visible servers redirect to only a few invisible servers, making the invisible servers authorities. On the other hand, a visible server that redirects to many invisible servers will have a high in-out ratio and become a hub in the redirection graph.

Looking at our ground truth dataset, 33.33% (9) of malicious redirections had invisible authorities, and 7.41% (2)had visible hubs. In comparison, for the benign redirections, only 3.22% (22) of them had invisible authorities while 80.23% (548) of them had visible hubs. For all the hubs, we further checked the number of IP addresses they redirected to. The intuition here is that malicious hubs may redirect to multiple domains hosted on the same IP address in order to minimize cost and maximize server utilization. On the other hand, benign hubs may redirect to multiple servers which belong to different organizations and thus have a large number of IP addresses. This intuition was also confirmed by our data: a malicious hub indeed redirected to two domains that shared the same IP address, whereas those benign hubs all redirected to multiple IP addresses.

5.3.2.3 Role Attributes

Some benign redirections are caused by advertisement networks. One notable characteristic of an advertiser is that it can redirect to a large number of both visible and invisible servers. In this work, we use the metric Num_{vis}, the number of redirections to visible servers, to define advertisers, and find 467 advertisers in the benign redirections with $Num_{vis} > 3$ (68.37%). We also define the reputation of an advertiser *Rep* as Num_{vis} / Num_{invis}, where Num_{invis} is the number of redirections to invisible servers. Essentially, an advertiser is considered to be more suspicious if it redirects to more invisible servers than visible servers. In fact, such a pattern has been used by certain cloaking servers, which redirect target users to either exploit servers or legitimate websites depending on users' operating systems and browser versions. 174 (37.26%) of advertisers redirected to more visible servers than invisible servers for benign redirections; therefore, they were more likely to be benign . In this dataset, there was no advertiser in malicious redirections.

5.3.2.4 Relation Attributes

Benign redirections often serve a purpose, for example, moving websites to another server, load balancing, delivering contents from local data centers, etc. We characterize such relationship from the following perspectives.

CDN: CDNs account for a large portion of the benign redirections. In addition to whitelisting well-known CDNs such as Akamai, CloudFront, etc., we apply a heuristic to attribute a redirection to a CDN if two servers involved have the same URL path and the path length[4] is longer than 2. 19.18% (131) of benign redirection in our dataset fell into this category whereas none of the malicious redirections had a CDN relationship.

[4]We measure the path length based on the number of slashes ("/") in URLs.

Other general partner relation: To identify other general partner relationships, we employ two heuristics leveraging both historical information and search engines results. First, if a specific redirection happens regularly over a long period of time T[5], we consider the two servers have a stable partner relationship. Second, we leverage search engines to reveal the potential partnership between two servers. More specifically, we query two servers involved in the redirection in the search engines at the same time, for example, search "visible.com and invisible.com" in Google. If both of them appear in the same search results, we believe that there likely exist the relationship between these two servers. In our dataset, most of such relationships were due to sharing contents and were also reported by websites like `siteslike.com` and `websitesalike.com`. To further eliminate potential false positives that may be caused by security websites analyzing a particular malicious server instance, we ignore such partner relationship if the search results contain any security related keywords such as "security", "virus", "malicious", and "malware", etc. As a result, only 5 (18.52%) of malicious redirections had partner relationships while 513 (75.11%) of benign redirections had such relationships.

Lessons learned: Detecting malicious redirections is a complicated task. Simply relying on features associated with compromised servers is immediately subject to evasion, given the attacker's freedom to use public services or multiple compromised servers. At the same time, detecting malicious terminal servers is hindered by their diversities and various cloaking techniques. However, observations from our measurement study convey a positive message: malicious redirections from visible servers to invisible servers exhibit distinguishable behaviors from their benign counterpart, which are more intrinsic to malicious infrastructures and difficult to evade.

[5]In current implementation, we set T=1 month.

5.4 System Design

5.4.1 System Overview

Based on the study in Section 5.3, we observe that there exist certain categories of malicious servers that are always invisible, and there exist notable differences between entrances to malicious invisible infrastructures and benign invisible infrastructures. Therefore, VisHunter focuses on the redirections from visible servers into invisible servers and leverages discriminative features to detect the entrances into malicious web infrastructures, especially for the entrances to the exploit infrastructure, where most traffic comes from compromised servers to exploit servers. Furthermore, VisHunter automatically identifies connections among invisible servers based on their relationships and infers other malicious servers under the malicious web infrastructure through graph-based propagation.

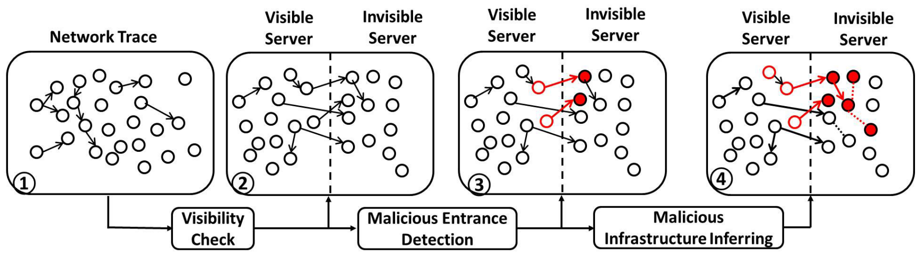


Figure 5.3: System overview.

An overview of VisHunter is shown in Figure 5.3. VisHunter takes HTTP traffic as an input and extracts all observed servers (nodes in ①) as well as the redirections among them (edges between servers in ①). Then, VisHunter checks the visibility of each server and divides them into two categories: visible servers and invisible servers (②). Next, only the redirections from visible servers to invisible servers are submitted to the malicious entrance detection component, where a trained classifier is used to identify malicious entrances (red edges in ③). Later, the malicious infrastructure inferring component will construct a relation graph-based on all the recorded redirections (solid edges in ④) and the newly-derived relationships among invisible servers (dash edges in ④). The malicious infrastructure inferring component then applies the PageRank algorithm on the generated relation graph to infer the entire malicious web infrastructure, essentially to find more compromised servers (red nodes in the left part of ④) and malicious servers (red filled nodes in the right part of ④).

5.4.2 Detecting Malicious Entrances

We define an "entrance" as a redirection from SLD of a visible server to SLD of an invisible server. An entrance is considered malicious if the destination invisible server is malicious, i.e., belonging to attackers' invisible malicious web infrastructures. Based on our intensive study in Section 5.3.2, we propose 12 features that characterize the differences between benign entrances and malicious entrances, which are difficult for attackers to evade without a significant amount of cost. As summarized in Table 5.4, the features leveraged by VisHunter are divided into four groups.

Table 5.4: Feature selection

Aspects	Features	Novelty
Location	IP location	[88]
	Whois location	New
	AS location	[88]
Graph	In-degree of invisible server	New
	Out-degree of visible server	New
	In-Out-ratio	[98]
	IP diversity of invisible server	[92]
Role	Advertiser	New
	Reputation of advertiser	New
Relation	CDN	New
	Partner based on history	New
	Partner based on search results	New

Location-based Features: This group aims to capture the location differences of the entrances. Since this group of features is derived from the physical locations of the entrances, it is difficult for attackers to forge.

Location-based features proposed in existing work are not as resilient as our proposed features. For example, the location differences between a landing server and a terminal server [88] can be easily changed by adding another redirection to the original server after an infection. An attacker can simply add an iFrame in the malicious server such that it sends users back to the landing server. In this way, it can evade location features in [88]. On the other hand, as long as attackers rely on compromised servers to redirect users to their malicious web infrastructures, VisHunter will detect the location difference.

Graph-based Features: This group aims to characterize the structure of entrances based on graph properties. Attackers would be required to change their fundamental operation structures for evasion. For example, to evade the invisible authority feature, malicious invisible servers would be allowed to use only a few compromised servers to redirect to them, which effectively limits the effectiveness of attackers' operations.

Role-based Features: This group aims to distinguish between benign and malicious entrances to the advertisement infrastructures. To evade this group of features, attackers are required to either abuse public known advertisers, for example, `googleadservices.com` to redirect traffic, which is not trivial since those services usually have strict scrutiny processes, or to directly redirect users using compromised servers, which can be detected through other features (e.g., location-based features).

Relation-based Features: The group aims to characterize general entrances of benign infrastructures. CDN-based features are hard to evade since malicious servers usually have different paths with compromised servers. At first glance, partner based on search results feature seems easier to evade, e.g., attackers may circumvent search engine partner relationships by posting compromised servers and malicious servers together. However, such behavior could lead existing work (e.g., EvilSeed [76] and PoisonAmplifier [78]) to find more malicious servers easily.

As a result, 12 features are extracted for each entrance, and a classifier (J48 decision tree) is trained with known malicious and benign entrances to detect the entrances to malicious web infrastructures. We acknowledge that attackers may artificially manipulate some of our proposed features to evade VisHunter. However, it is not trivial for attackers to evade the detection based on the combined use of all features without putting a significant amount of investment.

5.4.3 Inferring Malicious Infrastructures

Malicious entrance detection component identifies malicious entrances, which essentially characterizes the typical exploit process where an unsuspecting user is redirected by a compromised server and lured into an attacker's malicious web infrastructure. However, as described in Figure 1.2, a victim machine may directly visit malicious servers at the post-infection stage, e.g., bots connecting to their C&C servers. Therefore, this component is to infer the other missed malicious servers (e.g., post-infection servers) under the malicious infrastructures.

To obtain a complete view of the entire malicious infrastructure, the inferring component first builds a relation graph to characterize the relationship among malicious servers, then it exploits a graph based propagation algorithm to identify additional malicious servers based on their correlation with malicious servers already detected by the previous component and their visibility statuses. Specifically, we use the following two heuristics to build a propagation graph.

Malicious hosts are typically inter-connected. Recent work [85] shows that there exist a set of topologically dedicated malicious servers linking to 76.2% malicious redirections. Motivated by this observation, we first build the relation graph with all the redirections, not only between visible and invisible servers but also amongst visible/invisible servers themselves. Thus, through the propagation of malicious scores to those dedicated malicious hosts, we can find additional compromised servers and visible malicious servers. To exploit the locality among malicious servers, we further add an edge to the propagation graph if two servers share the same IP subnet (/24). Since most malicious servers are invisible, we add such edges only for invisible servers to reduce the graph complexity.

Malicious servers are typically accessed by only compromised clients. Post-infection malicious servers, such as C&C and drop-zone servers, are often intended to be accessed only by infected client machines. To characterize this relationship, we add edges between client nodes and their visiting invisible servers on the propagation graph.

In this way, we build our propagation graph as a heterogeneous graph. The nodes consist of both servers and clients, and the edges reflect the connection via redirections, IP sharing, and direct visiting. Next, similar to how the PageRank algorithm evaluates the importance of webpages based on their link structure, VisHunter propagates malicious scores from detected malicious servers to other servers. We note that some compromised servers are legitimate servers that may also be connected with other benign servers. Therefore, to propagate malicious scores inside malicious web infrastructures, we compute two scores for each server in a similar way to [85]: one to measure the server's reputation and another to measure the closeness to malicious infrastructures.

Specifically, we use the visibility property to characterize closeness to malicious infrastructures. At the beginning, we assign each server s_i an initial visibility score vis_{s_i} as follows: 1) all the invisible servers will have a visibility score 0; 2) all the visible servers that do not redirect to/from invisible servers will have a visibility score 1; 3) for those visible servers that have connections with invisible servers, we define their visibility score as: $vis_{s_i} = \frac{Num_{vis}}{Num_{invis}+Num_{vis}}$ where Num_{invis} and Num_{vis} represent the number of connections to invisible and visible servers from server s_i, respectively. Therefore, a high visibility score means that the server is far from malicious infrastructures, thus more likely to be a benign server.

In addition, VisHunter also assigns each server an initial malicious score mal_{s_i}, with $mal_{s_i} = 1$ for the servers among the identified malicious entrances and $mal_{s_i} = 0$ for the other servers. Then, VisHunter propagates both scores separately on the propagation graph with a PageRank-like algorithm. After rounds of iterations till both scores are converged, servers with high malicious scores and low visibility scores will be considered to be a part of malicious infrastructures.

5.5 Evaluation

In this section, we describe our collected datasets for evaluation and present evaluation results of each component of VisHunter and case studies. Then, we compare VisHunter with existing approaches.

5.5.1 Data Trace and Ground Truth

Enterprise Traffic: We collected 6 months (from July 2013 to Oct 2013 and from Nov 2014 to Dec, 2014) traffic from the same institute described in section 5.1.2.

Online Public Malware Traffic: We also crawled malware traces from a public website `malware-traffic-analysis.net` [31] which provided more than 402 pcap traces of malware collected from June 2013 to Jan 2015. Those traces showed detailed analysis on how malware was delivered, typically through compromised servers and drive-by-download exploit kits. Specifically, 213 cases used various redirection methods to deliver malware, including JavaScript redirection, iFrame redirection, and HTTP header redirection. Among them, 159 redirections were from visible servers to invisible servers. Others were either because the incomplete traces made it impossible to obtain the corresponding compromised servers or attackers leveraged public servers (e.g., dynamic DNS server `redirectme.net`) as their exploit servers.

5.5.2 Time Impacts on Visibility

As the server visibility depends on search engine results that may change dynamically, we measured the stability of visibility over time. Specifically, we recorded the visibility of each server in `M_Enter` and `B_Enter` when we collected the dataset (June 2014) and then re-checked their visibility status every two months. The results are summarized in Table 5.5. We can see that server visibility was relatively stable.

Only about 2% of the servers, regardless of their maliciousness, changed their visibility status after 6 months, though malicious servers seemed slightly more volatile. Further investigation showed that visible servers changed into invisible primarily because their domain names expired and hence were removed from search engine indexes. On the other hand, for those invisible servers that changed into visible, it was mainly because those servers were newly registered and recently indexed by search engines or they completed website construction and unblocked the crawlers in "robots.txt".

