2016

Revealing Malicious Contents Hidden In The Internet

Muhammad Nazmus Sakib
University of South Carolina

REVEALING MALICIOUS CONTENTS HIDDEN IN THE INTERNET

by

Muhammad Nazmus Sakib

Bachelor of Science
Bangladesh University of Engineering and Technology, 2008

Submitted in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2016

Accepted by:

Chin-Tser Huang, Major Professor

Srihari Nelakuditi, Committee Member

Csilla Farkas, Committee Member

Wenyuan Xu, Committee Member

Ying-Dar Lin, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

DEDICATION

To my parents, Muhammad Abdur Razzaque and Nazma Begum, who made it possible for me to come this far.

ACKNOWLEDGEMENTS

I want to thank my advisor Dr. Chin-Tser Huang for being my mentor and for his invaluable guidance and constant support. I would also like to thank Dr. Srihari Nelakuditi, Dr. Csilla Farkas, Dr. Wenyuan Xu and Dr. Ying-Dar Lin for sparing their valuable time to serve on my dissertation committee and for extending their invaluable knowledge and advice to help me improve my dissertation. Finally, I want to thank my wife, Rahnuma Akhter, who was always there by my side for this journey.

.

ABSTRACT

In this age of ubiquitous communication in which we can stay constantly connected with the rest of the world, for most of the part, we have to be grateful for one particular invention - the Internet. But as the popularity of Internet connectivity grows, it has become a very dangerous place where objects of malicious content and intent can be hidden in plain sight. In this dissertation, we investigate different ways to detect and capture these malicious contents hidden in the Internet. First, we propose an automated system that mimics high-risk browsing activities such as clicking on suspicious online ads, and as a result collects malicious executable files for further analysis and diagnosis. Using our system we crawled over the Internet and collected a considerable amount of malicious executables with very limited resources. Malvertising has been one of the major recent threats against cyber security. Malvertisers apply a variety of evasion techniques to evade detection, whereas the ad networks apply inspection techniques to reveal the malicious ads. However, both the malvertiser and the ad network are under the constraints of resource and time. In the second part of this dissertation, we propose a game theoretic approach to formulate the problem of inspecting the malware inserted by the malvertisers into the Web-based

advertising system. During malware collection, we used the online multi-AV scanning service VirusTotal to scan and analyze the samples, which can only generate an aggregation of antivirus scan reports. We need a multi-scanner solution that can accurately determine the maliciousness of a given sample. In the third part of this dissertation, we introduce three theoretical models, which enable us to predict the accuracy levels of different combination of scanners and determine the optimum configuration of a multi-scanner detection system to achieve maximum accuracy. Malicious communication generated by malware also can reveal the presence of it. In the case of botnets, their command and control (C&C) communication is good candidate for it. Among the widely used C&C protocols, HTTP is becoming the most preferred one. However, detecting HTTP-based C&C packets that constitute a minuscule portion of everyday HTTP traffic is a formidable task. In the final part of this dissertation, we present an anomaly detection based approach to detect HTTP-based C&C traffic using statistical features based on client generated HTTP request packets and DNS server generated response packets.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Most cyber crimes can be attributed to hacking or cracking, and computer virus or worm. Hacking or cracking falls into the category of malicious activity, in which the cyber criminal is online to perform malicious actions. On the other hand, computer virus or worm can be categorized as malicious content, in which the cyber criminal first injects the malicious contents into the victim system, and lets the malicious contents perform the malicious actions. In practice, there are many forms of malicious contents. A majority of them is classified as malicious software, or in short, malware. Malware is the primary and in many cases the only weapon of attack used by the cyber criminals. They usually use it in an intelligent way so that the victim remains unaware of the attack until very late. This is possible for the autonomous and active nature of software objects. Other forms of malicious contents involve some kind of malicious activity or communication and are passive on their own. Examples of such malicious content include botnet C&C communication, network intrusion packets, spam

emails, etc. These passive malicious objects are created with stealthy characteristics as well. Therefore, detection of malicious contents is imperative for ensuring the security of most modern cyber systems.

1.2 Problem Overview

1.2.1 Collection and Inspection of Malware Hidden in Online Advertising

To develop effective detection and mitigation techniques against malware, the first step is to develop a repository of existing malware samples for analysis and testing. For this purpose, we need an effecting malware collection system, which can provide us with the latest versions of active malware executable binary and other related files from various Internet sources. In the recent past, the online advertising system has become one of major sources of Internet malware. Over the years, this system has evolved to become very effective in reaching and delivering content to targeted audiences consisting of all kinds of Internet users. Recently, cyber-criminals have started exploiting this system as an effective and risk-free channel to disseminate malware. Many popular websites became victims to such exploitation and have had malicious advertisements placed on their webpages or widgets unknowingly, including Horoscope.com, The New York Times [1], the London Stock Exchange, Spotify [2], and The Onion. The most recent addition to this list was earlier in 2015 when HuffingtonPost website

served malicious ads via AOL ad-network [3]. Since the cyber criminal are already delivering their goods via malvertising, this should be a good source to find and capture active malware samples. But we have not seen any prior work where malvertising was considered a source of malware for collection. We want to address this issue in this dissertation.

Most malvertisements operate with the help of a tool called exploit kit [4], which can probe the vulnerabilities on the victim machine's web browser or plug-in in order to exploit and install the malware. There is an expensive price tag attached to the acquisition of these exploit kits. Moreover, in order to protect their "investments" on malicious ads and malware from detection by the ad network, malvertisers often apply to their campaigns a variety of evasion techniques such as fingerprinting the execution environment, redirecting to compromised IP addresses, and malware polymorphism (introduced in more detail in the next section). These evasion techniques also incur considerable overhead cost on the malvertiser. On the other hand, in order to control and limit the huge reputation damage and financial losses caused by malvertising campaigns [5], the ad network also spends a lot of money and efforts to apply inspection techniques on submitted ads, including live monitoring and code analysis. These inspection efforts also incur substantial overhead coming from labor, infrastructure, intellectual property fee for licensing diagnosis, time

needed to conduct analysis, and cost for establishing partnership with other companies for sharing of expertise and data [6]. However, we note that both the malvertiser and the ad network are under the constraints of resource and time, which makes it impossible and impractical for the malvertiser to always submit malicious ads and for the ad network to inspect every submitted ad. Therefore, the ad network needs proper guidelines to effectively manage its resources for inspection to maximize its chance to thwart possible malvertising campaigns. We intend to address this problem in this dissertation.

1.2.2. Maximizing Accuracy in Multi-scanner Malware Detection System

Malicious software or malware is one of the major tools of cyber attack. Every cyber attack involves some kind of malware. Therefore, detection of malware is one of the cornerstones of modern cyber security. For a long time, we have been relying on the anti-malware or anti-virus scanners to detect malware and to protect ourselves from it. Variety of anti-malware scanners have been developed over the years with different levels of performances. In the early days, a single scanner could detect most of the malware out there. But over time, the malware writers and their repository of malware has evolved and proliferated so much that no single anti-malware engine can protect us from all of them. Moreover, researchers proved that combining the power of multiple anti-malware engines improve detection accuracy and performance significantly. This is why we now

have a lot of online multi-AV scanning services and tools (VirusTotal [8], Jotti, VirScan, etc.) at our disposal.

Although we have many multi-AV scanning services and tools available, most of them are used only for informational purposes or as a source of second opinion. None of them directly provide an exact decision of whether a particular sample is malicious or benign. Instead, they work as an information aggregator and only list the individual results returned from each anti-virus scanning engine. The responsibility of making a decision based on these individual scan results is up to the human user. This may be convenient for personal use where an end-user is looking for a second opinion for an unknown sample downloaded from the Internet. But if we want to use these multi-scanner detection systems effectively for a large scale detection and collection operation, we need the system to automatically come up with the best decision. Now, the question remains - how the system can do that? Obviously, it has to use the available information at hand. Let's look at the available information we can have for the unknown sample set. Firstly, we have the individual scan results from various scanners, which can be considered merely as their opinions. We are labeling them as "opinions" since we don't know for sure whether they are right or wrong. Secondly, we have the statistics for each scanner indicating their accuracy and performance. These statistics are accumulated from previous

scanning results which can be proven as right or wrong over the course of time. These statistical accuracy values can be used to measure how right or wrong these scanners can be. In other words, these are the ratings that indicate how good these scanners are. Now, the original problem becomes determining how to combine these detection accuracy ratings and the actual scan results for a given unknown sample to classify the sample as benign or malicious with the best possible accuracy. We further investigate this problem in this dissertation.

1.2.3 Detection of HTTP Botnet Command and Control Traffic

A botnet is a network of compromised computers, each of which harbors a piece of malicious software called bot. The bot software is remotely controlled by a botmaster, who exploits the botnet for malicious purposes like launching a distributed denial-of-service (DDoS) attack, spamming, performing click-fraud scams, stealing personal user information, etc. At the heart of any botnet is its communication architecture, i.e. how the botmaster communicates with hundreds and thousands of bot members. Since the size of a botnet is particularly crucial for its business, the botnet needs to be formed over common and popular network infrastructure, especially the Internet. Therefore, the botmaster chooses legitimate communication channels to interact with the bots. The server that the botmaster uses for its communication is called Command and Control (C&C) server. Internet Relay Chat (IRC) used to be the most prevalent communication

channel among the earliest botnets. Over time, it has been proved that the botnets formed over IRC network was not stealthy and the entire botnet could be shut down by simply taking down the IRC server. Moreover, network traffic monitoring on IRC based botnets was easier and effective in identifying C&C communication among botnet hosts. Consequently, botnets have evolved to adopt more common and generalized networking protocols and thus developed a stealth mechanism. Of the newer protocols used by botnets, peer-to-peer (P2P) protocols and hypertext transfer protocol (HTTP) are the most notable. The main advantage of using P2P networks is that it removes the centralized architecture from the botnet and makes it harder to shut down. However, P2P botnets suffer from higher latency in C&C communication and increased complexity in controlling the botnet as a whole. By contrast, HTTP, still being a centralized client-server protocol, provides the botmasters with desirable trade-off between stealth and performance. The protocol that runs the World Wide Web (WWW) is one of the most widely used network protocols, which helps the botmasters in bypassing most firewalls. In addition, HTTP allows using encryption to avoid detection based on deep packet inspection.