Table 5.5: Stability of visibility

time	visible $\rightarrow$ invisible		invisible $\rightarrow$ visible	
	`M_Enter`	`B_Enter`	`M_Enter`	`B_Enter`
Jun,2014	-	-	-	-
Aug,2014	20 (1.12%)	1,043 (1.04%)	22 (1.23%)	885 (0.86%)
Oct,2014	34(1.91%)	1,782(1.78%)	30(1.68%)	1,306(1.31%)
Dec,2014	47(2.64%)	2,615(2.61%)	36(2.02%)	1,583(1.58%)

5.5.3 Search Engines Comparison

To evaluate the possible bias of the visibility results for different search engines, we randomly select 100 servers from B_Enter and M_Enter respectively, and test their visibility on different search engines. We use visibility from Google search results as the baseline. "+" means other search engines find new visible servers, and "-" means other search engines mislabel some visible servers to invisible servers. Table 5.6 shows the results. We can see that overall different search engines return similar results. Bing and Yahoo have very similar results since now Yahoo search engine is based on Bing. Google only mislabels few visible servers as invisible servers. The missed servers in `M_Enter` are probably because those search engines refrain from showing known malicious sites. In addition, the combination of different search engines can provide a better view of visibility, which could be leveraged in our future work.

Table 5.6: Comparison among different search engines

	Visible servers in `B_Enter`	Visible servers in `M_Enter`
Bing	+4,-4	+3,-14,
Yahoo	+4,-4	+4,-13,

5.5.4 Malicious Entrance Detection Results

To evaluate the performance of VisHunter, we first performed a 10-fold cross-validation of VisHunter's classifier with S_{train}, the ground truth dataset we used for the measurement study on redirections in Section 5.3.2. The J48 classifier achieved an average true positive rate of 96.2% at a 0.9% false positive rate.

We investigated the misclassified cases. The only missed malicious redirection (false negative) was because a visible server redirected to two invisible servers with different IP addresses, which was labeled as a benign advertisement behavior. For six false positive cases, they all redirected from a visible server to an invisible server. Two of the redirections, even though their domains were not detected by VirusTotal, included IP addresses labeled as malicious. For the redirection `deal4u.in` $\rightarrow$ `rggg.net`, the invisible server is now visible and for sale. The redirection `eoaclk.com` $\rightarrow$ `paragonhondaoffers.com` is an interesting case. `paragonhondaoffers.com` was the website of a car company who blocked search engines' crawlers. This was not usual because car dealers would want to promote their offers. We further checked its Whois information and found that the domain was registered by a third party

Table 5.7: Malicious entrance detection results

VisHunter	2013				2014		13-14
	Jul	Aug	Sep	Oct	Nov	Dec	Malware
	59	26	36	41	34	69	135
Confirmed	34	11	15	21	12	27	135
Suspicious	9	1	6	6	8	17	0
Manual	0	1	2	1	7	3	0
Expired	9	8	4	2	0	0	0
False Positive	7	5	9	11	7	22	0
False Negative	N/A	N/A	N/A	N/A	N/A	N/A	24

company that provided marketing services. The remaining two false positive redirections were essential between the partner websites that provided similar products. This could be addressed by calculating the topic similarity of two websites.

We further evaluated VisHunter with 6 months enterprise traffic and the public malware traces. Table 5.7 presents the number of malicious entrances VisHunter detected. To get the ground truth of those detected entrances, we checked the invisible servers in the entrances against VirusTotal. If at least two anti-virus software detected an invisible server as malicious, we believed that the corresponding entrance was malicious, and marked it as "confirmed". If only one anti-virus software detected it as malicious, we marked the corresponding entrance as "suspicious". If a domain was expired, we marked it as "expired". For all the remaining servers, we manually verified them. If a server was reported by other resources (security blogs, analysis reports, and etc) as malicious, we marked the corresponding entrance as "manual"; otherwise, it was considered to be a false positive.

Note that the false positives here were not always benign entrances. Some of them did have suspicious behaviors. For example, the redirection `???ssdns.com/wp-content/favicon1.png` → `???cloudproxy.com/2devnulltracker`, was suspicious as it was from an image file `favico1.png` to a proxy server. However, since we were unable to procure an evidence to confirm its maliciousness, we conservatively labeled it as a false positive. Some other false positives were due to CDNs. For example, for the entrance `nv.ahcdn.com/axx/598132.flv` $\rightarrow$ `88.208.57.3/bxx/598132.flv`, the two servers shared similar path patterns and the same filename. However, since our CDN-related features required that two servers shared the same URI, VisHunter detected the entrance as a false positive. For the remaining false positives, we found that they were the entrances to benign partner web servers. One complementary feature to eliminate these false positives is to check topic similarity of the two servers.

We also note that VisHunter was capable of detecting new malicious invisible servers that were missed by VirusTotal, even though one would expect that for data from 2013, which were more than 2 years old, VirusTotal would have already captured the most, if not all, malicious servers. In fact, for the recent data from 2014, we have successfully submitted several new malicious cases to VirusTotal. Therefore, we believe that VisHunter, as a behavior-based approach, is complementary to the widely used blacklisting and signature-based methods (e.g., IDS), and has a potential to detect targeted/stealthy attacks that elude public blacklists.

For the online malware traffic traces, VisHunter detected 84.9% of all malicious entrances. The missed cases were mainly the visible servers redirecting to multiple invisible servers with completely different IP addresses. Further investigation showed that this was because we aggregated all the redirections over one and half year together. For a shorter period of time, e.g. 1 month, the compromised servers or the public proxy servers abused by attackers only redirected to a limited number of invisible malicious servers.

For those confirmed malicious servers, we further extracted the earliest timestamp when they were detected by VirusTotal and compared it against our detection time.

Figure 5.4 shows the CDF of the detection time difference distribution. We can see that when VisHunter detected those malicious servers, around 50% of them were still not detected by VirusTotal.

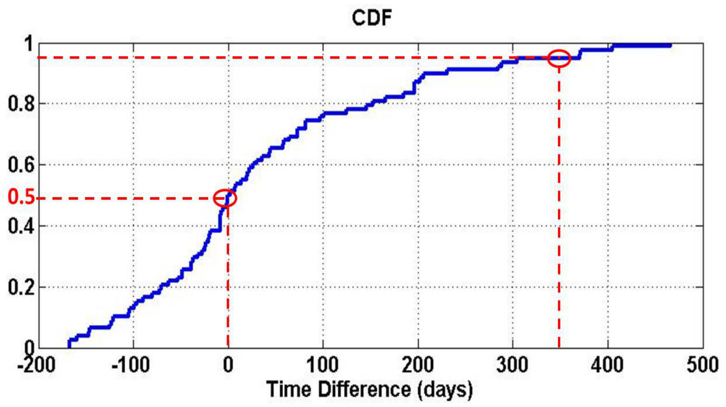


Figure 5.4: Detection time difference distribution

5.5.5 Inferring Results

We evaluated the performance of the inferring component for each month. We used *all* the detected malicious entrances as the seeds without removing any false positive cases. We believe this provides the most realistic and accurate assessment of how VisHunter would perform when used in practice.

To quantify the effectiveness of the inference results, we considered both the number of correctly inferred malicious servers, termed as "Hit Number", and the ratio of the hit number to the total number of inferred servers termed as "Hit Rate". Thus, a higher hit number indicates that the propagation component can catch more malicious servers, and a higher hit rate indicates the algorithm can infer malicious servers more accurately.

In general, the server with high malicious score (low reputation) and low visibility score (close to malicious infrastructures) is likely to be malicious. We used July data as the training set to choose the appropriate selection sizes for malicious score and filtering size for visibility scores. In particular, after the propagation converges, VisHunter will output top 600 servers with the highest malicious scores ($Top600_{Mscore}$) as selection size and top 1000 servers with highest visibility scores ($Top1000_{Vscore}$) as filtering size. Those servers that appear in $Top600_{Mscore}$ but *not* in $Top1000_{Vscore}$ will be inferred as malicious servers. This way, VisHunter achieved a hit rate around 70%.

Figure 5.5 shows the propagation results for the July data with different selection sizes and filtering sizes.

Varying Filtering Size based on visibility score (S_v). As can be seen from Figure 5.5(a), when filtering size increases, the hit rate also increases. For instance, assuming selection size is $Top600_{Mscore}$; if VisHunter filters top 1000 servers with the highest visibility score (filtering size is $Top1000_{Vscore}$), it can achieve a 65% hit rate. If the filtering criterion is set to top 20000 servers (filtering size is $Top20000_{Vscore}$), the hit rate will increase to 77%.

Moreover, as shown in Figure 5.5(b), the hit number decreases with the increase of the filtering size based on visibility score. This is mainly because we also filter some high reputation compromised servers.

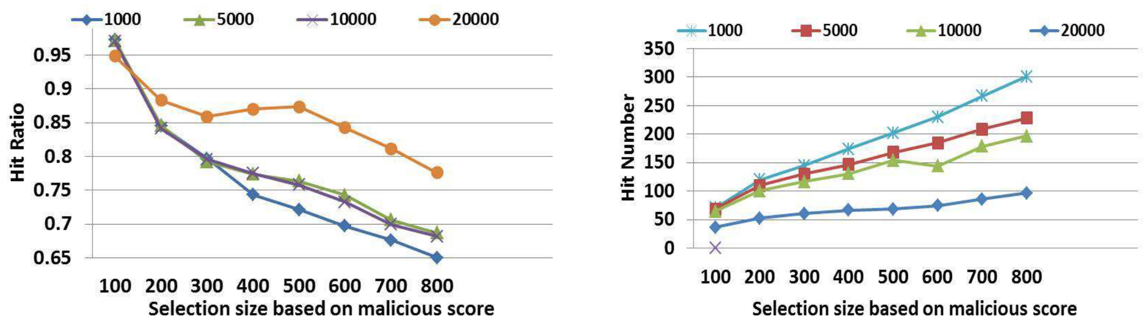
Varying Selection Size based on malicious score (S_m). Figure 5.5 also shows that with an increasing selection size, more malicious servers could be identified by VisHunter. More specifically, assuming filtering size is $Top1000_{Vscore}$; the number of malicious servers detected by VisHunter will increase from 71 to 295 when the selection size varies from top 100 ($Top100_{Mscore}$) to top 800 ($Top800_{Mscore}$) servers with the highest malicious score.

Table 5.8: Propagation results

Inferring Results	2013				2014	
	Jul	Aug	Sep	Oct	Nov	Dec
Inferred Servers	328	293	380	284	405	366
Visible servers	64	56	77	68	61	94
Invisible servers	264	237	303	216	344	272
Malicious servers	229	186	238	202	206	225
Hit rate	69.8%	63.5%	62.6%	71.1%	50.9%	61.5%

In addition, we can see from Figure 5.5(a), the hit rate will decrease when we increase the selection size, because we include servers with lower malicious scores which have a high chance to be legitimate.

In fact, figure 5.5 represents a typical trade-off between true positives (detected malicious servers) and false positives (misclassified benign servers), with two selection parameters governing the balance between them. To achieve a good trade-off, we use July data as the training set and select the appropriate selection sizes for S_m ($Top600_{Mscore}$) and filtering size for S_v ($Top1000_{Vscore}$).



(a) Hit Ratio (b) Hit Number

Figure 5.5: Propagation results

Table 5.8 summarizes the results for each month. On average, VisHunter was capable of finding 6-9 times more malicious servers on top of the malicious seeds, with around 65% hit rate. We further clustered the inferred malicious servers based on their visibility. We do not distinguish compromised servers and malicious servers here since VirusTotal does not provide such labels. However, our manual study shows that those visible servers were likely to be legitimate servers that were compromised or abused by attackers. On the other hand, those invisible servers were often part of attackers' malicious infrastructures.

5.5.6 Case Study

Next, we present some case studies detected by VisHunter.

Case Study of the Detection Component. Most of the malicious redirections detected by VisHunter were from one visible server to one invisible server. Since VisHunter does not require a diverse set of clients for detection [99], it can even detect the malicious infrastructure even only one client accesses it. One interesting case VisHunter detected was `Onlinefwd.com` campaign [54], which was associated with adware and browser hijackers. In our data, five domains redirected users to `Onlinefwd.com`, and all of those five domains were for sale. When we accessed them recently, both `saplab.org` and `ruby.runpaint.org` redirected users to fake AV websites.

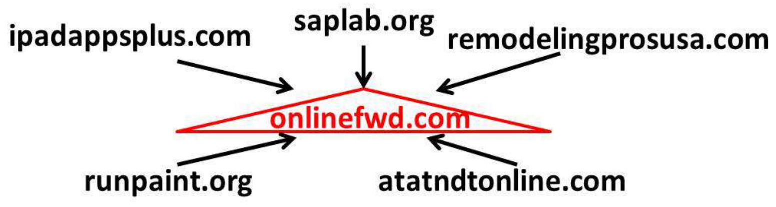


Figure 5.6: Onlinefwd.com campaign

Table 5.9: Typosquatting campaign

Domain	True domain	$Rank\_Vscore$	$Rank\_Mscore$
anvidea.com	nvidia.com	15666	342
carigslist.com	craigslist.org	13324	355
yourube.com	youtube.com	14587	396
creaers.net	creaders.net	16635	416
fandago.com	fandango.com	16647	417
...			

Case Study of the inferring Component. Since our propagation graph is built from both redirections and inner relationships, VisHunter can find more various types of malicious servers under the malicious cyber infrastructure. One interesting campaign VisHunter found was a typosquatting campaign. There were totally 21 typosquatting domains involved in the campaign and they all shared the same IP address. Table 5.9 lists the malicious typosquatting domains and their corresponding correct domains. All of these servers have high malicious scores and low visibility scores. Further investigation shows that these servers were propagated with high malicious scores because an infected client visited these typo servers. Moreover, since these typosquatting servers shared the same IP address, they also propagated the malicious score to themselves. Some of the domains were already expired, e.g., `anvidea.com`, while other, such as `yourube.com`, which was registered on 2005, were still redirecting users to fake AV servers.

5.5.7 Comparison with Existing Work

Unlike VisHunter, existing work on malicious redirection detection either target on specific attack channels or require a large and diverse user base, which limits their practicality. In this section, we quantitatively compare VisHunter with existing work SURF [88].

SURF makes use of 9 features to detect malicious servers involved in a search poisoning attack. Three of them belong to poisoning resistance, which is only applicable for a search poisoning attack. Besides these attack-specific features, there remain six features. Since all the redirections in VisHunter from visible servers to invisible servers are already cross-site redirections, we ignore "total redirections hops" and "cross-site redirections hops". In fact, as discussed, these two features could be easily manipulated by attackers who have a control over the compromised and/or malicious servers. Since the information about "page to load/render errors" was not available in our dataset, we implemented a classifier based on the remaining three features for comparison.

Unfortunately, SURF missed all the malicious redirections because most benign and malicious cases shared similar features. 72.47% of benign redirections had different locations, and 39.97% benign redirections also redirected from domain names to IP addresses. Moreover, none of the malicious redirections used cloaking techniques. We acknowledge that our comparison might not be comprehensive enough to draw a solid conclusion partially due to the fact that we were not able to completely reproduce SURF's classifier, and the main goal of SURF was to detect a search poisoning attack rather than general malicious redirections. Nevertheless, it is worth to note that redirection features alone are subject to circumvention by attackers, and leveraging visibility as a complementary feature allows VisHunter to achieve better detection accuracy and to be more robust against manipulation.

5.6 Discussion

Overhead: The most significant overhead in VisHunter is the visibility checking on the search engines. However, as shown in Section 5.5.2, visibility of servers is not changed frequently. In other words, we do not need to check the visibility of all the servers every day.

Limitation: For some visible malicious servers hosted on compromised servers, VisHunter may not guarantee to detect them. However, since those visible malicious servers are indexed by search engines, they become good targets for existing work, such as EvilSeed [76] and PoisonAmplifier [78], which explore the shared patterns among the malicious servers and use search engines to find them.

Evasion: An attacker who gains the knowledge of VisHunter may attempt to circumvent it by either manipulating the visibility of malicious servers or misleading the VisHunter classifier.

To manipulate the visibility of malicious servers, attackers can make their malicious domains be public leading to malicious redirection from visible servers to visible servers, which will be filtered by VisHunter. One way to promote malicious domains to be public is to inject them into other compromised servers. However, this will make them easily to be detected by the administrators of those compromised servers. In addition, researchers can easily find all of the compromised servers by searching the malicious domain in search engines. In addition, if cyber criminals directly submit their domains to search engines, such malicious servers will not link with other benign servers. Therefore, we can use other features such as the number of search results, to assign some weights to the server visibility. Malicious servers will be visible with fewer weights.

To mislead the classifier, as discussed in Section 5.4.2, those features can not be easily evaded by attackers without causing a significant amount of cost. Therefore, even some adversaries may still find ways to bypass VisHunter, the resource constraints would limit the effectiveness of the adversaries' campaigns or raise the higher cost for them.

6. INFERING PROMOTING SERVERS ON COMMENT SPAM*

We have introduced PoisonAmplifier, SMASH, and VisHunter. Although SMASH and VisHunter together can detect all types of servers in our proposed malicious cyber infrastructure model. However, these two systems are needed to be deployed at the edge of the network to passively monitor the network traffic. In this chapter, we study malicious cyber infrastructure from a new perspective, the servers promoted by spammers. These promoted servers could be any type of servers.

Spamdexing (also known as web spam, or search engine spam) [74] refers to the practice of artificially improving the search rank of a target website than it should have. The rise of such spam causes unnecessary work for search engine crawlers, annoys search engine users with poor search results, and even often leads to phishing websites or malware drive-by downloads. In a recent study [40], Google reports about 95,000 new malicious websites every day, which results in 12-14 million daily search-related warnings and 300,000 download alerts.

To boost the ranking of the target websites, spammers have already developed lots of spamdexing techniques [74] in the past few years, most of which also called Black Hat SEO (Search Engine Optimization). Text and Link manipulations are two main SEO techniques frequently abused by spammers to exploit the incorrect application of page rank heuristics. Through injecting excessively repeating contents in their websites or changing the perceived structure of web graph, spammers have successfully improved their search ranks in the past [13]. However, since search engines have already developed new techniques to detect these content/link manipu-

*Part of this chapter is reprinted with the permission from "NeighborWatcher: A Content-Agnostic Comment Spam Inference System" Jialong Zhang and Guofei Gu. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium(NDSS'13)*, Copyright © 2013 by Internet Society.

lation tricks [17, 19], spammers begin to change to another new trick of spamdexing, named comment spamming. Comment spamming refers to the behavior of automatically and massively posting random comments or specific messages (with links to some promoting websites) to a benign third-party website that allows user-generated content, such as forums (including discussion boards), blogs, and guestbooks. In this chapter, we refer to those benign victim websites that are frequently used by spammers to post comment spam as *spam harbors* (or sometimes *harbors* for short). This new trick of comment spam can benefit spammers in several ways: (1) spammers can easily inherit some reputations of these harbors with nearly zero cost; (2) the risk of being detected by search engines is reduced; (3) spammers can easily disguise themselves as normal visitors who also contribute content.

In this chapter, starting from a seed set of collected spam, we first perform a measurement study on spam harbors' quality and the graph structure of spam harbors, which reveals the structure of spammers' infrastructure. We find that spam harbors usually have relatively low qualities/reputations, which is quite counterintuitive because spammers are expected to spam on high-quality harbors to maximize their profits. To compensate the low reputations of these harbors, spammers intend to keep using a large variety of harbors for spamming. As for the structure of their spamming infrastructure, we find that spam harbors in the same campaign always post similar spam links at the similar time, which reflects that spam harbors in the same campaign always have close relationships while normal websites do not necessary have such relationships.

Based on observations from this measurement study, we design a system named NeighbourWatcher. Our intuition is that if the promoting link in a comment also appears in the neighbors (cliques in the spam harbor infrastructure) of this harbor, it has a higher possibility of being a spam message, because normal links are not necessary *always* been posted on *the specific set* of harbors that have been verified to be exploited by the same spammer/campaign. NeighbourWatcher uses a graph-based propagation algorithm to characterize neighborhood relationships among collected spam harbors in order to infer the spamming infrastructure. When a new comment is posted with some link, NeighbourWatcher performs a *neighborhood watch* on the graph and calculates a suspicious score based on the graph-based inference. With a prototype implementation running on a real-world dataset, we show that NeighbourWatcher can keep finding new spam and spam harbors every day.

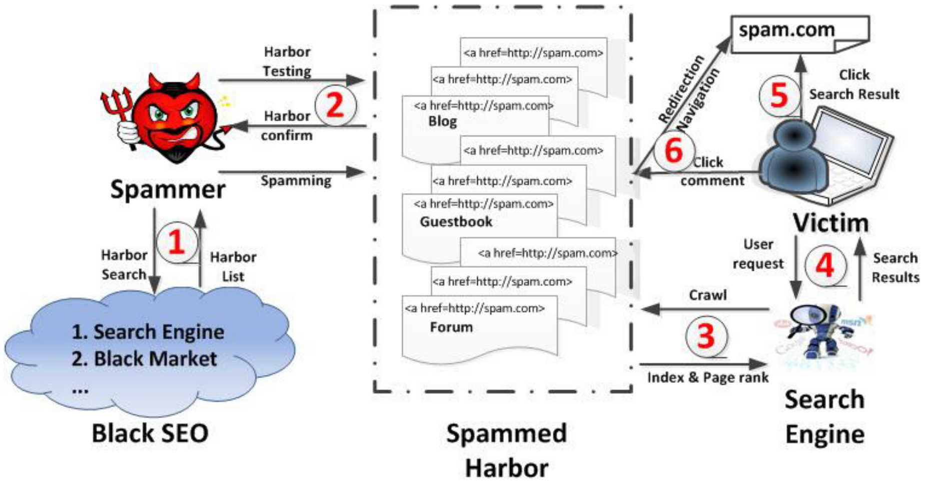


Figure 6.1: The workflow of comment spamming

6.1 Problem Statement

Most existing state-of-the-art approaches to detect comment spam use three types of features: (1) content-based features [80, 89], e.g., the utilization of certain words, content redundancy/frequency, topic or language inconsistency; (2) context-based features [91], which mainly refer to the existence of URL cloaking or redirection techniques (because normal URLs rarely use them); (3) behavior-based features [96, 100], which mainly refer to the time difference between main article and comment posting (which is typically short in normal cases). Unfortunately, all of them have their clear limitations and spammers already become agiler in evading them. To evade content-based detection [80, 89], spammers can artificially manipulate their posting content by mixing spam links with normal content, or posting very few content on each harbor site but spamming on a large variety of harbors. Thus they can easily disguise themselves as normal visitors, and remain not detected. In [91], the authors used context-based features, i.e., checking the use of URL cloaking or redirection tricks, which is effective for the certain type of spam. However, it has a low coverage in the detection scope, because most comment spam currently is mainly used for search rank manipulation [96]. Thus, URL hiding/cloaking is no longer necessary and less used. The limitation of time differences between main article and comment posting [96, 100] is also clear; it is applicable to only some blogs, which are time sensitive, while not easily applicable to more broader websites such as forums, guestbooks. As we can see, within these three types of features, the later two typically may not work well alone, thus they are suggested to combine with content-based features. Finally, we note that another limitation for content-based detection is that the training overhead is typically high; it needs to be trained and updated constantly (with the fast-changing web contents) and it has to be customized specifically for each *individual* website.

Complementary to previous work on comment spam detection, we approach the problem from a new perspective, i.e., we focus on exploiting the structure of spamming infrastructure, an essential component of comment spamming (which is relatively stable), rather than the content (which changes frequently). The intuition behinds structure-based inference is that, while spamming content can be dynamic, spamming campaigns and spamming structure are much more persistent. If we can recognize spamming patterns that characterize the structure of spammers' infrastructure, then we can continue to detect spam even if the spammers frequently change their spam content.

6.1.1 Threat Model

Driven by profits, spammers always want to take full advantage of their resources (e.g., spam harbors). Usually, there are two ways to achieve this goal: massive spamming on a few harbors, or posting few on each harbor but spamming on a large variety of harbors. If spammers post similar spam content massively on a few harbors, it is easy to be detected by content-based detection systems. However, if spammers post spam on a large number of harbors, the chance of detection at the individual harbor is reduced. In particular, spammers typically keep using their familiar harbors, because these websites may not have effective sanitization/filter mechanisms or posting on them can be easily automated, thus making a comfort zone for the spammers.

As illustrated in Figure 6.1, in comment spamming, spammers typical need to first find out suitable harbors, e.g., those with good reputations, those that can be automatically spammed, or those that have weak or no sanitization mechanisms. To achieve these goals, spammers usually use different Google dorks [6] to find target harbors or simply buy some harbors from underground markets. In this way, they can get a set of harbors that can be used to automatically post spam (labeled as ①). These collected harbors are usually some forums, blogs, or guestbooks that support user contributed content, and normal users typically contribute a lot to them. In this case, spammers can easily disguise them as normal visitors, which makes content-based detection inefficient. Then spammers need to verify these collected harbors to assure that they can automatically post spam on these harbors without being easily blocked. They can do this by posting random normal content. After verification, spammers begin to spam on validated harbors in a large scale (②). As a result, when search engine bots crawl these harbors, they will index the spam URLs posted in these harbors, which can finally improve the search rank of the promoted websites in the spam (③). Thus, when users search certain keywords through search engines, those promoted spam websites may have higher search ranks than they should have (④), which may lead victims to click spam websites in search results (⑤). Or victims may directly click the embedded spam links when they read those spam comments, which will directly lead them to spam websites (⑥)

6.1.2 Categories of Spam Harbors

As described in Section 6.1.1, to launch efficient comment spam, spammers need to carefully choose their spam harbors. Next, we describe three most common types of harbors frequently used by comment spamming.

Web blogs are typically websites where people can post articles. Usually, these blogs have comment areas for each article, which allow visitors to comment on corresponding articles. Thus, spammers can also use these comment space for spamming. A possible detection feature of such spamming is that spammers may post repeating spam comments, and also the time difference between the article and the spam comment could be longer than the normal case [96].

Forums are typically websites where people can have conversations in the form of posting messages. Since most forums need users to register beforehand, spammers can also post their spam as long as they can automatically register accounts for these websites. Since some of the conversations could last for a long time, it is hard to detect this kind of spam based on the posting time.

GuestBooks are typically platforms to let visitors to communicate with corresponding websites. Most companies websites have their guestbook pages to let people leave a message to their companies. GuestBooks are agiler than blogs and forums because there are no normal timing patterns and no conversations; everyone can leave any message anytime with fewer restrictions.

Table 6.1 summarizes the effectiveness of existing different types of detection features on different types of harbors. We can see that content-based and behavior-based features are relatively effective for blog spam. Articles in blogs usually have specific topics, thus it is easy to detect spam whose content is not related to articles. Also, it is less common for normal users to comment on an out-of-date article in a blog. Unfortunately, most blogs still do not have any such detection system, which leaves them still to be a good platform for spammers. In the contrast, there are typically no specific topics in guestbooks; everyone can leave any message at any time. Thus guestbook spam is hard for all existing detection features. Context-based features, i.e., detecting URL cloaking, do not have good results on all the harbors because of the very limited effectiveness scope. In short, as we can clearly see that existing detection features are certainly not enough in fighting comment spam. The need for effective and complementary techniques is pressing.

Table 6.1: Effectiveness of different types of detection features

Features	Content	Behavior	Context
Blogs	good	good	bad
Forums	medium	medium	bad
GuestBooks	bad	bad	bad

6.2 Spam Harbor Measurement

Comment spamming, as a relatively new trick of spamdexing, has been reported for quite a while, ever since 2004 [11]. However, until recently, comment spamming has not been sufficiently studied, and existing approaches are clearly insufficient as discussed earlier. To gain an in-depth understanding of the comment spamming, we study the problem from a new perspective, i.e., spam harbors, which form the basic infrastructure for spammers. Why spammers choose these harbors? Are there any special properties of these harbors? Can we use these properties to help defend against comment spam? In this section, we will try to answer these questions and present what we have learned from these harbors.