Security researchers have been working for many years on botnet detection and mitigation. Over the recent years, we have seen a significant number of proposals on how to detect different types of botnets. A

straightforward approach is to apply C&C traffic signatures which can be very effective for a specific botnet. The problem with this approach is that new botnets emerge very fast with newer communication patterns, which require new signatures to detect. To address this issue, most of the network traffic based methods apply some kind of machine learning algorithm to train and identify communication patterns and thus adapt to newer threats. However, these methods still focus on identifying botnet communication itself based on certain features, rather than isolating legitimate communication from the malicious ones. It is far easier for the botmasters to avoid certain patterns and come up with new techniques when they already know what patterns the defenders are looking for. Consequently, the detection methods begin to suffer from deteriorating performances against newer botnets. The detection of botnet C&C traffic becomes much more difficult when it comes to HTTP based C&C, since the percentage of C&C packets among the overall everyday Web traffic is in microscopic range. We investigate this problem further in this dissertation.

1.3 Overview of Dissertation

In this dissertation, we address four problems in the area of detection of malicious contents hidden in the internet. The organization of this dissertation is as follows.

In Chapter 2, we provide a background on online advertising and malicious display ads and propose automated simulation of the user clicks and automatic downloads to collect and analyze malicious executable files generated in the process. We implemented an automated system to mimic harmful and risky browsing activities such as clicking on suspicious online ads, and thereby to collect malicious executable files for further analysis and diagnosis. Using our system we crawled over the Internet for a period of 3 months to collect a significant amount of ad frame or placeholder URLs, which has been monitored for another period of 3 months to collect more than 13 thousand malicious executables. The experimental results showed that our system is quite effective in collecting online malware samples within a short period of time using very limited resources compared to other honeypot systems.

In Chapter 3, we provide a brief background on game theory and model the malvertising inspection problem as a game between an attacker (the malvertiser) and a defender (the ad network). We define the strategies and payoff functions of each player. We assume both players are aware of each other's strategies, cost and payoff functions, and the rate of malvertising detection by the ad network. We then calculate pure strategy and mixed strategy Nash equilibria for the game. Through the game model, we intend to better understand the relationship between the malvertiser and the ad network and

extract insights that can guide the ad network in its choice of inspection strategies.

In Chapter 4, we address the problem of finding an optimum configuration in multi-scanner malware detection systems by first deriving a mathematical model named Combined Probability Model (CPM) to capture the combined outcome of a specific combination of scanners, given their individual detection rates. The mathematical model consists of a set of formulas involving individual detection probabilities of the scanners. The model gives us a good approximation of the combined true and false detection probabilities of the combined system of scanners, which can be used to calculate the overall accuracy of the multi-scanner system for a specific configuration. Therefore, if we can calculate the accuracy of all configurations of the system, we can compare them to determine the optimum configuration that provides us with the maximum accuracy. We also present two other greedy heuristic based approximation models called Greedy Approximation Model (GAM) and Complementary Greedy Approximation Model (CGAM). These models apply greedy approximation over CPM formulas to improve runtime and at the same time try to maintain the accuracy as much as possible. In addition to the original problem, we also try to answer the following two questions - (1) Is it always beneficial to increase the number of scanners in a multi-scanner detection system? (2) How

can we select a subset from all available scanners, which will provide us with a maximum accuracy for a size of the given subset? To address the second question, we come up with a ranking system for the scanners which allows us select a best subset from the full set of scanners. To verify the accuracy of our models and to answer these additional questions, we first numerically simulate our models over randomly generated hypothetical datasets and test case scenarios. From the simulation results, we found that if the average false positive rate of the scanners is high enough, the accuracy value of multi-scanner system can decrease at some point with the increase in the number of scanners. At the end, we provide experimental evaluation based on real-world malware and goodware datasets and corresponding anti-virus scanning results using a popular online multi-AV scanning service, VirusTotal. From the evaluations, we can verify the accuracy of our simulation results and establish that our models along with the ranking system perform reasonably well in predicting the optimum configuration to achieve maximum accuracy based on available information.

In Chapter 5, we introduce an anomaly detection based approach to detect HTTP-based botnet C&C communication which focuses on how to prevent the botnet from upgrading itself to avoid detection. That means, we want to make it very hard for the botmaster to mimic the legitimate HTTP communication and

hide C&C activities. Our approach is based on identifying anomaly in client generated HTTP request packets as well as DNS server generated response packets for the same HTTP communication. Based on some initial analysis of both legitimate and botnet C&C HTTP traffic, we have selected some statistical features that are suitable for detecting anomaly in a large set of captured HTTP traffic. These features are based on patterns emerging from HTTP request packets, more specifically, the URL string that is used to fetch data from an HTTP server. Using these features we primarily run an unsupervised anomaly detection algorithm to distinguish between HTTP request packets generated by human actions and HTTP request packets generated by a software bot, both legitimate and malicious. Then, to further narrow down the isolated packets, we extract the primary domain names involved in those packets and run a semi-supervised anomaly detection algorithm using a selected set of features based on the DNS server response packets that particularly contain resolved IP address list (A or AAAA record). Eventually, we are left with a list of domain names that are highly probable to be involved in malicious C&C communication. Results indicate that our method can achieve more than 90% detection rate while maintaining a reasonably low false positive rate.

Finally, we conclude the dissertation with a brief summary of the research and directions for future work in Chapter 6.

CHAPTER 2

COLLECTION OF MALWARE DISSEMINATED VIA MALVERTISING

2.1 Background

2.1.1 Online Advertising

Online advertising is a form of advertising that uses the Internet as the delivery channel for promotional marketing messages to consumers. It includes all sorts of online marketing such as email marketing, search engine marketing (SEM) [7], social media marketing, display advertising, mobile advertising, etc. In this chapter, our focus is only on display advertising, the type of advertising that is located on websites in a wide range of different formats and contains items such as texts, images, flash, video, and audio. Besides the consumer, there are three major participants in online advertising described as follows. The Ad Publishers are the owners of the websites or online contents, who integrate or place advertisements into their contents. The Ad Networks are the companies that work as the middlemen who connect the Advertisers to interested Ad Publishers that want to host advertisements. Online Ad Networks usually maintain a

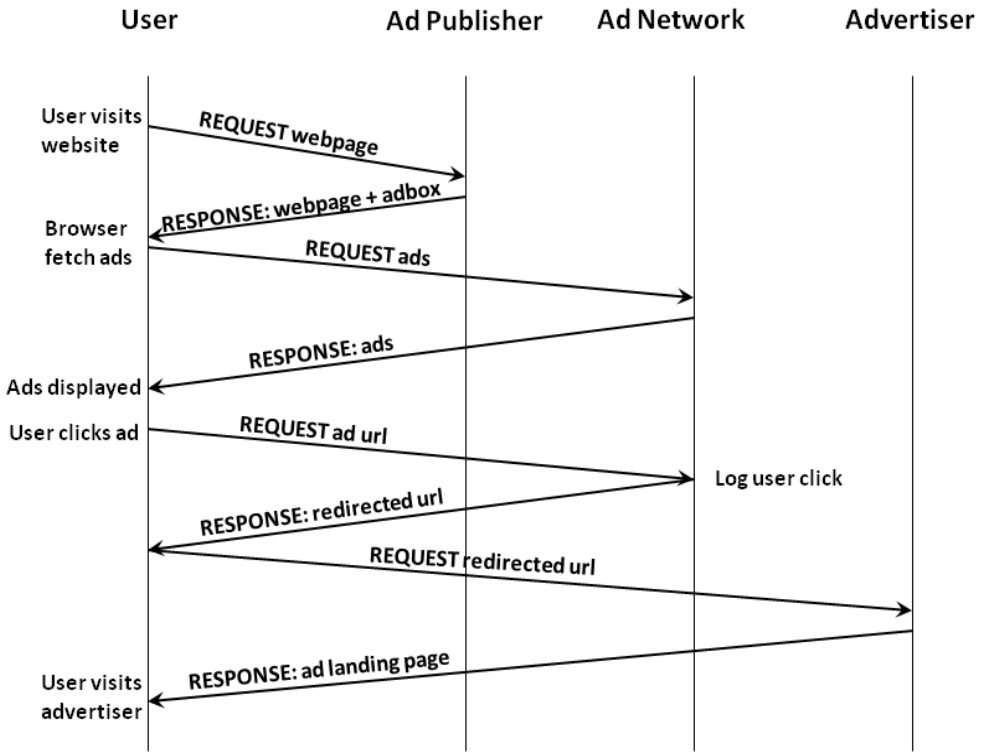


Figure 2.1 Advertisement delivery system.

central ad server which delivers advertisements to consumers, and also facilitates ad related activities such as targeting, tracking, reporting and billing. Lastly, the Advertisers are the business entities or individuals who are interested in promoting their products through online advertising. Figure 2.1 illustrates a typical scenario of how the ad delivery system works. From Figure 2.1 we can see four request-response style communications. The first such interaction is when the user opens a webpage hosted by the ad publisher who displays the ad frame or placeholder (referred to as adbox in the figure). This action triggers a background interaction of the browser with the ad network to fetch the actual

ads. This is shown as the second pair of request-response communication. The third interaction happens when the user actually clicks on the ad. The ad network sends the redirected URL as response which triggers the browser to request it to generate the final request-response communication. As a result, the browser gets the ad landing page.