6.2.1 Dataset

To collect spam harbors, we started from 10,000 verified spam links S_{study} (which are collected from our previous work [113]) and collected a dataset containing 38,913 real-world spam harbors, which are represented with unique domain names. Specifically, we searched all these spam links in Google and collected the search results. Among these search results, not all of them are spam harbors, e.g., some are security websites that report those search links as spam, and some are benign websites[1] that link to those search links. However, we observe that spam harbors typically contain embedded hyperlink tags (e.g., anchor tag $< a\ href = "..." >$ and BBCode $[URL]...[/URL]$). This is because spammers perform automated posting on massive websites, and typically they are unsure whether their target spam harbors support embedded links or not. Thus, in order to achieve a high success rate of posting the spam links, they choose to use embedded hyperlink tags [97]. Based on this obser-

[1]Since some of those search links are compromised benign links, they may also be linked by other benign websites.

vation, we only extracted those search results with embedded hyperlink tags in their contents as spam harbors.[2]

For each spam harbor, starting from the page that contains our verified spam links, we further crawled all possible pages on that website and recorded timestamps on the page (typically these webpages always record the time when a message is posted). Table 6.2 provides an overview of our collected data. To study the differences among three categories of harbors, we roughly group the collected harbors based on their URLs. That is, if a harbor URL contains keywords such as "blog","forum","guestbook", we will cluster them into blog, forum, and guestbook category, respectively. We do not further distinguish remaining harbors without clear keywords (listed as "other" in Table 6.2), and treat them as the mixing of these three categories. Among 38,913 spam harbors returned by search results, 35,931 are still active so we can crawl further pages on these harbors. We have crawled more than 9 million postings in total.

Table 6.2: Data collection of comment spam harbors

	blog	forum	guestbook	other	total
# of search results	27,846	29,860	31,926	500,717	590,349
# of harbors (domain)	4,807	2,515	3,878	27,713	38,913
# of active harbors	4,685	2,185	3,419	25,642	35,931
# of postings	532,413	640,073	1,469,251	6,497,263	9,139,000

6.2.2 Quality of Harbors

Since the goal of comment spamming is to improve the search ranks of spam websites, the higher quality spam harbors have, the more effective comment spamming

[2]Note that we may not extract a complete list of harbors in this way. Instead, our conservative goal here is to extract spam harbors with a higher confidence.

is. In this section, we try to evaluate the quality (e.g., reputation) of these spam harbors in the following three perspectives.

PageRank Score is calculated by PageRank algorithm [35], which is widely used by search engines to rank the importance of websites. A high PageRank score indicates a better reputation of the website, which can lead to a high search rank. To evaluate the overall quality of spam harbors, we use PageRank scores of spam harbors as an indicator of the quality of them. We randomly choose 1,000 spam harbors in each category, and use Google toolbar [16] to automatically request PageRank scores of these spam harbors. Figure 6.2 shows the PageRank scores distribution of these harbors.

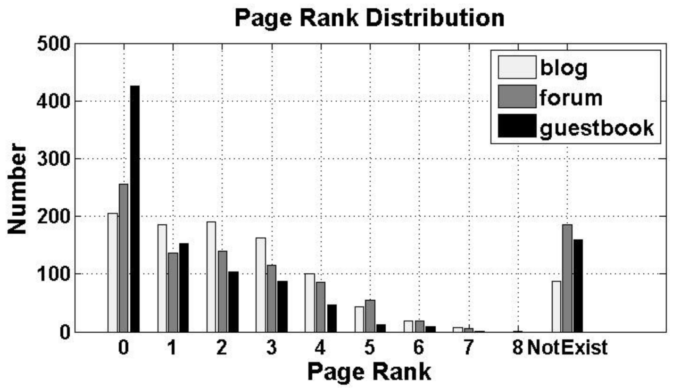


Figure 6.2: PageRank score distribution of harbors

We can see that spammers target on both high-reputation and low-reputation harbors. From the graph, less than 20% harbors have a PageRank score higher than 3, which is the average PageRank score based on [53]. The reason is mainly because that websites with high PageRank scores usually have stronger spam sanitization mechanisms or more strict posting rules, which make it harder for spammers to keep automatically spamming on these websites. In addition, about 40% guestbook harbors have PageRank scores of 0, because most of them are small company websites that do not have notable reputations. However, spam links can still inherit and accumulate some reputation from a large number of such harbors. At least, they can use this way to let search engines to index them.

Life Time is defined as the time interval between the posting time of the first spam and the recent spam (based on our crawled dataset). Spammers tend to find some stable harbors that they can keep using. Thus, a long life time should be a good indication of high quality for spammers. Since there is no ground truth for the first spam and last spam, we randomly choose 100 harbors in each category and manually check their postings. Since we may not be able to crawl all the pages inside a given harbor, our estimated life time based on the limited crawled dataset is simply a lower bound. Figure 6.3 shows the distribution of lifetime of spam harbors.

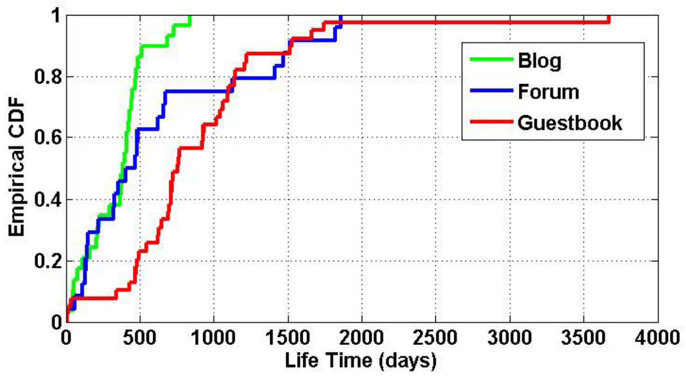


Figure 6.3: Distribution of harbor life time

We can see that for both blog and forum harbors, more than 80% harbors have a long lifetime more than 1 year. And more than 70% guestbook harbors have a lifetime longer than 2 years. During the manual checking process, we found that for most harbors, the initial postings are benign but later these harbors are frequently exploited by spammers for spamming. Especially for guestbook harbors, almost all the later postings are spam, which confirms that these spam harbors are kept being used by spammers for a long time.

Google Indexing Interval is defined as the time difference between two consecutive Google crawling (indexing) time of the same spam harbor. To reduce the crawler overhead but also keep pace with dynamically updated web pages, search engine bots need to periodically crawl websites. Thus, there always exist a time lag between posting time and search engine indexing time. A shorter time lag (indexing interval) should be a sign of a high quality for spammers, because search engines can quickly index the new spam. Google Cache[14] contains the time when Google bot crawled the web page. Thus we randomly choose 100 active spam harbors in each category [3] and crawl their cache pages every day. Figure 6.4 shows the distribution of Google indexing interval.

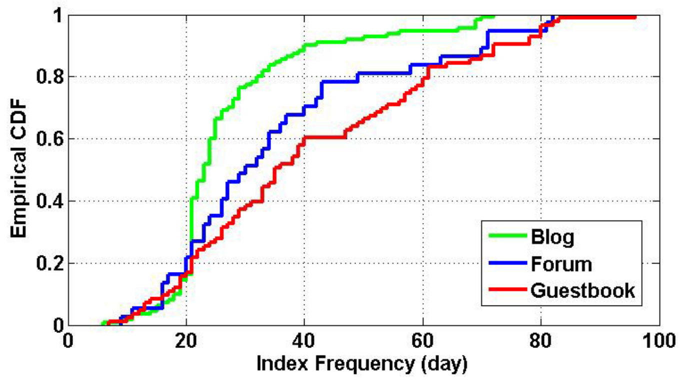


Figure 6.4: Googe indexing interval

[3]Note that Google Cache has a request limitation per day. Thus we only choose 100 harbors in this test. Also, Google does not cache all web pages, so we choose those pages that are cached by Google

Compared with guestbook harbors, blog and forum harbors have relatively shorter indexing intervals because normal postings on them update much more frequently than postings on guestbooks. Thus, Google bots crawl blog/forum harbors much more frequently than guestbook harbors. However, still, nearly 80% of all harbors have an indexing interval larger than 20 days, which indicates that the overall indexing frequency is still not too high.

Lessons learned: Although high-reputation harbors should be the best choice for spammers, high-reputation websites usually have more strict spam filtering mechanisms or have stronger authentication systems, which makes it harder for spammers to automatically spam on them. Thus, spammers tend to *keep using* a large number of harbors for spamming *regardless of their reputations* to compensate the relatively poor quality of individual harbors. However, our observations also convey a positive message to the defenders: since there typically exists *a long time lag* between spamming time and search engine indexing time, if we can detect these spam before search engines index them, we can still efficiently prevent comment spamming from impacting search ranks.

6.2.3 Spam Harbors Infrastructure

After finding out qualified spam harbors, spammers intend to take full utilization of these harbors for spamming. In this section, we study how spammers utilize these harbors for spamming, and what are the relationships that spammers formed on these harbors.

6.2.3.1 Relationship Graph

To reduce the possibility of being detected, and also to take full utilization of their spam harbors, spammers tend to distribute their spam among multiple spam harbors. Thus, different spam harbors may *always* share similar spam, because spammers intend to recycle these harbors. This special close relationship among these harbors, which may rarely occur in the normal case, gives us a chance to study the spamming behaviors of spammers. To characterize such relationship, we build a relationship graph $G = (V, E)$ among these spam harbors. We view each spam harbor as a node v and build up an edge e between two spam harbors if they share same spam (links) in their postings. The resulting graph G for our entire database consists of 13 connected subgraphs, each of which has more than two nodes. The largest connected component G_0 contains 97 % spam harbor domains.

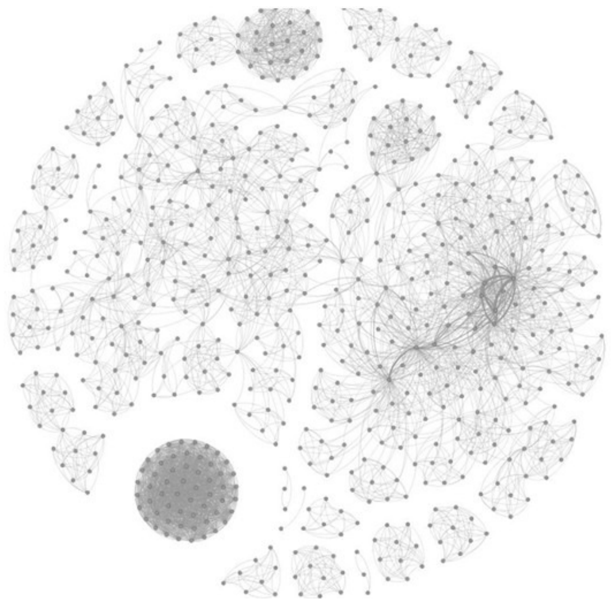


Figure 6.5: Relation graph of spam harbors

Figure 6.5 is a partial visual representation of G_0. We can see that there exists a large number of *communities* within G_0, i.e., a group of nodes *closely* interconnected with each other and only loosely interconnected with other communities. Here, each community represents a set of harbors in close relationships with each other, possibly used by the same spammer. In addition, although different spammers may have different harbors, there always exist some harbors shared by multiple spammers, which provides us a good opportunity to find more other harbors (even starting from a small seed set).

6.2.3.2 Spam Harbor Community

In the spamming process, spammers first need to choose spam harbors to post their spam, and then need to distribute their spam on these selected harbors. In this section, we will study how spammers choose their harbors, and how they distribute spam to selected harbors.

Choosing Spam Harbors. In this part, we analyze how spammers choose spam harbors, i.e., we examine how many harbors are used for spamming each time. Some spammers may spam on *all* their available harbors to fully use their resources, and some may only sample parts of their harbors for spamming to avoid the exposure/detection of all their resources. To measure how spammers choose harbors each time, we define a metric named "Distribution Ratio", which is the ratio of the number of harbors posting same spam over the number of harbors in their community [7]. Thus, a higher Distribution Ratio indicates that spammers tend to fully use their spam harbors for spamming. Figure 6.6 shows the distribution of Distribution Ratio for S_{study}.

We can see that 80% of spammers tend to use only less than 50% of their spam harbors for the same spam. In this way, they can reduce the possibility of all their resources being detected/exposed. However, since spammers always have a limited number of harbors, to keep spamming, they have to recycle these harbors. As a result, we will finally observe a relatively stable relationship among harbors.

Distributing Spam. After selecting spam harbors, spammers need to decide

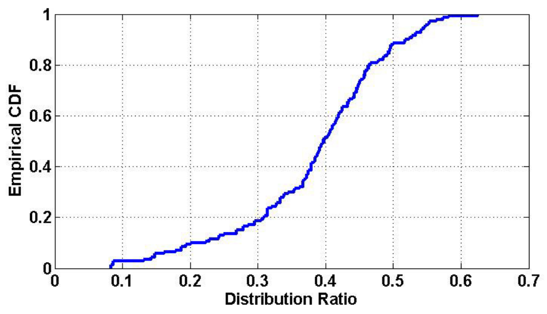


Figure 6.6: Spam distribution ratio

how to distribute their spam to these selected harbors. For example, some spammers may post the same spam on their selected harbors at a similar time. In that case, posting time on these harbors should be similar. Other advanced spammers may choose to distribute different spam on different selected harbors. In this study, we simply consider two spam messages are posted in a similar time if they are posted in the same month. To measure the similarity of spam posting time, we design a metric, named "Time Centrality Ratio", which is the ratio of the maximal number of harbors that post the spam in the same month over the total number of harbors that post this spam. The intuition here is that if all the harbors post a spam message in the same month, it is possible that this spam is distributed to all selected harbors. Otherwise, spammers will distribute different spam to selected harbors. Thus, a high Time Centrality Ratio indicates that spammers distribute the same spam to most of their selected harbors at a similar time. Figure 6.7 shows the distribution of Time Centrality Ratio of S_{study}.

We can see that about 60% spam have a high ratio larger than 0.6. This means that about 60% spam is distributed by spammers to more than 60% of their selected harbors in one month for spamming.

Lessons learned: To efficiently utilize spam harbors, spammers intend to keep

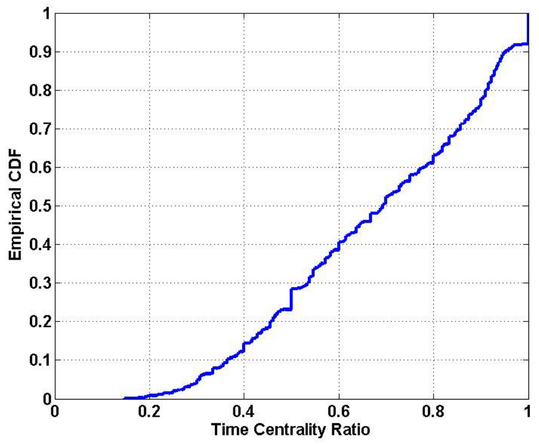


Figure 6.7: Distribution of time centrality ratio

utilizing the spam harbors from a relatively stable set (pool) that they own. Thus, essentially spammers build an artificial relationship among these spam harbors, which is considered as their spamming structure. In addition, since spammers have a limited number of harbors, they must use/recycle these harbors with a large scale of spamming to maximize their profits. Also, although different spammers may have different strategies to find their harbors, there always exist some intersections among them, which gives us a chance to find more other spam communities even starting from a small seed set.