2.1.2 Infection Process of Malicious Ads

The infection process of malicious ads can be largely divided into two categories: silent infection and user triggered infection. Silent infection can occur when an Internet user only visits a legitimate website that contains malicious ads. In this case, the malicious ad itself contains malicious code (written in JavaScript, Action Script, etc.) which can find Web browser vulnerabilities and exploit them to infect the user system. This is the most dangerous form of infection, since it does not require any interaction or trigger from the user. The mere action of visiting a legitimate and otherwise safe website triggers the infection. On the other hand, user triggered infections require some form of user interaction such as click or key press events. By refraining from risky interactions, the user can prevent infection in most cases. After the first interaction with the malicious ad, the user usually ends up visiting a malicious ad landing page hosted by the malicious advertiser. From this page, the user may also be infected in two ways: either automatically or based on further user interaction. The ultimate outcome can be

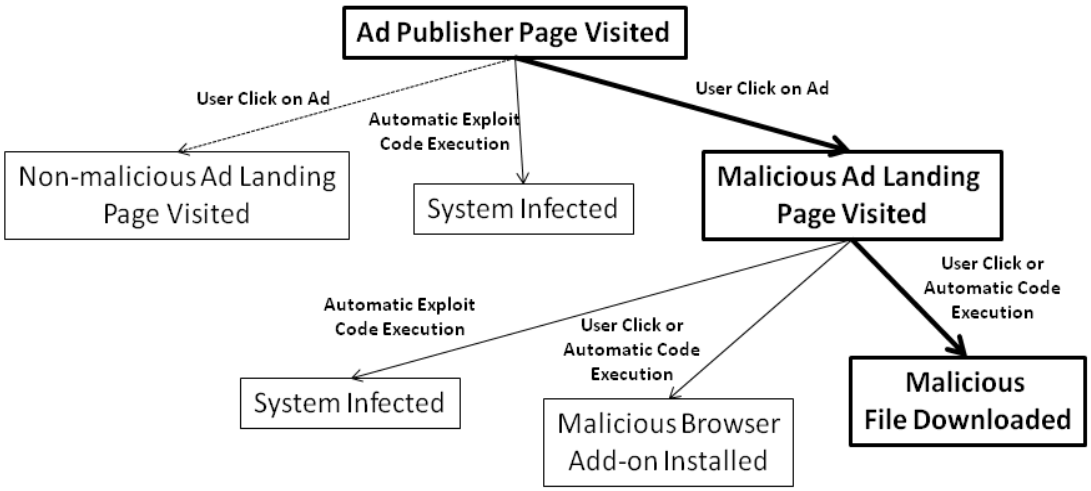


Figure 2.2 Infection paths of malicious advertisements.

one the following three results: user system being infected, a malicious browser add-on being installed, or a malicious executable file being downloaded. Figure 2.2 depicts various paths and outcomes generated when an ad publisher webpage is visited. The bold-faced sections of the figure highlight the path we focus on in this chapter.

Figure 2.3(a) shows a sample malicious ad frame. This is a typical malicious ad falsely claiming that the user needs to update his or her media player. If the user clicks on anywhere inside the ad frame (not just the buttons), it will open a new page where the user will be prompted to download a malicious binary executable file with names like "mediaplayer.exe" or "mediaplayerupdate.exe". Figure 2.3(b) shows the underlying JavaScript code for the same ad frame. We can clearly see here that the target ad landing page URL

(a)

```
<script type="text/javascript">
//<![CDATA[
try{if (!window.CloudFlare) {var
CloudFlare=
[{verbose:0,p:0,byc:0,owlid:"cf",bag2:1,mira
ge2:0,oracle:0,paths:{cloudflare:"/cdn-
cgi/nexp/dok3v=1613a3a185/"},atok:"4c750bef3
6588ce4bee6042874cc1ca5",petok:"0ed33e7fea8d
7142916672b16cf977bbf6748137-1430250590-
1800",zone:"watchcric.net",rocket:"0",apps:
{}}];CloudFlare.push({"apps":
{"ape":"e6cfda6f818b03dfc144e9707702a5f0"}})
;!function(a,b){a=document.createElement
("script"),b=document.getElementsByTagName
("script")[0],a.async=!
0,a.src="//ajax.cloudflare.com/cdn-
cgi/nexp/dok3v=7e13c32551/cloudflare.min.js"
,b.parentNode.insertBefore(a,b)}()}catch
(e){};
//]]>
</script>
```

(b)

Figure 2.3 (a) Sample malicious ad frame, (b) JavaScript code for the same malicious ad frame.

cannot be identified straightforwardly. Only after this JavaScript code is executed in the Web browser, we can see the target URL. This is the primary reason why we need to simulate user clicks on the ads to find the target URL.

2.2 System Design

Our system can be divided into four major components, including (i) Crawler, (ii) Detector, (iii) Extractor, and (iv) Verifier, as depicted in Figure 2.4. It also shows the input and output of each component. Each component implements a major stage in the overall process of collection and analysis of malicious executable files. Description of each of the components is given in the following subsections.

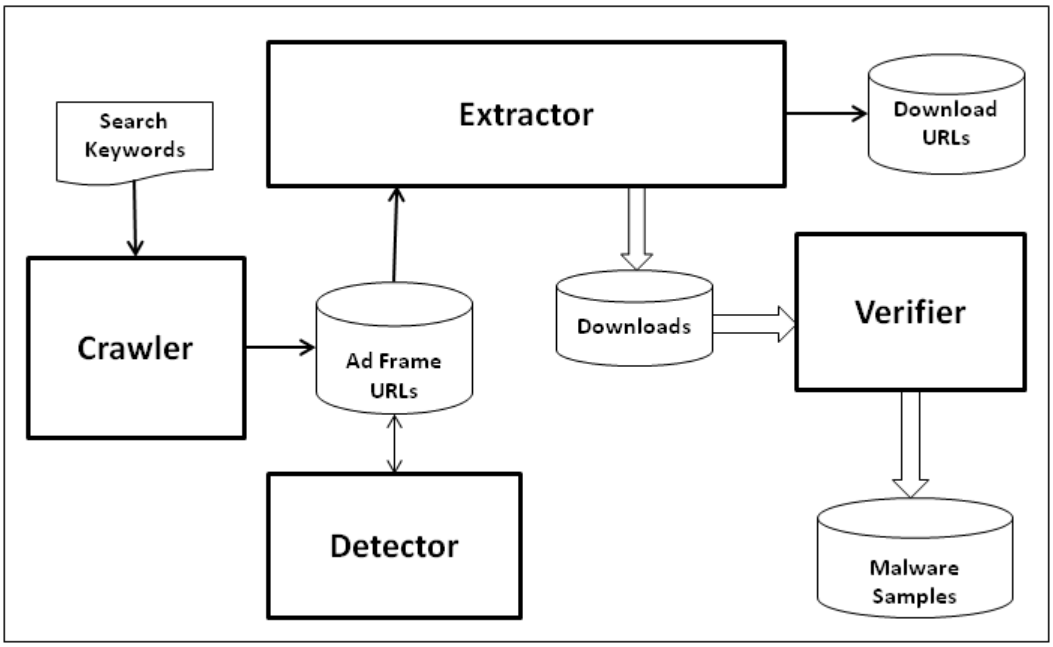


Figure 2.4 System architecture.

2.2.1 Crawler

The main task of this component is to crawl over the Internet and find ad frames or placeholders in various websites. To increase the effectiveness in finding websites with more ads and potential malicious ads, it makes use of the popular Internet search engines like Google, Bing, Yahoo, etc. A pre-defined list of search keywords is used to search websites via different search engines and extract a list of URLs. Then, the web pages of these URLs are fetched and parsed to detect ad frames or placeholders that display textual or graphical ads. If detected, the ad frame or placeholder URL is recorded into a list of ad frame URLs for further processing. Figure 2.5 shows a flowchart of the overall process of the Crawler.

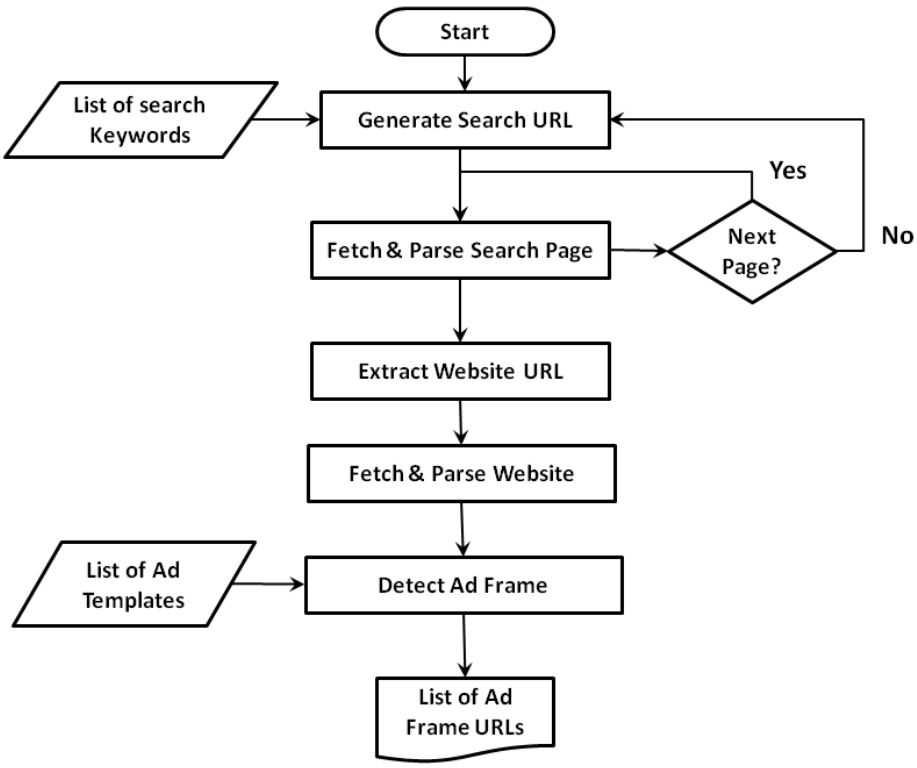


Figure 2.5 Flowchart of Crawler.

2.2.2 Detector

This component uses the list of ad frame URLs generated by the Crawler and detects whether the ad eventually results in a malicious download or not. To achieve this, we need to know what the target URLs are for the ads and test whether any one of them lead to an executable file download event. A simple HTML ad will contain the target URL as part of a plain HTML element. However, with the widespread use of Web 2.0 technologies, most of the ads now contain complex JavaScript or Action Script code (as shown in Figure 2.3), where it is very hard to find or generate the target URL. Therefore, we intend to

simulate user interaction such as mouse click on the ad itself which will trigger the target ad landing page URL for us. Once we can fetch the ad landing page, we can further parse and inspect to determine whether it contains any download URL which lead to a download of an executable binary file. If the Detector detects at least one such download URL, it records the download URL along with the ad frame URL. Figure 2.6 shows a flowchart of the overall process of the Detector.

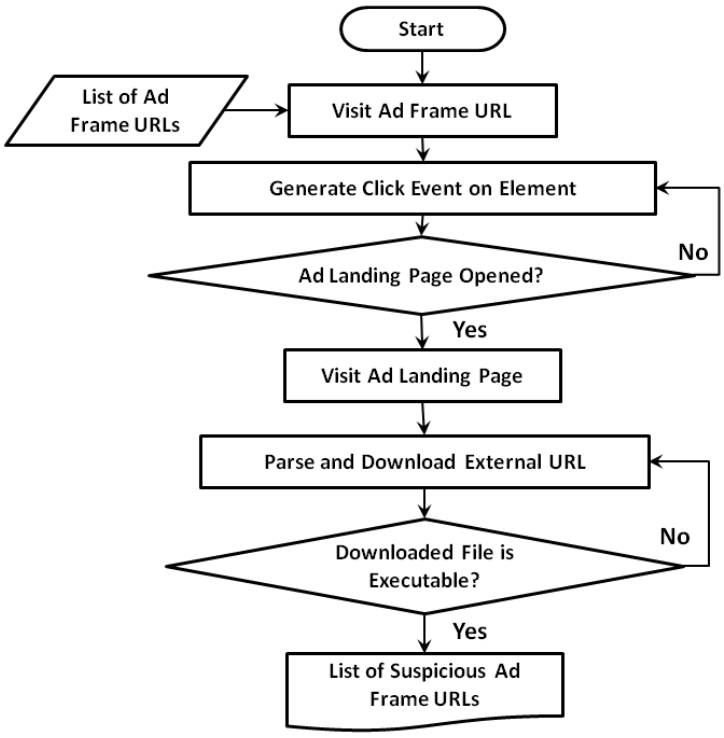


Figure 2.6 Flowchart of Detector.

2.2.3 Extractor

We make use of the ad frame URLs that is generated by the Detector to download more executable files. The design of the Extractor is similar to the Detector except that it is used to periodically monitor (and download from) only the already detected ad frame URLs. Using this approach we can maximize the number of downloaded files from a minimum number of ad frame URLs, since the contents of the ad placeholder changes dynamically over time with a probability of generating a new ad every time. Thus, by monitoring a single malicious ad placeholder we can extract many different malicious files. In addition, we can investigate the behavior of a single malicious ad frame URL and find out answers to questions like how frequently ads change, how many distinct download URLs are generated from the same ad frame, etc. Examples of such analysis results are further discussed in section 2.4.

2.2.4 Verifier

This is the part where we automatically submit the collected executable files to anti-malware scanning engines to verify the maliciousness of them. Instead of using a single scanner, we used the online service provided by VirusTotal [8] where at most 56 anti-virus scanning engines are used to generate the scanning report.

2.3 System Implementation

We have implemented our system using Python 3.4. Details of the implementation for each component are given in the following sub-sections.

2.3.1 Crawler

We used a list of search keywords generated from Table 2.1. Each keyword is generated by combining one or more qualifier keywords and one content keyword. To create this list of potential keywords, we tried to answer the following question: what types of websites are most likely to host high number of advertisements? To find the answer, we manually analyzed 20 ad-filled websites collected from various Internet discussion forums. We found that most of the websites offer free services or contain free contents, for which they try to compensate by placing as many ads as possible.

Table 2.1 Search Keywords

Qualifier Keywords	*Content Keywords*	
Free	Movies	News
Download	Ebook	Sports
Watch	Pdf	Highlights
Stream	Music	Software
Hack	Mp3	Freeware
	Wallpaper	Cracks
	Fonts	Password

Therefore, a good qualifier keyword is "free". Similarly, we found that one of the most desired services is downloading some content or data from the Internet. Hence, "download" should be a good qualifier keyword.

The fetching and parsing of websites has been done using Python libraries "Requests" [9] and "lxml" [10] respectively. The detection of ad frames or placeholders has been done using the Python library "adblockparser" [11] and a list of filters from EasyList [12], an ad filter provider service designed for the most popular ad blocking Web browser extension Adblock Plus [13].

2.3.2 Detector

To implement the Detector, we needed some way to simulate the user click events on the ads. Selenium WebDriver API [14] provided us with such features. This Python API can be easily used to simulate the behaviors exactly like what a normal human Web user will do, such as opening a URL in the browser, clicking on an ad, switching to new pages as a result of the clicking, responding to any JavaScript alert generated in the process, etc. A difficult task was to determine where to click, since ads are dynamically generated with varying sizes. Fortunately, Selenium provides a way to click on a specific HTML element. Therefore, we iterated over all the HTML elements of the ad and generated click event for them. The assumption we make is that at least one of the elements should be clickable and should produce our desired ad landing page as a result

of the click event. This is a valid assumption since usually the ads are generated such that the entire ad frame is clickable and the user can click anywhere inside the frame to produce the ad landing page.

After the ad landing page is opened, we parse the HTML source and find all target URLs leading to external resources. Here, the Selenium WebDriver executes most of the internal JavaScript code and we can use the innerHTML property (the property that sets or returns the HTML content of an element) for each element to get the generated HTML code from JavaScript code. In this way, we can make sure that we don't miss any target URL generated by internal JavaScript codes.

2.3.3 Extractor

The implementation of the Extractor is similar to Detector. The only difference lies in the input and output. The input for the Extractor only contains those ad frame URLs that have been detected already, and the output contains downloaded executable files with corresponding download URLs for a specific ad frame URL. We map the downloaded files and URLs to a specific ad frame URL for further analysis later.

2.3.4 Verifier

The VirusTotal API [15] provided by VirusTotal is used to implement the Verifier. For each of the downloaded files, we generated a scanning report from VirusTotal which contains how many anti-virus scanners have detected the file as malicious and what classes of malware the file belongs to.

2.4 Results and Analysis

We tested our system for a total period of 6 months and divided it into two stages of 3 months each. In the first stage, we deployed the Crawler and the Detector for 3 months. The Crawler used the search keywords generated from Table 2.1 and crawled 51,467 websites, where 10,950 of them contained at least one advertisement. The number of detected ad frame URLs were 73,240, which were passed to the Detector. We detected ad frames containing at least one target URL which lead to the download of executable binary files. In total, we found 895 such ad frame URLs. This is our input to the second stage of experiment.

In the second stage, we ran the Extractor for 3 months to monitor and extract downloads from 895 suspicious ad frame URLs. It ran a single iteration over all 895 of them 3 times a day. We recorded the download URLs along with the downloaded files for each individual ad frame URL. In total, we found 13,648 distinct executable binary files downloaded in the process. These files were fed to

the Verifier to identify false positives checking against 56 anti-virus scanners provided by VirusTotal. Only 115 files out of 13,648 pass all the anti-virus scanners as benign and 13,353 files were identified by at least one scanner as malicious. This means that 99% of the files collected by our system were identified as malicious by VirusTotal. Table 2.2 lists the number of different types of malware detected. A single malware sample can belong to two or more different categories, since modern malware is packaged with multiple features and functionalities. Here, we have considered all the labels for a single sample labeled by different scanners. From the VirusTotal reports, we found that on an average each sample was detected as malicious by at least 9 out of 56 scanners.