6.3 Inference System Design

In this section, we present a brief overview of our inference system, then describe in details its two core components: building spamming infrastructure graph and spam inference.

6.3.1 Overview

From the measurement study in Section 6.2, we can see that if a link (in a comment) is posted on a set of harbors that have a close relationship (e.g., within the

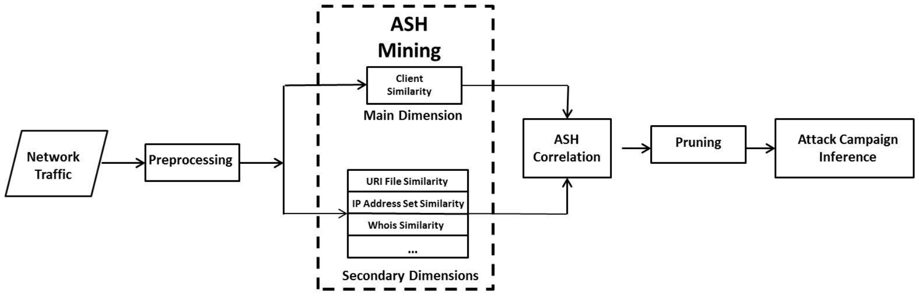


Figure 6.8: System architecture of NeighbourWatcher

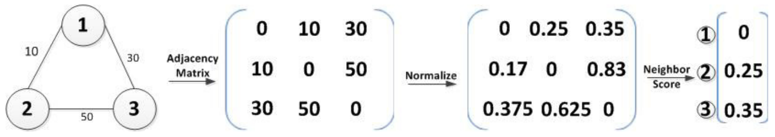


Figure 6.9: Normalized neighborhood relationship matrix

same spam community in the infrastructure graph) at a similar time, it has a high possibility to be spam. Following this intuition, we design a spam inference system, named NeighbourWatcher. NeighbourWatcher infers comment spam purely based on the links promoted in the postings, ignoring other content information. An overview of NeighbourWatcher is shown in Figure 6.8. In practice, NeighbourWatcher keeps monitoring (and updating) spam harbors in our database and builds the spamming infrastructure graph based on their posting history. In the inference phase, when a new post is given, NeighbourWatcher extracts the embedded links, and also finds out the websites that have been posted with the same links (we call the set of these websites as a "real posting structure"). Based on the spam infrastructure and the real posting structure, NeighbourWatcher calculates a suspicious score to tell how likely this is spam. Next, we will describe the algorithms of building our spamming infrastructure graph and inferring comment spam.

6.3.2 Building Spamming Infrastructure Graph

From Section 6.2, we know that spammers always have their own preferred spam harbors, and they intend to keep utilizing these spam harbors for spamming. Thus if multiple harbors always share similar postings in their history, it should be a good indication that they are exploited by the same spammer for spamming, and also have a high probability to be spammed by the same spammer in future. In this case, if we find a new posting occurs on these harbors at a similar time, we could infer this posting as spam with a reasonably high confidence. Following this intuition, we build spammers' spamming infrastructure based on shared postings (in historic data) among these spam harbors. We define the spamming infrastructure (or sometimes we simply use "spamming structure" to denote the same concept) as neighborhood relationships among spam harbors. Thus, spam harbors spammed by the same spammers should have close neighborhood relationships because they always share similar spam postings in history. To quantify such relationships, given an input harbor, we calculate neighborhood scores for all other spam harbors. A higher neighborhood score of a harbor indicates a much closer neighborhood relationship with the given (input) harbor.

To formalize the above intuition, we view all neighbor relationships among spam harbors as a weighted undirected graph $G = (V, E)$, in which V denotes the set of all spam harbors, and each link $(i, j) \in E$ denotes that harbor v_i and harbor v_j share at least one common posting. The weight on the edge should reflect the strength of the relationship between two harbors. In our case, let L_i be the set of postings (represented with their embedded URLs) in node v_i and L_j be the set of postings in node v_j, then we define weight $w_{i,j}$ as $|L_i \bigcap L_j|$. Thus, the more postings two harbors share, the much closer they are. We further normalize $w_{i,j}$ by dividing $\sum_j^n w_{i,j}$ as shown in Figure 6.9.

Next, we design a graph-based propagation algorithm to propagate neighborhood score from the input harbor(s) to its neighbors based the neighborhood relationship graph G. Table 6.3 shows the notations used in our algorithms.

Table 6.3: Notations used in this chapter

W	Normalized adjacency matrix of the neighbor graph
I	Input harbor vector, $I_i = 1$ if i is a input harbor
N	Neighbor score vector. N_i is the neighbor score of harbor i
R	Real spam posting vector. $R_i = 1$ if harbor i posts the same input link
α	Dampen factor. $\alpha = 0.85$
n	The number of spam harbors

Before propagation, we first assign an initial score I_i to each node V_i. For the input harbor j, I_j is assigned with 1, and others are assigned with 0. Then we calculate neighborhood scores for all harbors as follows:

$$N = I \cdot W \tag{6.1}$$

Eq.(6.1) can capture the immediate neighbors of the input harbor. In this case, each immediate neighbor is assigned with a neighborhood score based on the number of common postings shared with the input harbor. The more common postings they share, the higher score they should have. As shown in Figure 6.9, node 3 has a higher score than node 1, because node 3 shares more common postings with the input harbor node 1 in history. However, as we show in Section 6.2, spammers might not always spam on all their harbors for each message, and our observed history relationships may be only a subset of the spammers' real relationships. To illustrate this scenario and demonstrate how we handle this problem in a generalized way, we show a case study in Figure 6.10.

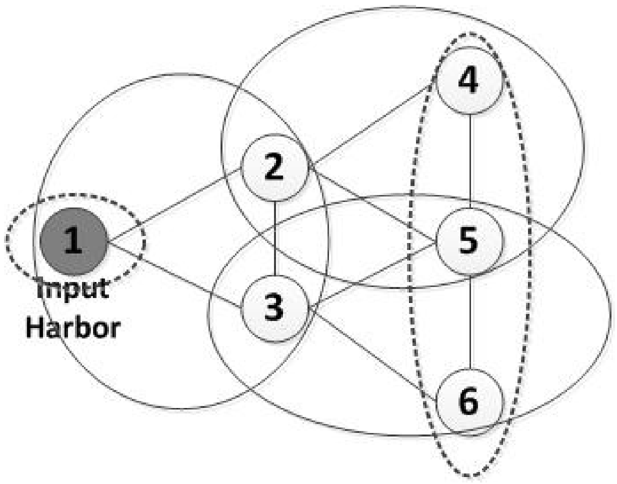


Figure 6.10: A case study of comment spamming on different subsets of harbors

For this example, in the spamming process, a spammer first spams on node 1,2,3 for one spam message. And then the spammer spams on node 2,4,5 and node 3,5,6 with different spam messages. The neighborhood graph based on the history information is shown in solid circles. Now if the spammer spams on node 1, 4, 5, 6 (as seen in dashed circles), applying Eq.(6.1) with node 1 as the input harbor will assign score 0 to node 4, 5, 6, because they are not directly connected with the input harbor node 1. This makes neighborhood score be less efficient to capture potential relationships between the input harbor and other harbors that will possibly be spammed by the same spammer. To overcome this problem, we need to deeply propagate the neighborhood relationship, similar to the page rank algorithm. Specifically, if we propagate the neighbor scores one further hop, this gives us $I \cdot W \cdot W = I \cdot W^2$, which will propagate neighbor scores from node 1 to 4. Thus the average received score for each node in this case is $(I\dot{W} + I \cdot W \cdot W)/2$. Naturally, the propagation scores should decay along with the distance from the input harbor. To achieve this goal, we dampen matrix W by a constant α ($0 < \alpha < 1$).[4] Thus the farther the distance between a harbor and the input harbor, the less neighbor score it can inherit. Based on all of above, we propagate neighborhood scores for each harbor as follows:

$$N = \frac{\sum_{i=1}^{t} I \cdot (\alpha \cdot W)^i}{t} \tag{6.2}$$

Once the neighbor score vector converges after t propagation steps, we can obtain the final (stable) neighborhood scores for each harbor, and this score reflects how close the neighbor relationship is between the input harbor and the corresponding harbors. Next, we will present how to use these scores to infer a given new spam message.

6.3.3 Spam Inference

To infer whether a given new message/posting is spam or not, we also need to crawl other harbors to check if the link in the given posting also appears on them. Thus, we obtain a real posting structure vector R, $R_i = 1$ if harbor i is also posted with the given link. Now we have both real posting structure and neighborhood scores for each harbor, next we present how to combine them to infer spam.

Intuitively, if harbors with high neighborhood scores have also been posted with the same messages, it has a high possibility that the input message is spam. Neighborhood scores reflect the neighbor relationships between other harbors and the input harbor. Thus, if harbors have high neighbor scores, they may have a high possibility to be spammed by the same spammer, which means if we find the input message appears in these harbors, it should have a high possibility to be spam. Thus, we

[4]We empirically set $\alpha = 0.85$ based on [66].

infer the spam by computing how real posting structure and neighborhood scores combine together to contribute to a suspicious spam score. Specifically, we use a modified cosine similarity function $F(R, N)$ to characterize the similarity between the real posting structure and the learned neighborhood relationship in the spamming infrastructure. We define the final spam score for the input message/URL i as follows:

$$Score_i = F(R, N) = \frac{R \cdot N}{\sum_i^n R_i} \tag{6.3}$$

Here, a higher spam score means that the real posting structure matches very well with closer neighbors of the input harbor. Thus we should have a higher confidence to infer the input message as spam.

6.4 Evaluation

In this section, we evaluate our system in two stages. For the first stage, we evaluate NeighbourWatcher regarding its inference stability and effectiveness. We also measure how many new spam and harbors can be inferred everyday, and the topic diversity of these spam postings. In the second stage, we discuss possible applications of our inference results.

6.4.1 Dataset and Ground Truth

To build the neighborhood relationship graph, we use the collected spam harbors in Section 6.2. After that, we keep monitoring these spam harbors every day and extracting new postings from them for spam inference. Also, after inference, we search inferred spam links in Google and use the same way in Section 6.2 to extract new harbors from search results. To evaluate the effectiveness of our inference system, we need to choose true spam links and benign links for testing. For the former, we extract random postings from harbors and manually choose 500 verified spam messages (that contain spam links). To find a normal postings set, we assume domains in Alexa [2] top 20,000 are highly reputable websites that have less chance to be posted in comment spam. Thus, we check how many links in our collected postings have the intersection with these domains. In this way, we get 754 normal postings and combine them with 500 spam postings as our testing dataset S_{test}.

6.4.2 Stability of Spamming Structure

Our system exploits spammers' posting infrastructure (or spamming structure) to infer spam. Thus, if such infrastructure changes frequently, it will make our system less effective. For example, if spammers keep using new harbors for spamming every day, our system cannot infer their spam. To evaluate the stability of the spamming structure, we essentially examine how the neighborhood relationship among spam harbors change over the time. We build the neighborhood relationship graph of spam harbors by setting $w_{i,j} = 1$ to indicate that two spam harbors share at least one common link and $w_{i,j} = 0$ to represent no relationship between two harbors. Then we consider such relationship graph in the time window $[t, t+\Delta t^{5}]$ as the initial spamming structure and the relationship graph in the next window $[t + \Delta t, t + 2\Delta t]$ as the testing spamming structure. Thus, the difference between two structures indicates the instability of the spamming structure. To quantify such instability, we use Hamming Distance [23] to measure the number of changed relationships CR between these two-time window, as shown in Eq.(6.4). Here n is the number of total harbors.

$$CR_i = \sum_{j}^{n} |W_{i,j}^{t+2\Delta t} - W_{i,j}^{t+\Delta t}| \tag{6.4}$$

Thus, a smaller value of CR implies a much more stable spamming structure. For each spam harbor i in our database, we calculate its changed relationship CR_i.

[5] We empirically set Δt 1 month here.

Figure 6.11 shows the distribution of changed relationships for all spam harbors.

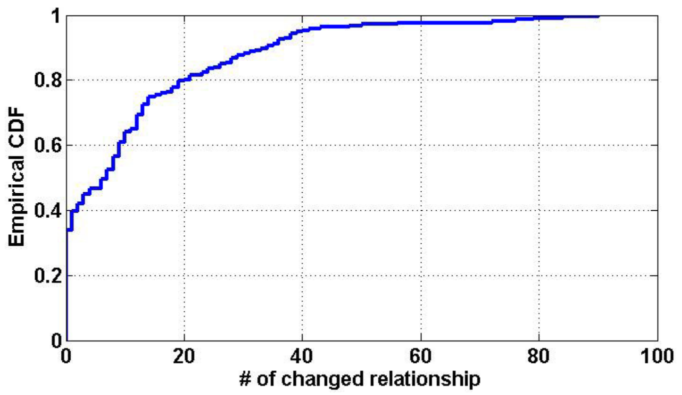


Figure 6.11: Changed relationship distribution

We can see that about 40% harbors do not change their neighbor relationships because spammers keep utilizing the same harbors. In addition, about 80% spam harbors change their relationships less than 20, which is also less than half of the average community/clique size 50. Thus, even if a community loses 20 harbors, we can still infer spam with the remaining 30 harbors as long as spammers keep recycling their harbors. Furthermore, the continuous updating of our harbor database can somehow compensate such instability, we will discuss more this in Section 6.5.