Table 2.2 Types of Malware Detected

Malware Type	*Total Number*	*Percentage*
Adware	12,952	97%
Trojan	10,816	81%
Virus	4,406	33%
Backdoors	3,872	29%
Potentially Unwanted Program	12,151	91%

Some interesting results were observed when we grouped the malware samples and their download URLs by corresponding ad frame URL. We observed that every time we extracted the download target, a new distinct URL can be found. Even though the downloaded binary files looked exactly same with respect to name and size, the files were found to be different when MD5 hash was calculated. We found that during the 3 month period, we could extract

approximately 49 malware samples on an average from a single ad frame URL, with a maximum of 255 samples. After the 3 month period ended, we replayed all download URLs to check their validity. 21% of them were still working while the rest of them were redirecting to a different webpage. In addition, malware samples collected from a single ad source usually fell into the same malware family or class. From these observations, we conclude that (1) a single malicious ad frame URL can be monitored for a long period of time to consistently collect malware samples, (2) URL that hosts the malware is changed frequently to provide a constant availability of malware as well as to thwart takedown efforts, (3) even though the malware samples disseminated by a single ad source belong to the same class or family, they could be distinct in binary content, which means every now and then a new malware payload is generated with a relatively short lifetime.

Additionally, we tested the captured download URLs via VirusTotal URL scanner service. We found that only 34% of the URLs were flagged as malicious by at least one URL scanner. Therefore, the list of malicious download URLs generated from our system can be a good addition to online URL blacklist services.

2.5 Related Work

In the scientific literature, malicious online advertising is better known as "malvertising" by taking the portmanteau of the words "malicious" and "advertising". Although numerous news articles have been published on malvertising, not many research articles can be found on this topic.

Sood et al. [16] provided one of the earliest accounts of how malvertising works. They explained several malvertising modes and offered a few guidelines to prevent them. S. Manfield-Devine presented the recent state of malvertising in [17], describing the use of Flash and mobile websites. Zhang et al. [18] proposed a detection scheme to detect malvertising cases using depth of the URL strategies. In addition, Google has opened a website [19] dedicated to prevent malvertising compromises in all of Google's and partners' ad properties in an effort to build community awareness against it.

There has been a considerable amount of research done regarding Web-based malware collection. In the year 2006, researchers from Microsoft [20] came up with an automated Web patrolling system to automatically identify and monitor malicious websites that install malware programs by exploiting browser vulnerabilities. Since then, we have seen many other research efforts to automate malware collection from the Web. Worth mentioning among these are HoneyBow [21], PhoneyC [22], Rozzle [23], WebPatrol [24], HoneyInspector [25],

and PMCCS [26]. HoneyBow toolkit is an automated malware collection system based on high-interaction honeypots, which are able to collect autonomous spreading malware in an automated manner. PhoneyC is a virtual honeyclient that mimics the behavior of the user-driven network client applications such as Web browsers and is exploited by an attacker's content to reveal the attack in the process. Rozzle is a JavaScript multi-execution environment that can reveal environment specific Internet malware. WebPatrol automatically collects Web-based malware scenarios including complete Web infection trails to enable further detailed analysis. HoneyInspector is another active honeypot system that collects malware from malicious websites as well as from shared P2P files. Proactive Malware Collection and Classification System (PMCCS) uses P2P software to actively search suspicious malware samples such as software crack tools. Although each of these research works presents a way to collect Web-based malware samples, none of these have explored malvertising and considered it as a source of malware collection and analysis.

2.6 Summary

Our main contribution in this work is, we have designed and implemented an automated system to collect malware samples from online advertising sources. To the best of our knowledge, this is one of the first efforts to automate

information collection for malvertising research, which can reveal many new paths of investigation and analysis in this area. Moreover, the collected samples are instances of live and active malware that are infecting Internet users at this very moment, which are extremely useful for research purposes.

As of now our system can only collect information related to downloadable executable binary files via malvertising sources. If we refer back to Figure 2.2, we can see that we have only implemented one path in the malicious ad infection process. There are still two more paths yet to be explored. One is where the system is infected in the background, that is, a malicious code is executed in the browser through browser vulnerabilities and plug-in exploits. The other one is where a malicious add-on is installed into the browser. We can further extend our work to incorporate both of these infection paths.

Along with the malicious executable files, we can collect the HTML, JavaScript and Action Script sources of the malicious ads and further investigate to find patterns so that they can be used in the future to detect malicious ads before they are executed. Moreover, the defenders can use the information about these patterns to develop mitigation strategies. This can be a very important future extension of our work.

CHAPTER 3

A GAME THEORETIC MODEL OF MALVERTISING

3.1 Background

3.1.1 Overview of Game Theory

Game theory identifies multi-person decision scenarios as games where each player selects actions which result in the best possible self rewards, while anticipating and considering the rational actions from other players. A player is the basic entity of a game who makes choices of what actions to perform. A game is a formal description of the strategic interaction that includes the constraints of, and payoffs for, a set of actions that the players can choose from, without specifying what actions they actually take. A solution concept is a formal description of how the game will be played by applying the best possible strategies and what the results might be. A strategy for a player is a complete set of actions in all possible scenarios throughout the game. If the strategy specifies to take a unique action in a scenario then it is called a pure strategy. If the strategy specifies a probability distribution for all possible actions in a scenario then the strategy is referred to as a mixed strategy.

Nash equilibrium is a solution concept that describes an equilibrium state of the game where no player would prefer to change his strategy as that would lower his payoffs given that all the other players are adhering to their respective strategies. This solution concept only specifies the equilibrium state but does not specify how that state is reached in the game. The Nash equilibrium is the most famous equilibrium and one of the most used solution concepts in game theory.

3.1.2 Game Theory Definitions

Game

A game is a formal description of the strategic interaction between opposing or co-operating entities where constraints and payoff for actions are taken into consideration.

Player

A player is a basic entity in a game that is required to make choices for actions.

Action

An action is a player's move in the given game.

Payoff

The payoff is the positive or negative reward to a player associated with a given action.

Strategy

A strategy is a set of actions that a given player can choose during game play.

3.1.3 The Malvertising Game

The major motivation behind malvertising is the potential lucrative profit. Many malvertising campaigns install on vulnerable machines a variety of ransomware, which encrypts user data and files and forces users to pay a ransom of several hundred dollars to obtain the decryption key. According to the 2016 Annual Security Report published by Cisco [27], the estimated yearly income from ransomware per successful malvertising campaign could reach as high as $34M. However, this potentially huge profit does not come for free; there is a cost associated with launching a campaign. Most malvertisements operate with the help of a tool called exploit kit [4], which can probe the vulnerabilities on the victim machine's web browser or plug-in in order to exploit and install the malware. Malvertisers need to either develop the exploit kit from scratch (need a lot of investment), hire someone to do it (there is a list of task prices in the Deep Web black market [28]), purchase it (about $20-30K [28]), or rent it (about $500/month [28]). There is an expensive price tag attached to any option. Moreover, in order to protect their "investments" on malicious ads and malware from detection by the ad network, malvertisers often apply to their campaigns a variety of evasion techniques such as fingerprinting the execution environment,

redirecting to compromised IP addresses, and malware polymorphism (introduced in more detail in the next section). These evasion techniques also incur considerable overhead cost on the malvertiser.

On the other hand, in order to control and limit the huge reputation damage and financial losses caused by malvertising campaigns [5], the ad network also spends a lot of money and efforts to apply inspection techniques on submitted ads, including live monitoring and code analysis. Similar to the case of launching malvertising campaigns, these inspection efforts also incur substantial overhead coming from labor, infrastructure, intellectual property fee for licensing diagnosis tools (sometimes including purchasing exploit kits for analysis purpose), time needed to conduct analysis (ranging from a few minutes to tens of hours, on average around 10 hours per case), and cost for establishing partnership with other companies for sharing of expertise and data [6].

However, we note that both the malvertiser and the ad network are under the constraints of resource and time, which makes it impossible and impractical for the malvertiser to always submit malicious ads and for the ad network to inspect every submitted ad. Therefore, the malvertising inspection problem can be modeled as a game between an attacker (the malvertiser) and a defender (the ad network).

3.1.4 Attacker and Defender Strategies in Malvertising Game

Malicious advertisers employ many strategies to evade detection including fingerprinting, redirection, just-in-time assembling and compilation, obfuscation, timing based evasion, etc. Researchers at Malwarebytes and GeoEdge [29] investigated malvertising campaigns for several months and found out about an effective evasion technique used by the threat actors called fingerprinting. This technique is actually not new, rather has been used by the exploit kits for some times now. Now it is being used earlier rather than late in the malvertising chain, helping the malicious advertisers to decide whether to display a malicious ad or a benign ad. Basically, the fingerprinting technique employs sophisticated obfuscated code inside the ad to detect indications that can identify a machine belonging to a security researcher or a honeypot. Researchers at Invincia [30] identified a new technique called "just-in-time" (JIT) or on-host assembly of malware. This novel approach can evade detection from network sandbox and traditional endpoint security solutions while compromising vulnerable systems. JIT malware uses late-binding techniques to assemble a malware executable on the target endpoint itself in order to evade network sandbox analysis. In addition, native Windows components from the target machine are used to assemble the payload. This helps in evading endpoint white-listing approaches that allow only approved programs to run. The most recently discovered

AdGholas [31] malvertising campaign have been found to have used steganography and file whitelisting approach to evade detection.

Most of the malvertising campaigns involve an exploit kit to carry out the infection or delivery of malicious payload. Prominent examples of exploit kits [4] include SweetOrange, Angler, Magnitude, Rig, Nuclear, etc. Exploit kits are also equipped with evasion techniques [4] such as fingerprinting, obfuscation, etc. Researchers have found that through a vulnerability in Internet Explorer, an attacker can check the presence of files or folders in an affected system, thereby detecting whether the system is a virtual machine or has an antivirus software installed. For obfuscation purposes, the use of Pack200 archive format has been seen in use by Angler exploit kit. Other evasion techniques include encrypted payload, IP and domain fluxing, domain shadowing, and file-less infections [32].