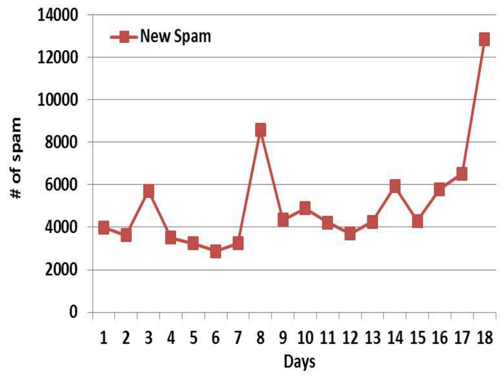
6.4.3 Effectiveness of Inference

To evaluate the effectiveness of our system, we test our system with S_{test}. We consider both the number of correctly inferred spam, termed as "Hit Count", and the ratio of this number to the total number of inferred spam, termed as "Hit Rate" (i.e., accuracy). Thus, a higher value of Hit Count indicates that our system can catch more spam; and a higher value of Hit Rate indicates that our system can infer spam more accurately.

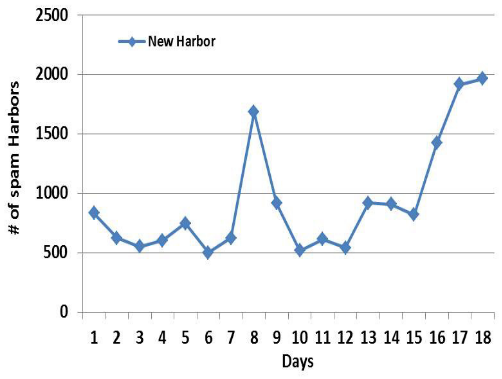
Table 6.4: Sensitivity of the hit rate and hit count to the choice of similarity threshold

Sim. Threshold	Hit Rate	Hit Count	False Positive
0.3	57.29%	432	322
0.4	75.93%	426	135
0.5	**97.14%**	**408**	**12**
0.6	97.8%	360	8

Since we infer spam based on the spam score threshold, we also check how the threshold contributes to the inference results, a way similar to [94]. In our case, a higher (thus more conservative) threshold may lead to a higher hit rate but with a lower hit count. Table 6.4 shows how Hit Rate and Hit Count vary with different settings of the threshold. We can see that a spam score threshold of 0.5 yields to a relatively high hit rate (97%) with a relatively high hit count. Also, there are still 92 links that can not be correctly inferred as spam (i.e., false negatives) and 12 incorrect inferred links based on our labels (i.e., false positives). We further check these links, most of the false negatives only appeared in its input harbor, which means these spam links do not appear in other harbors in our database. This is possible because our database is relatively small and may not cover all spammers' harbors. However, the effectiveness could be easily further improved if one can build a larger dataset, e.g., by recursively updating the spam harbors database, which will be discussed in Section 6.5. Among 12 false positives, 5 are actual benign websites posted by spammers in order to conduct harbor testing (as discussed in Figure 6.1). 7 of them are those link to some Alexa top 20,000 websites and we expected (labeled) them to be non-spam as explained in our test dataset preparation. However, they turn out to be actual spam posted on reputable websites (e.g., in some Google groups). If we exclude them, our actual false positives are only five, which is pretty low. Finally, we note again that our inference algorithm only uses the spamming structure information, and it does not leverage other existing content features yet. Once combined with existing features, we surely can expect a better performance.



(a) New Spam



(b) New Spam Harbors

Figure 6.12: Constancy of NEIGHBOURWATCHER

6.4.4 Constancy

To evaluate the constancy of our system, we essentially examine whether our system can continue finding new spam over time. We keep monitoring spam harbors every day to check new postings. These new postings are submitted to our system for spam inference. For each new inferred spam, we further search it in Google to find new spam harbors (using the same method mentioned in Section 6.2) that are not included in our database, then add them to our database every day.[6] After 3 weeks' study, we have totally inferred 91,636 new spam and 16,694 new spam harbors. Figure 6.12 is the distribution of new inferred spam and new spam harbors over time. We can see that our system can constantly find new spam and spam harbors as long

[6]We may add a few normal websites in our database in this way because our inference hit rate is not 100%. However, we note that these websites most likely will not have close relationships with other spam harbors, thus will not impact our inference results much.

as spammers keep posting new spam and also spamming on new harbors.

Diversity of New Spam. To have a sense of the variety of spam content we inferred, we surveyed 10,000 randomly chosen spam postings and clustered them in 7 categories based on their anchor keywords. Table 6.5 shows the keywords we used for clustering spam, and Figure6.13 shows the category results. We can see that pharmacy is still the most popular spam, and spammers always try to promote them to have a higher search rank [84]. On the other hand, our system can keep capturing general spam (other than just pharmacy) in terms of their spam content.

Table 6.5: Keywords for different spam category

Categary	Terms
Rogue Pharmacy	cialis,viagra,prescription,levitra ...
Rogue software	virus,windows,desktop,software...
Porn	porn,sexy,fuck,adult...
Gambling	casino,poker,roulette,gambling...
Money	insurance,debt,mortgage,credit...
Accessories	handbag,dress,luxurious,louis...

Context-based Analysis of Spam. For those newly inferred spam, we randomly sample 1,000 spam links and use the same method in [91] to find out URL-redirection and cloaking spam. Specifically, we send requests to each link 3 times by setting different HTTP header parameters to emulate Google Bot crawling, directly visiting, and visiting through search result clicking, respectively. We consider it as cloaking spam if any of two visits lead to different websites through redirection. In this case, among 1,000 spam links, we only see 34 spam using cloaking techniques, which also reflects that context-based detection has a low coverage in the face of current comment spam. However, we test these 34 spam links with Google Safe Browsing

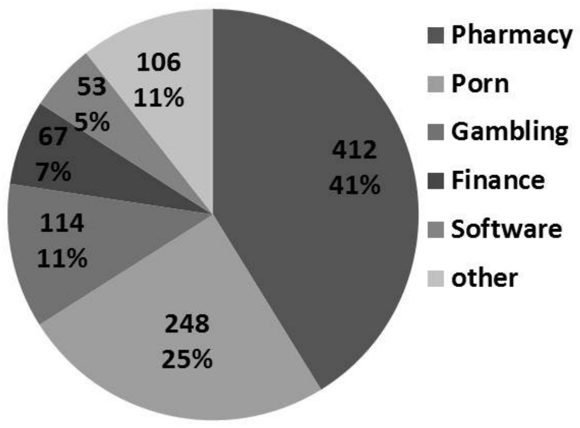


Figure 6.13: Spam category

(GSB) [18], none of them has been reported. Thus context-based detection is still an effective way to find new spam within their limited scope, and our system also can cover such spam (but can detect more other spam that the context-based detection can not).

6.4.5 Applications of Our System

In this part, we will evaluate how our inference results can be applied to provide early warning for search engine bots, and to complement current BlackList services.

Early Warning. "nofollow"[34] is an HTML tag which is designed to instruct search engines that the corresponding links should not be counted for ranking. Usually, different search engines have little different policies in the face of "notfollow" tags. As for Google [38], it will not transfer PageRank or anchor text across corresponding links. In this case, search engines can efficiently filter comment spam if webmasters attach each posting with a "nofollow" tag. However, among 35,931 spam harbors we found, only 4,367 harbors contain such tags, which means spam on other harbors can be successfully exploited for search rank manipulation. Fortunately, according to our measurement study in Section 6.2, there always exists a time lag between the time that spammers post spam and the time search engines index the spam. Thus, if we can detect the spam before Google indexes them, we can also efficiently filter comment spam. To measure this timeliness, we examine the number of "zero-day spam", which is the spam that can not be searched out by Google at the time. Totally we collected 1,364 "zero-day spam" in our test using NEIGHBOUR-WATCHER. Interestingly, when we manually check these "zero-day spam", we find that some spam messages contain randomly generated domains that have not been registered yet. Thus, it is possible that spammers may first promote these links and then register them later based on their promoting results. Figure 6.14 shows the distribution of daily "zero-day spam".

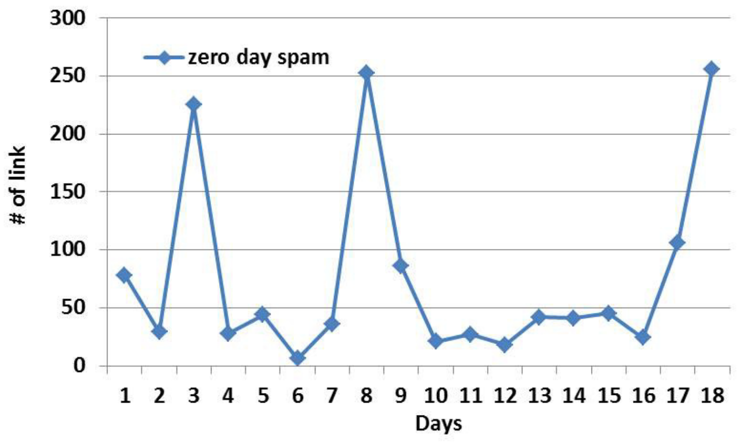
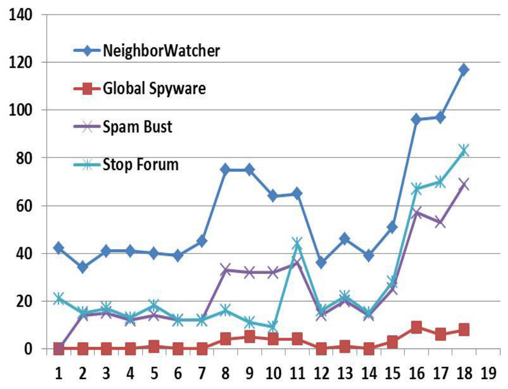


Figure 6.14: Zero day spam distribution

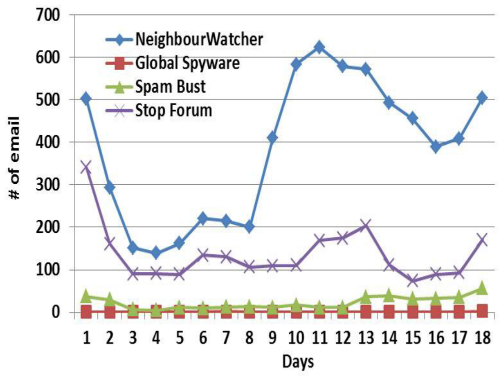
We can see that currently we can only detect few "zero-day spam", because most spammers intend to promote a certain set of spam links that may have already been indexed by Google before (but search ranks were probably not good enough).

However, as long as spammers begin to promote new links, our system can quickly find more “zero-day spam“. In this case, we can give an early warning to search engine bots, or search engine bots can integrate our inference system to help better filter these comment spam.

BlackList. Existing comment spam blacklists usually provide a collection of source IP addresses, register emails, or usernames of spammers. Most of existing online blacklists [12, 42, 44] collect such spammer information by building honey blogs/forums. However, we can early imagine that honey blogs could only collect limited number of spam, thus limiting the effectiveness of such approaches.



(a) Daily IP



(b) Daily Email

Figure 6.15: Daily comparison with existing spam blacklists

To measure our inference approach can complement existing solutions, we compare it with 3 popular online BlackList services. Note that since our system targets on more general spam harbors, not all of the harbors provide complete IP and email information of the posting users in public. Luckily, there does exist some spam harbors in our database that provide IP or email address information. Thus, we can compare these inferred IPs and emails with these 3 Blacklist services.

Table 6.6 shows the comparison result. We can see that for both IP and email, our system can always infer new spammers that are not observed by existing services. Figure 6.15 shows the daily comparison with these 3 BlackList Services. From the figure, each day we can find new spammers that are not labeled by any of these existing BlackLists. In addition, considering the dynamic property of IP addresses, most IP-based BlackLists need to be daily updated. Thus, for IPs detected by these existing systems, we further check their "last seen time", the time of last observation by these existing services. We find most of them are out of date, which means existing BlackLists observe these spammers long time ago but in fact they are still active at the moment. In summary, our system could be a good complement to existing BlackList systems to actively find new spam and to improve the coverage of existing systems. Furthermore, we find some constant email addresses and IPs keep contributing to spam, which also reflects that spammers intend to keep utilizing these spam harbors.

Table 6.6: Comparison with existing blacklist systems

BlackList	# of IP	# of Email
NeighbourWatcher	378	5,945
StopForum	231	1,937
SpamBust	185	122
GlobalSpyware	29	3

6.5 Discussion

Although NEIGHBOURWATCHER shows promise for inferring new spam and spam harbors, there is still much room for improvement. In this section, we discuss several specific improvement areas, and the possible evasions to our current inference algorithm.

6.5.1 Improvement

Combining More Features. NEIGHBOURWATCHER only uses spamming structure information to infer spam, other spamming behaviors could also be incorporated together to help improve the accuracy. For example, we find some spammers post their spam with both $< a\ href = "..." >$ tags and BBcode tags to assure they can embed spam links, because they have no idea about what kind of method the target spam harbors support to embed links. In addition, some spam links are posted on websites with different languages because spammers do not care about harbor's native languages. For example, some spammers post Russian spam in Chinese websites, Korean websites, and English websites, which is extremely unlikely to be normal postings. In this case, we can incorporate these features to improve the overall inference accuracy.

Improving Algorithms. In Eq.(6.3), we assume these postings on different harbors have the same weight. However, postings on different harbors are not necessarily equal. For example, if postings on a harbor have a similar posting time, same email, or same IP address with the input posting, then it has a high possibility that they are spammed by the same spammer at the same time. Thus we could assign different weight to different harbors by considering these factors.

Updating Spam Harbors. In our system, we find new spam harbors by recursively searching inferred spam in search engines. Thus, as long as spammers keep utilizing these spam infrastructures, we can always find out new spam harbors. However, our current system cannot infer those spam posted on only one harbor in our dataset, thus we cannot find out other harbors that are also posted with this spam. In this case, we could use the following strategy. From the search results of the posting link, we attempt to build a relationship graph among returned websites based on whether they already share similar postings (excluding the searched link) on existing pages on the websites. If these websites show very close relationships (e.g., dense connections), the searched link has a good chance to be spam (and thus the corresponding search result websites are possible spam harbors). The intuition here is that although a few normal links may appear on a variety number of websites, it is extremely unlikely that normal users will *keep* posting *similar* postings on certain websites. Our ongoing work will design and test new algorithms to efficiently update our spam harbors dataset.

6.5.2 Possible Evasion

Evasion by exploring new harbors. Obviously, spam harbors that we collected are only the subset of spammers' harbors. Thus if spammers know our collected harbors, they may try to spam on other harbors that are not included in our database, or they may find brand new harbors. In this case, our neighborhood-based inference algorithm could not detect their new structure. However, as long as spammers also post spam on both these new harbors and old harbors, we can still find out their new harbors by keeping updating our database. Otherwise, spammers need to keep finding new harbors and give up existing qualified harbors, which is less likely to happen considering the cost.

Evasion by changing spamming behaviors. If spammers know that we use spam links to build up relationship graphs, spammers may spam different links on different harbors. Thus we can not build up their spamming structure. However, in this case, spammers need to keep finding more harbors to post their variety links, which will increase their cost and time. Otherwise, they need to post the same spam several times on certain harbors (in order to boost search ranks), which will increase the possibility of being detected by content-based detection systems.

Evasion by spamming polymorphic URLs. Our algorithm relies on grouping identical URLs (e.g. we infer a possible spam message if the spamming URL also appears on many of its neighborhood clique harbors). Thus, spammers may try to evade our system with polymorphic URLs (i.e., each URL can be different on different harbors). However, in general, it is not always possible to make full polymorphic URLs for a given spam URL to be promoted. If spammers choose URL shortening services to achieve polymorphic URLs, we can always use the resolved final URLs in our system. Furthermore, we can use the domain of a spam URL instead of the full URL as input, which is relatively stable if spammers want to promote certain domains.

7. SUMMARY AND LESSONS LEARNED

In this chapter, we plan to answer the following questions: How are our systems related to and different from each other? What lessons have we learned from designing these systems? Can we apply the lessons learned from malicious network infrastructures to other platforms such as malicious social network infrastructures and malicious mobile app infrastructures?

7.1 Summary of Our Inference Systems

Table 7.1 provides a brief summary of our systems.

Table 7.1: Summary of our inference systems

	PoisonAmplifier	SMASH	VisHunter	NeighbourWatcher
Host or network based	network	network	network	network
Require multiple infections	N/A	No	No	N/A
Detection target	CS[1]	MA[2]and MB[3]	All	All
Passive or proactive	proactive	passive	passive	proactive
Supervised or unsupervised	supervised	unsupervised	supervised	supervised
Deployment	online service	network edge	network edge	online service

[1] Compromised server
[2] Type A malicious server
[3] Type B malicious server

We can see that these systems have some similar features. They all study the malicious cyber infrastructures from the network level. Therefore, they are robust to malware obfuscation and encryption. In addition, all of them do not require multiple infections or a large diverse user base. Therefore, they are applicable to be deployed at enterprise networks.

Despite the above similarity, these systems are different in several dimensions. In terms of the detection target, PoisonAmplifier only focuses on compromised servers and SMASH focuses on Type A and Type B malicious servers. Although VisHunter mainly focuses on redirections from compromised servers to Type A malicious servers. Its propagation component can somehow detect Type B malicious servers. NeighbourWatcher is designed to detect the spammer promoted servers, which could include all type of servers. In terms of the detection mode, SMASH and VisHunter need to monitor network traffic to passively detect malicious servers. However, PoisonAmplifier and NeighbourWatcher could be deployed as online services, and they can pro-actively find more compromised servers and malicious servers through the Internet. In terms of the detection method, PoisonAmplifier, VisHunter, and NeighbourWatcher are supervised systems and they require initial seeds to either train the system or to bootstrap the propagation. However, SMASH complements to these systems by using an unsupervised method. Therefore, it can discover possible zero-day attacks.

As we can see, although these systems have their advantages and limitations, they greatly complement each other by being united and provide a relatively complete and multi-perspective inferring framework for the malicious cyber infrastructure detection.

7.2 Lessons Learned

As we have highlighted earlier regarding the challenges in the malicious cyber infrastructure detection, different malicious servers often join forces to leverage their diverse functionalities makes the malicious cyber infrastructure more powerful and complicated. From the failure of the previous detection approaches (as discussed in Chapter 2) and the success of our designed systems in their desired principles, we have learned the following valuable lessons.

Detecting malicious cyber infrastructures could be an effective way to counteract malware infection. Malware can evolve quickly and malware authors have enough resources (e.g., time and state-of-the-art security tools) to test version after version of their malware. Therefore, it is challenging to detect malware directly on infected hosts. The evaluation of our systems shows that malicious cyber infrastructure detection can detect malicious servers that are missed by state-of-the-art anti-virus tools and find new infected clients. Therefore, detecting malicious cyber infrastructures could be an excellent complement to existing host-based detections.

Our proposed model of malicious cyber infrastructures is effective to characterize general malicious cyber infrastructures. As we noticed, malicious cyber infrastructures usually comprise a variety of servers with diversity functions (e.g., exploit server, C&C server, and drop-zone server). Characterizing these servers based on their functionalities could restrict the systems to some specific type of servers. However, our proposed model based on the access patterns could avoid this problem, and the evaluation results further demonstrate that the systems based on our model can detect a variety of malicious servers.

An effective malicious cyber infrastructure detection solution should capture some fundamental properties of the malicious cyber infrastructure and study malicious cyber infrastructures from multiple dimensions to get a complete view. Servers involved in malicious cyber infrastructure are complicated and flexible. Even our proposed model has already simplified the malicious cyber infrastructure, in fact, no *single* technique or principle can perfectly detect *all* malicious servers involved in it. Each of our systems can only focus on the part of malicious cyber infrastructures, and each of our principles is effective only within its defined detection scope. However, we can combine multiple techniques together to cover multiple perspectives to improve the coverage of servers in malicious cyber infrastructures, similar to the case in which we use four principles to characterize malicious cyber infrastructures from different perspectives. This dissertation provides our experience in detecting malicious cyber infrastructure from different perspectives. We showed that our inference systems have different focuses and are complementary.

7.3 Applicability to Malicious Infrastructures over Other Platforms

With the research present in this dissertation, researchers can now make use of the characteristics of malicious cyber infrastructures we proposed in this dissertation to study the malicious cyber infrastructure over other platforms, such as malicious social network infrastructures and malicious mobile app infrastructures. For the malicious cyber infrastructures in this dissertation, we design four systems to characterize it from its attacking pattern (PoisonAmplifier), inner-correlation pattern(SMASH), hiding pattern (VisHunter) and promoting pattern (NeighbourWatcher). Next, I will try to outline how to apply these patterns can help to understand and detect the malicious social network infrastructure and the malicious mobile app infrastructure.

7.3.1 Malicious Social Network Infrastructure

The social network has emerged in recent few years but is growing rapidly in popularity. This viral growth makes them a very lucrative attack target. Cybercriminals have utilized social networking platforms (e.g., Twitter and Facebook) to conduct their malicious behaviors including sending spam [50], spreading malware [48], hosting botnet command and control (C&C) channels [49], and launching other underground illicit activities.

A large number of malicious accounts are created, and lots of benign accounts got compromised by cybercriminials to facilitate crimes and gain illegal profits. Those malicious accounts and compromised accounts essentially form the malicious social network infrastructure. Then we present how each pattern of malicious network infrastructure can guide the study of the malicious social network infrastructure.

Regarding the attacking pattern, in the social network context, we can still observe that spammers tend to explore some popular terms and tags in their postings to trick benign users. Therefore, we can still use search engines on social network platforms to *pro-actively* find *more* malicious accounts that post those special terms. In PoisonAmplifier, after finding a server sharing attackers' promoted contents, we use Search Poisoning Attack as an oracle to evaluate whether it belongs to compromised server or not. However, in social network platforms, definitely we need features extracted from their domain (e.g., Twitter spam [108]) as an oracle to achieve this.

Regarding the inner-correlation pattern, there do exist correlation among those malicious accounts. For example, from the follower/following perspective (similar to client dimension in SMASH) in the social network, those malicious accounts usually share many followers/followings. From the server perspective, those malicious accounts may post the same malicious URL. Therefore, we can still correlate malicious accounts from different perspectives to infer a group of correlated malicious accounts.

Regarding the hidden pattern, the malicious URLs promoted by malicious accounts still rely on the existing malicious network infrastructures to launch further malicious activities, in other words, our visibility feature could be still effective to catch these malicious URLs.

Regarding the promoting pattern, we observe that spammers will keep using compromised accounts to promote their malicious content. Those accounts form the promoting platforms for spammers. Similar to our system NeighbourWatcher, if we find the same content are promoted on previous uncorrelated accounts, such content has a high possibility to be the spammer promoted content.

Therefore, if we apply the patterns of malicious network infrastructure present in this dissertation to the malicious social network infrastructure plus the features extracted from social network domains, I believe it will provide a better view of malicious social network infrastructure.

7.3.2 Malicious Mobile App Infrastructure

Different from spread desktop malware, smartphone malware authors can utilize app markets to spread their malicious apps more efficient. Therefore, those malicious apps and the market accounts used to publish those malicious apps form the malicious mobile app infrastructure.

Regarding the inner-correlation pattern, there do exist correlation among malicious apps. For example, two malicious apps published by the same/correlated accounts could be correlated. From the network perspective, two malicious apps connecting to the same remote servers could be correlated. Therefore, we can still correlate malicious apps from different perspectives (e.g., market accounts level and network level) with some domain specific features (e.g., app permissions) to infer a group of correlated malicious apps.

Regarding the hidden pattern, the malicious servers connected by malicious apps still rely on the existing malicious network infrastructures and are usually invisible to benign users. Therefore, our visibility feature may still somehow catch these malicious servers.

Regarding the promoting pattern, app infrastructure is different with social network infrastructure and network infrastructure. Attackers in app market promote their malicious apps through posting fake reviews. Such review contents are usually positive comments for the malicious apps. However, the reviewers still form the promoting platforms of attackers. And the reviewers *always* promote unrelated apps together may become more suspicious.

Although the malicious app infrastructure has much difference with the malicious social network and malicious network infrastructure because of its context, the cybercrimnals of the malicious app infrastructure still need to promote their malicious apps and those malicious apps could be still correlated due to the limited resources (e.g., malicious servers and market accounts) the cybercriminals have.

8. CONCLUSION AND FUTURE WORK

8.1 Conclusion

Obfuscation and encryption techniques make directly malware detection less effective. Fortunately, malicious cyber infrastructures build by cybercriminals for malware distribution, control, and monetization gives us a chance to counteract malware. Thus, we urgently need solutions to characterize and disrupt malicious cyber infrastructures.

In this dissertation, we model the malicious cyber infrastructures based on how clients access to them. As a result, we characterize all the servers in malicious cyber infrastructures into three categories: compromised servers, malicious servers accessed through redirections, and malicious servers accessed through directly connecting. Then we proposed an inference framework to infer servers involved in malicious cyber infrastructures. Our framework consists of four prototype detection systems (PoisonAmplifier,SMASH,VisHunter, and NeighbourWatcher) with different focus of malicious cyber infrastructures. We have discussed the techniques and principles we used and summarized the lessons we have learned for each system.

Our inference framework meets the three goals proposed in Chapter 1.

First, each system is guided by a sound principle that captures some fundamental invariants of malicious cyber infrastructures. PoisonAmplifier explores the fact that attackers need to inject special terms and links to promoted those compromised servers. Therefore, as long as attackers want to trick users, PoisonAmplifier can always be used to find more compromised servers no matter how those servers are compromised. SMASH explores the fact that cybercriminals usually utilize multiple servers together to launch malicious activities, which characterize the nature of cyber infrastructure. VisHunter leverages an insight that cybercriminals make efforts to conceal their core servers in order to continue their malevolent activities without being detected. Thus, if cybercriminals exposure their servers to be visible, they have high chances to be detected by existing security system [76, 78]. NeighbourWatcher leverages an insight that cybercriminals usually have limited number of promoting platforms (normal websites that allow users to leave messages such as forums, wikis, and guestbooks), and they want to make full utilization of these platforms. It is not easy to find those perfect promoting platforms and frequently changing them will incur significant costs for cybercrimnials. Therefore, we believe our four principles can also be applicable to detecting future malicious cyber infrastructures.

Second, our solution characterizes malicious cyber infrastructures from different perspectives and provides four complementary systems to detect malicious cyber infrastructures. These systems are complementary in terms of their detection targets, deployment method, and detection model (passive vs proactive). For example, PoisonAmplifier can only detect compromised servers while SMASH mainly focuses on malicious servers detection. For the deployment, SMASH and VisHunter need to be deployed at the edge of the network to monitor HTTP traffic passively while PoisonAmplifier and NeighbourWatcher can be implemented as online services to find more compromised and malicious servers pro-actively on the Internet. Therefore, each system has advantages and limitations, and works well in its desired detection scope. We combine these different systems with different detection focuses/perspectives to provide a comprehensive and complementary inference framework for malicious cyber infrastructure detection.

Third, our solution is general. In design, each system is not restricted to a specific type of malicious servers (e.g., exploit server or C&C server), but instead, it targets on certain categories of servers (e.g., compromised servers or malicious servers).

Finally, our systems are practical and capable to work in real world. Our systems are evaluated on real-world network traces and/or with online public dataset. Experimental results show that our systems can accurately detect real-world malicious servers with a very low false positive rate on real-world normal network traffic.

8.2 Future Work

In the future, we plan to study the following directions:

- Improvement on efficiency and coverage of our inference systems. We plan to study new principles to improve the efficiency of systems and increase the detection coverage of malicious servers in malicious cyber infrastructure. For example, the current implementation of SMASH only explores a few dimensions to characterize the relationship among malicious servers. However, they can be extended by considering time-based dimension [69].

- Cooperative detection in depth and breadth. My current studies show that modern attacks become much more complex than before due to multiple diverse entities involved in and multiple stages they have. Therefore, in the future, we plan to study malicious cyber infrastructure in depth and breadth. By in depth, we mean that we will study the connection between those multiple diverse entities involved in malicious activities from novel perspectives. By breadth, we mean that we will correlate security results from different layers (e.g., Host, DNS, Proxy) since different layers usually have different views of the threats. My final goal is to design a framework to apply big data platform to coordinate security controls from different layers and perspectives to gain a complete view of threats.

REFERENCES

[1] 50,000 websites infected with spam from 'wplinksforwork'. http://news.softpedia.com/news/50-000-Websites-Infected-with-Spam-From-Wplinksforwork-223004.shtml/.

[2] Alexa Internet. http://www.alexa.com/.

[3] The bagle botnet. http://securelist.com/analysis/36046/the-bagle-botnet/.

[4] Blacklist check. http://whatismyipaddress.com/blacklist-check.