There has been some work done by both industry and academic researchers on the strategies that can be employed by the defender, i.e. the ad network or the ad publisher. GeoEdge [6] is a commercial provider for ad verification and protection services. Their services include automated ad verification solution that monitors live advertisements using a globally distributed network of monitors. Similar techniques involving crawling and monitoring have been found in some prior academic research works as well [33,

34, 35, 36]. Another focus of research was to detect malicious exploit kits. Taylor et al. [37] proposed a network-centric technique to detect malicious exploit kits by capturing tree-like web request structures and finding similarities among them. Their approach is based on the insight that to infect a client browser, a web-based exploit kit must guide the client browser to visit its landing page through multiple redirections generating a pattern of multiple web requests. This pattern can be identified as a tree-like structure and used for the purpose of detection of malicious exploit kits. Stock et al. [38] presented Kizzle, an antivirus signature generator for detecting exploit kits. Wang et al. [39] presented an approach for identifying new undetected landing pages that lead to drive-by downloads by using malicious content patterns identified in previously known collection of Malware Distribution Networks. Malicious obfuscated JavaScript code has been an integrated part of malvertising campaigns. Lu and Dubray [40] presented an approach for automatic de-obfuscation of JavaScript code using dynamic analysis and slicing that preserves code semantics. The resulting code becomes observationally equivalent to the original program with obfuscation removed which exposes the core logic of the computation it performs. Xu et al. [41] presented JStill, a mostly static approach to malicious obfuscated JavaScript detection that uses static analysis of function invocation and lightweight runtime inspections. Dong et al. [42] proposed AdSentry, a sandbox for JavaScript-based

advertisements that enables flexible controlling on ad script behaviors by completely mediating its access to the web page (including its DOM) without restricting the JavaScript functionality exposed to the ads. Dewald et al. [43] presented ADSandbox, an analytical sandbox system for malicious websites that executes any embedded JavaScript within an isolated environment and log every critical action. Analyzing these logs using heuristic rules, ADSandbox can decide whether the site is malicious or not. Another useful evasion technique employed by the attackers is URL redirection. Mekky et al. [44] presented a method to identify malicious chains of HTTP redirections using supervised decision tree classifiers.

3.2 The Malvertising Game Model

Our solution aims to apply game theory to formulate the problem of inspecting the malware inserted by the malvertisers into the Web-based advertising system. We define a normal form game of two players, the Attacker and the Defender. The Attacker represents the malvertiser, whose goal is to distribute as many copies of its malware to vulnerable machines as possible when unwitting users visit legitimate websites (i.e. ad publishers). The Defender represents the ad network, whose goal is to detect and remove malicious online ads before they are posted on the ad publishers' websites. We assume that both players are *rational*;

that is, they both aim to maximize their payoffs, and will choose the strategy which is the best response to the strategy chosen by the other player. The Attacker has two strategies, namely "to post a benign ad" (denoted as ***B***) and "to post a malicious ad for distributing malware" (denoted as ***M***). The Defender also has two strategies, namely "to inspect the submitted ad" (denoted as ***I***) and "not to inspect the submitted ad" (denoted as ***No-I***).

Next, we define the payoff functions for each possible combination of the two players' chosen strategies. The notations used in the payoff functions are defined as follows:

- c_m: Attacker's cost of launching malvertising.
- c_i: Defender's cost of inspecting online ads.
- g: Attacker's gain of successful malware distribution through malvertising. We can assume that $g > c_m$ holds because otherwise the Attacker will not have sufficient motivation to post malicious ads.
- l: Defender's loss due to undetected malvertising. We can assume that $l > c_i$ holds because otherwise the Defender will not have sufficient motivation to inspect submitted ads.
- α: probability of Defender detecting malvertising, where $0 \leq \alpha \leq 1$.

Figure 3.1 shows the matrix of the payoff functions under each possible combination of the two players' chosen strategies. In each square, the first value represents the Attacker's payoff, while the second value represents the Defender's payoff. Several payoff functions are straightforward, so we will only explain the payoff functions in the bottom left square. When the Attacker plays strategy ***M*** and the Defender plays strategy ***I***, the Attacker incurs cost c_m for launching malvertising but can get the gain g of successful malware distribution with probability 1-α; the Defender incurs inspection cost c_i but can reduce the loss due to undetected malvertising by αl.

Defender (*D*)

Attacker (*A*)	***I***	***No-I***
B	$0, -c_i$	0, 0
M	$-c_m+(1-\alpha)g,$ $-c_i-(1-\alpha)\ell$	$-c_m+g, -\ell$

Figure 3.1 Payoff functions of the game.

3.3 Finding Nash Equilibrium of the Game

In this section, we discuss the Nash equilibria computed from the game theoretic model. We explain how to find the pure-strategy and mixed-strategy Nash equilibria respectively.

3.3.1 Pure-Strategy Nash Equilibria

According to the payoff functions of each possible combination of strategy chosen by the Attacker and Defender as defined in Figure 3.1, we can compute the Nash equilibria of this game. To this end, we need to first determine the best response of each player toward each strategy chosen by the other player.

For the Attacker, we need to determine his best response to each of the Defender's two possible strategies, namely ***I*** and ***No-I***, respectively. When the Defender plays ***I***, we compare the Attacker's payoff for playing ***B***, which is 0, and playing ***M***, which is $-c_m+(1-\alpha)g$. If $-c_m+(1-\alpha)g \leq 0$, which is equivalent to $\alpha \geq \frac{g-c_m}{g}$, then ***B*** is Attacker's best response to Defender's strategy ***I***. If $-c_m+(1-\alpha)g \geq 0$, which is $\alpha \leq \frac{g-c_m}{g}$, then ***M*** is Attacker's best response to Defender's strategy ***I***. Note that when $\alpha = \frac{g-c_m}{g}$, both ***B*** and ***M*** can be Attacker's best response to Defender's strategy ***I*** according to the definition of best response. When the Defender plays ***No-I***, we compare the Attacker's payoff for playing ***B***, which is 0, and playing ***M***,

which is $-c_m+g$. We can get $-c_m+g > 0$ since $g > c_m$. Thus, ***M*** is Attacker's dominant strategy to Defender's strategy ***No-I***.

For the Defender, we need to determine his best response to each of the Attacker's two possible strategies, namely ***B*** and ***M***, respectively. When the Attacker plays ***B***, we compare Defender's payoff for playing ***I***, which is $-c_i$, and playing ***No-I***, which is 0. Since cost c_i must be positive, hence $-c_i < 0$, we can get that ***No-I*** is Defender's dominant strategy to Attacker's strategy ***B***. When the Attacker plays ***M***, we compare the Defender's payoff for playing ***I***, which is $-c_i-(1-\alpha)l$, and playing ***No-I***, which is $-l$. If $-c_i-(1-\alpha)l-(-l) = -c_i+\alpha l \geq 0$, which is equivalent to $\alpha \geq \frac{c_i}{l}$, then ***I*** is Defender's best response to Attacker's strategy ***M***. If $-c_i+\alpha l \leq 0$, which is $\alpha \leq \frac{c_i}{l}$, then ***No-I*** is Defender's best response to Attacker's strategy ***M***. Note that when $\alpha = \frac{c_i}{l}$, both ***I*** and ***No-I*** can be Defender's best response to Attacker's strategy ***M*** according to the definition of best response.

From the best responses of both players discussed above we can determine the Nash equilibria of this game. If $\frac{c_i}{l} \leq \alpha \leq \frac{g-c_m}{g}$, then the strategy profile (***M***, ***I***) is a pure-strategy Nash equilibrium, because when this condition holds, strategy ***M*** is Attacker's best response to the Defender's strategy ***I***, and strategy ***I*** is also the Defender's best response to the Attacker's strategy ***M***. In the same way, we can derive that if $\alpha \leq \frac{c_i}{l}$, then strategy profile (***M***, ***No-I***) is a pure-

strategy Nash equilibrium. However, if $\alpha > \frac{c_i}{l}$ and $\alpha > \frac{g-c_m}{g}$, then no pure-strategy Nash equilibrium exists. This is because when one player chooses the best response strategy corresponding to the other player's chosen strategy, the latter player will shift to another strategy for it is the best response to the former player's chosen strategy, and then the former player will also shift to another strategy, which forms a loop as demonstrated in the example of the well-known Rock-Paper-Scissors game. However, a mixed-strategy may exist when $\alpha > \frac{c_i}{l}$ and $\alpha > \frac{g-c_m}{g}$, in which the Attacker and the Defender randomize their strategies instead of sticking to the same strategy at all times.

3.3.2 Mixed-Strategy Nash Equilibrium

Next, we show how to derive the mixed-strategy Nash equilibrium of this game. As shown in Figure 3.2, we assume that the Attacker plays strategy ***B*** with probability x and plays strategy ***M*** with probability 1-x, and assume that the Defender plays strategy ***I*** with probability y and plays strategy ***No-I*** with probability 1-y.