[5] Bloom filter. http://en.wikipedia.org/wiki/Bloom_filter.

[6] Botnets and google dorks: A new recipe for hacking. http://www.darkreading.com/vulnerabilitymanagement/167901026/security/vulnerabilities/231500104/botnetsandgoogledorksanewrecipeforhacking.html.

[7] cfinder. http://www.cfinder.org/.

[8] CMU researcher finds web hackers profiting from illegal online pharmacies. http://www.darkreading.com/insider-threat/167801100/security/client-security/231400204/cmu-researcher-finds-web-hackers-profiting-from-illegal-online-pharmacies.html.

[9] DNS-BH-Malware Domain Blocklist. http://www.malwaredomains.com/.

[10] Easylist. https://easylist.adblockplus.org/en/.

[11] The (evil) genius of comment spammers. http://www.wired.com/wired/archive/12.03/google.html?pg=7.

[12] Globalspyware. http://globalspyware.com/.

[13] Google bombs. http://www.searchenginepeople.com/blog/incredible-

google-bombs.html.

[14] Google cache. http://www.googleguide.com/cached_pages.html.

[15] Google fights poisoned search results. http://www.securitynewsdaily.com/google-poisoned-search-results-0603/.

[16] Google pagerank api in php. http://www.fusionswift.com/2011/10/google-pagerank-api-in-php-october-2011/.

[17] Google rolls out content spam detection. http://www.nationalpositions.com/blog/seonewsgooglerollsoutcontentspamdetection/.

[18] Google safe browsing. http://code.google.com/apis/safebrowsing/.

[19] Google search and search engine spam. http://googleblog.blogspot.com/2011/01/google-search-and-search-engine-spam.html.

[20] Google trend. http://www.google.com/trends.

[21] Googledork. http://googledork.com/.

[22] Googlesuggest. http://code.google.com/p/google-refine/wiki/SuggestApi.

[23] Hamming distance. http://en.wikipedia.org/wiki/Hamming_distance.

[24] Httpclient. http://hc.apache.org/httpclient-3.x/.

[25] The keyword shop. http://www.blackhatworld.com/blackhat-seo/buy-sell-trade/.

[26] Keyword stuffing. http://www.seo.com/blog/keyword-stuffing/.

[27] Malware domain blocklist. http://www.malwaredomains.com/.

[28] Malware domain list. http://www.malwaredomainlist.com/.

[29] Malware domain list. http://www.malwaredomainlist.com/.

[30] Malware evolving faster than security software. http://www.stuff.tv/sg/news/malware-evolving-faster-security-software.

[31] Malware traffic analysis. http://malware-traffic-analysis.net/.

[32] N-gram algorithm. http://en.wikipedia.org/wiki/N-gram.

[33] Nearly 1 million new malware threats released every day. http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/.

[34] Notfollow. http://en.wikipedia.org/wiki/Nofollow.

[35] Page rank. http://en.wikipedia.org/wiki/PageRank.

[36] The pharmacy example. http://www.cmu.edu/news/stories/archives/2011/august/aug11_onlinepharmacyhackers.html.

[37] Phishtank. http://www.phishtank.com/.

[38] rel="nofollow". http://support.google.com/webmasters/bin/answer.py?hl=en&answer=96569.

[39] Royal wedding, obama birth certificate search poisoned with fake av links. http://www.eweek.com/c/a/Security/Royal-Wedding-Obama-Birth-Certificate-Search-Poisoned-with-Fake-AV-Links-489242/.

[40] Safe browsing-protecting web users for five years and counting. http://googlepublicpolicy.blogspot.com/2012/06/safe-browsingprotecting-web-users-for.html.

[41] Sality botnet. http://en.wikipedia.org/wiki/Sality.

[42] Spambust. http://spambusted.com/.

[43] Spyeye tracker. https://spyeyetracker.abuse.ch/.

[44] Stop forum spam. http://www.stopforumspam.com/.

[45] TeamViewer. http://www.teamviewer.com/.

[46] TLD List. https://wiki.mozilla.org/TLD_List.

[47] Trending topics. http://support.twitter.com/entries/101125-about
-trending-topics.

[48] Twitter accounts spreading malicious code. https://www.helpnetsecurity.com/2010/12/03/twitter-accounts-spreading-malicious-code/,.

[49] Twitter-based botnet command channel. https://www.arbornetworks.com/

blog/asert/twitter-based-botnet-command-channel/,.

[50] Twitter vulnerability allows cyber criminals to spread spam. https://www.one.com/en/web-hosting-news/website/twitter-vulnerability-allows-cyber-criminals-to-spread-spam-links$800076628.htm.

[51] VirusTotal. https://www.virustotal.com/\#url.

[52] Websense 2013 threat report. http://www.websense.com/assets/reports/websense-2013-threat-report.pdf.

[53] What does your google pagerank mean. http://www.redfusionmedia.com/google_pagerank.htm.

[54] What is onlinefwd.com. http://botcrawl.com/how-to-remove-onlinefwd-virus/.

[55] Word press. http://wordpress.com/.

[56] Wot (web of trust). http://www.mywot.com/.

[57] Zeus botnet. http://en.wikipedia.org/wiki/Zeus_(Trojan_horse).

[58] Zeus tracker. https://zeustracker.abuse.ch/.

[59] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: characterizing internet scam hosting infrastructure. In *USENIX Security Symposium'07*, 2007.

[60] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting malware domains at the upper DNS hierarchy. In *USENIX Security Symposium*, 2011.

[61] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: Detecting the rise of DGA-based malware. In *USENIX Security*, 2011.

[62] M. Antonakakis, Perdisci R, W. Lee, N. Vasiloglou, and D. Dagon. Detecting malware domains at the upper DNS hierarchy. In *USENIX Security Sympo-*

sium'11, 2011.

[63] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive DNS analysis. In *NDSS*, 2011.

[64] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. In *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[65] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. Delta: automatic identification of unknown web-based infection campaigns. In *CCS*, 2013.

[66] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the seventh international conference on World Wide Web*, 1998.

[67] Aydın Buluç and John R. Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC)*, 34(4):170 – 191, 2012.

[68] M. Cova, C. Kruegel, and G. Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. In *WWW'10*, 2010.

[69] H. Gao, C. Yegneswaran, Y. Chen, P. Porras, S. Ghosh, J. Jiang, and H. Duan. An empirical reexamination of global DNS behavior. In *sigcomm*, 2013.

[70] M. Graziano, D. Canali, L. Bilge, A. Lanzi, and D. Balzarotti. Needles in a haystack: mining information from public dynamic analysis sandboxes for malware intelligence. In *Proceedings of the 24th USENIX Conference on Security Symposium*, 2015.

[71] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *USENIX Security Symposium*, 2008.

[72] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotSniffer: De-

tecting botnet command and control channels in network traffic. In *NDSS*, 2008.

[73] Q. Gu and P. Liu. Denial of Service Attacks. In *Technical Report*.

[74] Z. Gyongyi and H. Garcia-Molina. Web Spam Taxonomy. In *Technical report, Stanford Digital Library Technologies Project, Mar*, 2004.

[75] C. Hsu, C. Huang, and K. Chen. Fast-flux bot detection in real time. In *RAID*, 2010.

[76] L. Invernizzi, S. Benvenuti, P.M. Comparetti, M. Cova, C. Kruegel, and G. Vigna. EVILSEED: A Guided Approach to Finding Malicious Web Pages. In *IEEE Symposium on Security and Privacy (Oakland'12)*, 2012.

[77] L. Invernizzi, P. Comparetti, Stefano Benvenuti, C. Kruegel, M. Cova, and G. Vigna. EVILSEED: A Guided Approach to Finding Malicious Web Pages. In *IEEE Symposium on Security and Privacy*, 2012.

[78] Z. Xu J. Zhang, C. Yang and G. Gu. PoisonAmplifier: A Guided Approach of Discovering Compromised Websites through Reversing Search Poisoning Attacks. In *Proceedings of the 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID12)*, 2012.

[79] J. John, F. Yu, Y. Xie, M. Abadi, and A. Krishnamurthy. deSEO: Combating search-result poisoning. In *Proceedings of the 20th USENIX Security*, 2011.

[80] P. Kolari, T. Finin, and A. Joshi. SVMs for the blogosphere: Blog identification and splog detection. In *Proceedings of AAAI Spring Symposium on Computational Approaches to Analysing Weblogs, March*, 2006.

[81] S. Lee and J. Kim. WarningBird: Detecting suspicious URLs in Twitter stream. In *NDSS'12*, 2012.

[82] N. Leontiadis, T. Moore, and N. Christin. Measuring and Analyzing Search-Redirection Attacks in the Illicit Online Prescription Drug Trade. In *USENIX*

Security Symposium'11, 2011.

[83] N. Leontiadis, T. Moore, and N. Christin. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. In *Proceedings of the 20th USENIX Security*, 2011.

[84] N. Leontiadis, T. Moore, and N. Christin. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. In *Proceedings of the 20th USENIX Security*, 2011.

[85] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang. Finding the Linchpins of the Dark Web: a Study on Topologically Dedicated Hosts on Malicious Web Infrastructures. In *IEEE Symposium on Security and Privacy (Oakland'13)*, 2013.

[86] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang. Knowing your enemy: understanding and detecting malicious web advertising. In *CCS'12*, 2012.

[87] Zhou Li, Sumayah Alrwais, XiaoFeng Wang, and Eihal Alowaisheq. Hunting the Red Fox Online: Understanding and Detection of Mass Redirect-Script Injections. In *IEEE Symposium on Security and Privacy*, 2014.

[88] L. Lu, R. Perdisci, and W. Lee. SURF: Detecting and Measuring Search Poisoning. In *Proceedings of ACM Conference on Computer and Communications Security*, 2011.

[89] G. Mishne, D. Carmel, and R. Lempel. Blocking Blog Spam with Language Model Disagreement. In *First International Workshop on Adversarial Information Retrieval on the Web, at 14th international conference on World Wide Web(WWW)*, 2005.

[90] T. Nelms, R. Perdisci, and M. Ahamad. Execscent: Mining for new C&C domains in live networks with adaptive control protocol templates. In *USENIX Security Symposium*, 2013.

[91] Y. Niu, Y.M. Wang, H. Chen, M. Ma, and F. Hsu. A quantitative study of forum spamming using context-based analysis. In *Proceedings of Network and Distributed System Security Symposium(NDSS), February*, 2007.

[92] R. Perdisci, I. Corona, and G. Giacinto. Early Detection of Malicious Flux Networks via Large-Scale Passive DNS Traffic Analysis. In *IEEE Transactions on Dependable and Secure Computing, 9(5), Sept.-Oct. 2012, pp. 714-726*, 2012.

[93] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *USENIX NSDI*, 2010.

[94] A. Ramachandran, A. Dasgupta, K. Weinberger, and N. Feamster. Spam or ham?: characterizing and detecting fraudulent not spam reports in web mail systems. In *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference(CEAS 11)*, 2011.

[95] C. Seifert, I. Welch, and P. Komisarczuk. HoneyC - The Low-Interaction Client Honeypot. In *NZCSRCS'07*, 2007.

[96] Y. Shin, M. Gupta, and S. Myers. Prevalence and mitigation of forum spamming. In *Proceedings of the 30th Annual IEEE Conference on Computer Communications (INFOCOM)*, 2011.

[97] Y. Shin, M. Gupta, and S. Myers. The Nuts and Bolts of a Forum Spam Automator. In *Proceedings of the Wkshp. on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[98] J. W. Stokes, R. Andersen, C. Seifert, and K. Chellapilla. WebCop: Locating Neighborhoods of Malware on the Web. In *USENIX LEET*, 2010.

[99] G. Stringhini, C. Kruegel, and G. Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *CCS*, 2013.

[100] E. Tan, L. Guo, S. Chen, X. Zhang, and Y. Zhao. Spam behavior analysis and

detection in user generated content on social network. In *Proceedings of 32nd International Conference on Distributed Computing Systems (ICDCS 2012), Macao, China, June 18-21,*, 2012.

[101] D. Wang, S. Savage, and G. Voelker. Cloak and Dagger: Dynamics of Web Search Cloaking. In *Proceedings of ACM Conference on Computer and Communications Security*, 2011.

[102] D. Y. Wang, S. Savage, and G. M. Voelker. Cloak and Dagger: Dynamics of Web Search Cloaking. In *CCS'11*, 2011.

[103] Y. Wang, M. Ma, Y. Niu, and H. Chen. Double-Funnel: Connecting Web Spammers with Advertisers. In *proceedings of the 16th international conference on World Wide Web, pages 291300*, 2007.

[104] B. Wu and B. Davison. Cloaking and redirection: A preliminary study. In *Adversarial Information Retrieval on the Web*, 2005.

[105] B. Wu and B. Davison. Identifying link farm spam pages. In *Special Interest Tracks and Posters of the International Conference on World Wide Web*, 2005.

[106] B. Wu and B. Davison. Detecting semantic cloaking on the Web. In *Proceedings of International Conference on World Wide Web*, 2006.

[107] B. Wu and B. D. Davison. Cloaking and redirection: A preliminary study. In *AIRWeb'05*, 2005.

[108] C. Yang, R. Harkreader, and G. Gu. Die free or live hard? empirical evaluation and new design for fighting evolving twitter spammers. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID'11)*, 2011.

[109] T. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *DIMVA*, 2008.

[110] J. Zhang and G. Gu. Neighborwatcher: A content-agnostic comment spam

inference system. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium(NDSS'13)*, 2013.

[111] J. Zhang, X. Hu, J. Jang, T. Wang, G. Gu, and M. Stoecklin. Hunting for invisibility: Characterizing and detecting malicious web infrastructures through server visibility analysis. In *Proceedings of IEEE International Conference on Computer Communications((INFOCOM'16)*, 2016.

[112] J. Zhang, S. Saha, G. Gu, S.J. Lee, and M. Mellia. Systematic mining of associated server herds for malware campaign discovery. In *Proceedings of the 35rd International Conference on Distributed Computing Systems (ICDCS'15)*, 2015.

[113] J. Zhang, C. Yang, Z. Xu, and G. Gu. Poisonamplifier: A guided approach of discovering compromised websites through reversing search poisoning attack. In *Proceedings of the 15st International Symposium On Research in Attacks, Intrusions and Defenses (RAID'12)*, 2012.