To compute x, consider that the Attacker will randomize his choice of strategy to make Defender indifferent between ***I*** and ***No-I***; that is, the expected payoff is the same for the Defender no matter he plays ***I*** or ***No-I***. From Figure 3.2, we get

$$x(-c_i)+(1-x)(-c_i-(1-\alpha)l) = 0x-(1-x)l$$

$$x = \frac{\alpha l - c_i}{\alpha l} \qquad (3.1)$$

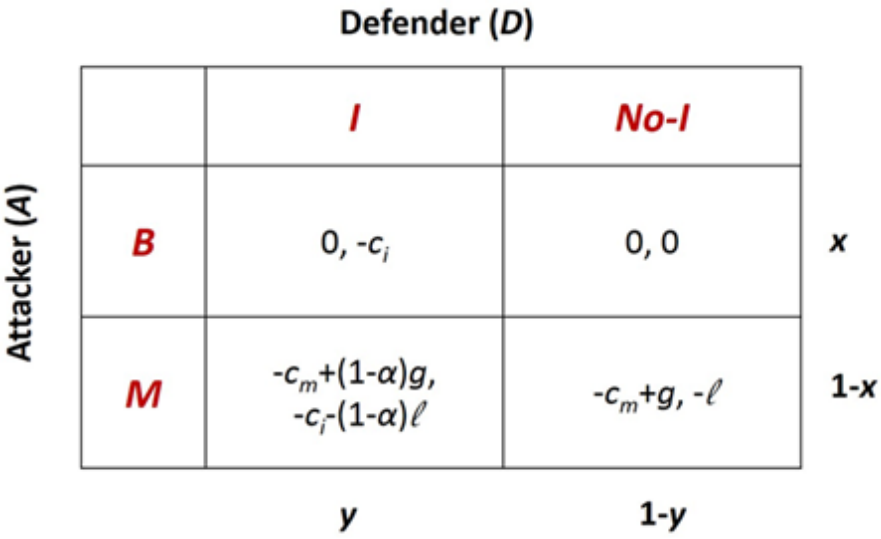


Figure 3.2 Attacker and Defender randomize their choice of strategies.

On the other hand, y can be computed with the consideration that the Defender will randomize his choice of strategy to make Attacker indifferent between ***B*** and ***M***; that is, the expected payoff is the same for the Attacker no matter he plays ***B*** or ***M***. From Figure 3.2, we get

$$y(-c_m+(1-\alpha)g)+(1-y)(-c_m+g) = 0y-(1-y)0 = 0$$

$$y = \frac{g - c_m}{\alpha g} \qquad (3.2)$$

Therefore, we can derive that if $\alpha > \frac{c_i}{l}$ and $\alpha > \frac{g-c_m}{g}$, then the strategy profile {$x\boldsymbol{B}$ + (1-x)$\boldsymbol{M}$, $y\boldsymbol{I}$ + (1-y)$\boldsymbol{No\text{-}I}$} is a mixed-strategy Nash equilibrium, where probabilities x and y are as computed above.

3.4 Evaluation and Analysis

In this section, we discuss the evaluation and analysis of our game theoretic model. We developed a Python program to evaluate our model numerically. The variables needed in the numerical formula for pure and mixed strategy equilibrium are α, c_i, c_m, l and g. We have done the numerical simulations for the Defender's payoff and Attacker's payoff when one of these variables is varied with all the other variables assigned a fixed value. Note that the values used in the simulations are just for the purpose of providing examples and generating charts so that the effects of one variable on another variable can be observed.

3.4.1 Simulations

We give a brief overview of the purpose and results of each simulation as follows. In the first simulation, we aim to observe the effects of detection rate α on the Defender's payoff. We vary α from 0.0 to 1.0 with a step size of 0.05 and all the other parameters remain fixed to calculate the Defender's payoff. The values of the other parameters are chosen as follows: c_i = 0.4, c_m= 0.3, g = 0.9, and l is assigned three different values 0.6, 0.7, and 0.8 in order to obtain three curves

based on l. Figure 3.3 shows that the Defender's payoff remains constant at -l when $\alpha \leq 0.57$ or $\frac{c_i}{l}$, which corresponds to the first case of pure-strategy Nash equilibrium. When $\alpha > 0.57$ and $\alpha \leq 0.66$ or $\frac{g-c_m}{g}$, the Defender's payoff steadily increases. We see a switch from pure strategy to mixed strategy when $\alpha > 0.66$. From this figure, we see that when the detection rate α is low, it has no effect on the Defender's payoff until α exceeds the first threshold ($\frac{c_i}{l}$). After that, the Defender's payoff increases as α continues to increase.

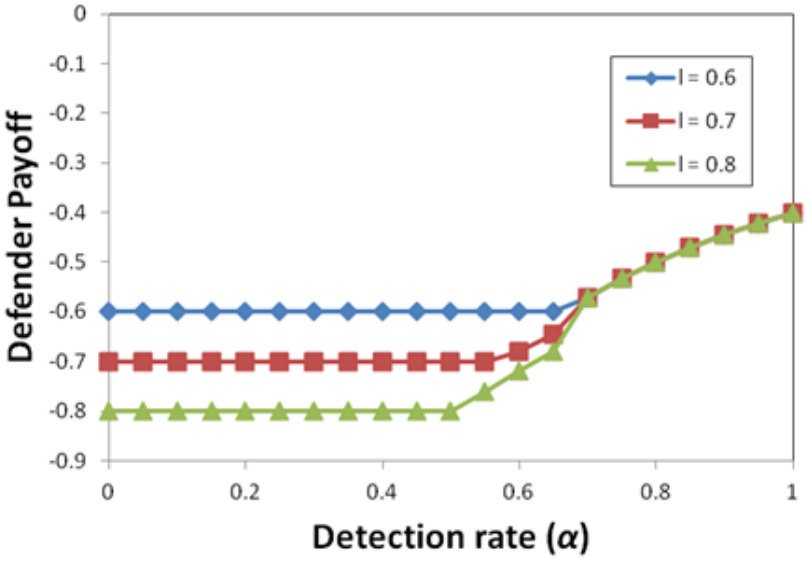


Figure 3.3 Variation in Defender's payoff with α.

In the second simulation, we aim to observe the effects of detection rate α on the Attacker's payoff. We vary α from 0.0 to 1.0 with a step size of 0.05 and all the other parameters remain fixed to calculate the Attacker's payoff. The values

of the other parameters are chosen as follows: c_i = 0.4, c_m= 0.3, l = 0.7, and g is assigned three different values 0.7, 0.8, and 0.9 in order to obtain three curves based on g. Figure 3.4 shows that the Attacker's payoff remains constant at g - c_m when $\alpha \leq 0.57$ or $\frac{c_i}{l}$, which is the first case of pure strategy Nash equilibrium. When $\alpha > 0.57$ and $\alpha \leq 0.66$ or $\frac{g-c_m}{g}$, the Attacker's payoff sharply comes down to 0.078 and then steadily decreases until it reaches zero. It remains constantly at zero when $\alpha > 0.66$. From this figure, we see that when the detection rate α is low, it has no effect on the Attacker's payoff until α reaches the first threshold ($\frac{c_i}{l}$). Then, there is a sharp drop in the Attacker's payoff. As α continues to increase, Attacker's payoff continues to decrease until it reaches zero.

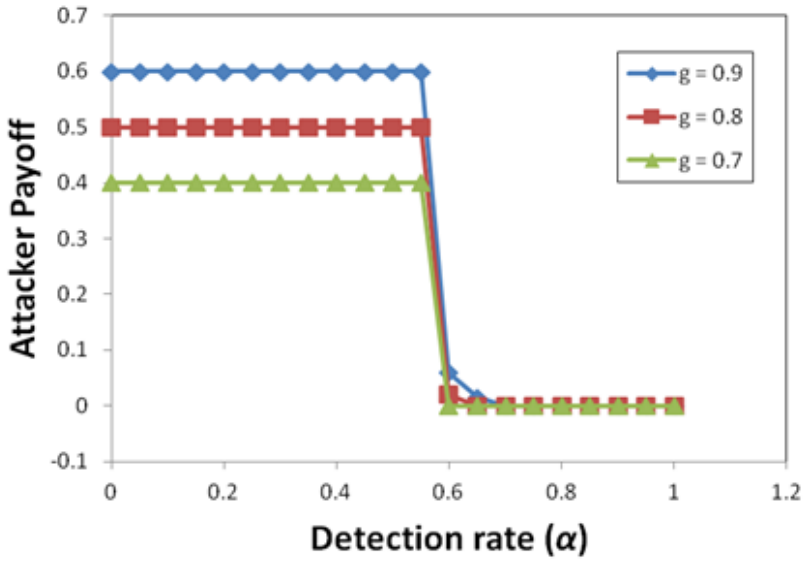


Figure 3.4 Variation in Attacker's payoff with α.

In the third simulation, we aim to observe the effects of the Defender's cost c_i on the Defender's payoff. We vary Defender's cost c_i and all the other parameters remain fixed to calculate Defender's payoff. We vary c_i from 0.0 to 0.65 with a step size of 0.05 (c_i stops at 0.65 since according to the assumption in Section 3.2, c_i must be less than l, which is assigned as 0.7 here). The values for the other parameters were as follows: $c_m = 0.3$, $l = 0.7$ and $g = 0.9$. The value of α is assigned 0.3, 0.5, 0.7 to obtain three different curves. We can see in Figure 3.5 that for all three curves, when $\alpha > \frac{c_i}{l}$, defender's payoff steadily decreases and reaches the constant value of $-l$. It remains at this value when $\alpha \leq \frac{c_i}{l}$.

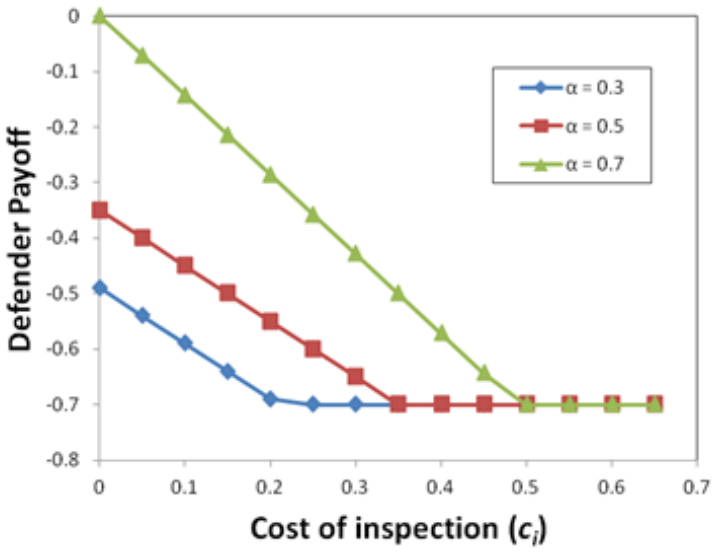


Figure 3.5 Variation in Defender's payoff with c_i.

Figure 3.6 shows the results for the simulation of Attacker's payoff vs. Attacker's cost c_m. The fixed parameters c_i, l, and g have the same values as in

previous simulations, with only c_m being varied from 0.0 to 0.85. The value of α is assigned 0.3, 0.4, 0.5, 0.6, 0.7 to obtain five different curves. We see that as the Attacker's cost c_m increases, the payoff linearly decreases until it reaches 0. The starting point of each payoff curve (i.e. when $c_m = 0$) depends on the value of the detection rate α. The higher the value of α, the lower the starting value of the payoff.

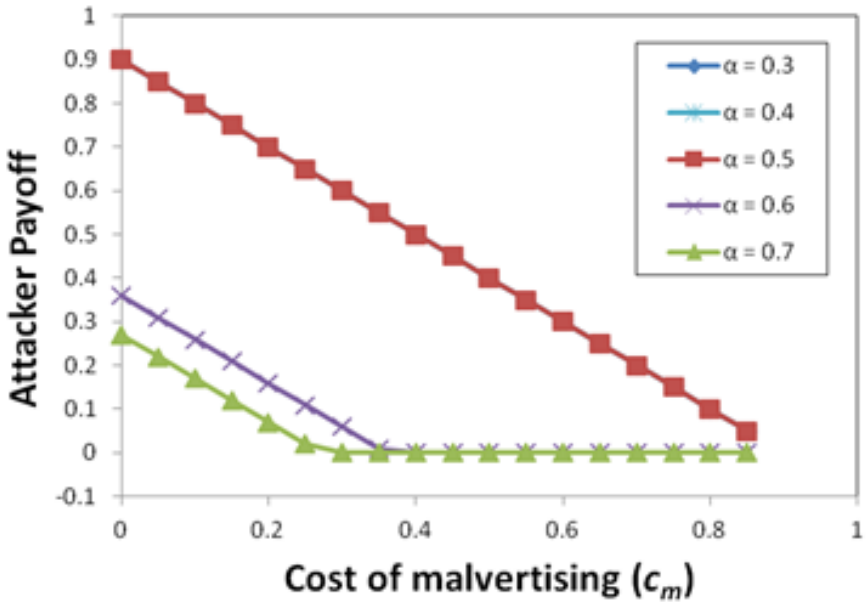


Figure 3.6 Variation in Attacker's payoff with c_m.

Figure 3.7 shows the results for the simulation of Defender's payoff vs. Defender's loss l. The fixed parameters c_m, c_i, and g have the same values as in previous simulations, with only l being varied from 0.45 to 1.25 (l starts from 0.45

since according to the assumption in Section 3.2, l should always be greater than c_i, which is assigned as 0.4). The value of α is assigned 0.3, 0.4, 0.5, 0.6, 0.7 to obtain five curves. We see that as the Defender's loss l increases, the payoff linearly decreases. According to the Nash equilibria we derived, there is a switch of strategies for the Defender from ***No-I*** to ***I*** in the middle depending on the value of α. After the point of switch, the rate of decrease in the Defender's payoff slows down. The higher the value of α, the higher the change in the rate of decrease in the payoff. We see that when α = 0.7, the payoff becomes almost constant after the switch of strategies.

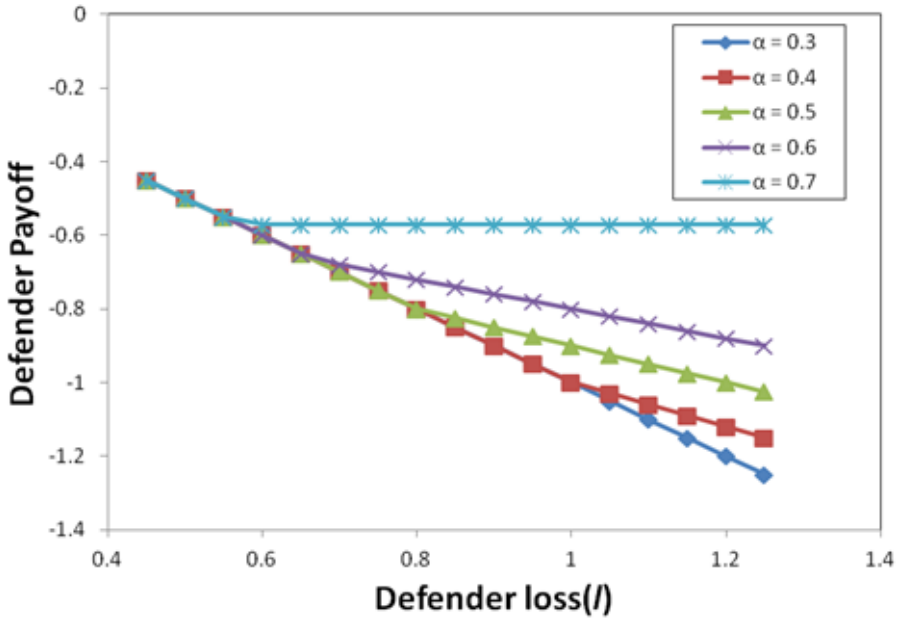


Figure 3.7 Variation in Defender's payoff with l.

Figure 3.8 shows the results for the simulation of Attacker's payoff vs. Attacker's gain g. The fixed parameters c_m, c_i, and l have the same values as in previous simulations, with only g being varied from 0.35 to 1.25. The value of α is assigned 0.3, 0.4, 0.5, 0.6, 0.7 to obtain five curves. We see that as g increases, the attacker's payoff always increases at a constant rate when α is lower (0.3, 0.4, or 0.5). However, for a higher α (0.6 or 0.7), the payoff remains zero when g is not high enough, and starts to rise only when g is higher than a threshold.

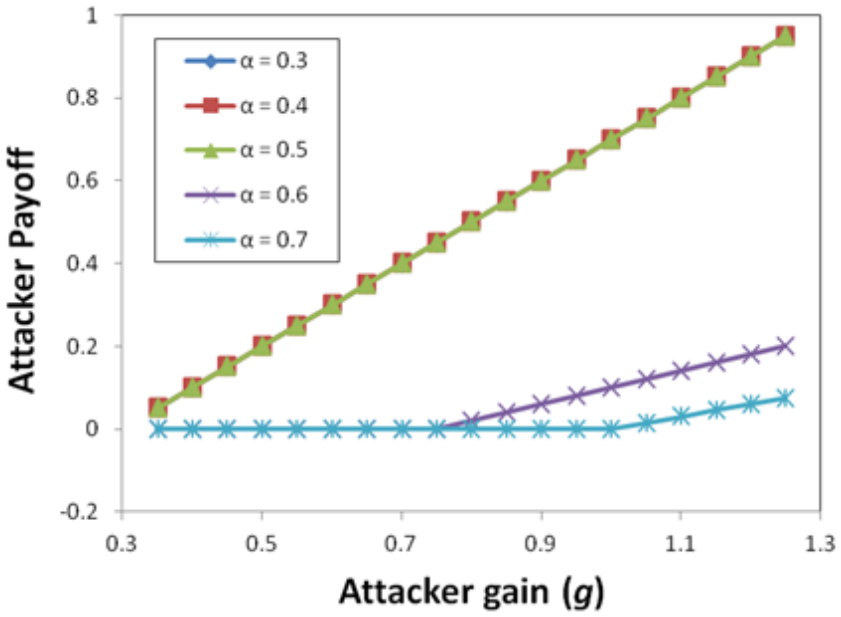


Figure 3.8 Variation in Attacker's payoff with g.

3.4.2 Analysis of the Game Theoretic Model

From the analysis of the payoff functions of the Attacker and Defender, the conditions of each Nash equilibrium, and the results of above simulations, we can derive the following insights:

1. This game is not a zero-sum game, because the Attacker's gain does not come from the Defender's loss.

2. Although performing inspection (playing strategy ***I***) will not bring the Defender any positive gain, it will lower his loss if he can detect the malicious ads with a sufficiently high rate. Therefore, the Defender is still motivated to inspect the submitted ads before letting them pass and be posted on ad publisher's website.

3. If the detection rate is too low ($\alpha \leq \frac{c_i}{l}$), then the Defender will just choose not to inspect the ads. This is because in this case the reduction of Defender's loss due to inspection is less than the cost spent on inspection, and thus will not lower the overall cost.

4. If the detection rate is not high enough ($\alpha < \frac{g-c_m}{g}$), then the Attacker will always post malicious ads. This is because that although some malicious ads submitted by the Attacker will be detected by the Defender's inspection techniques, the gain brought in by those malicious ads

successfully delivered to vulnerable user machines is still higher than the cost of launching malvertising.

5. If the detection rate is high enough ($\alpha > \frac{g-c_m}{g}$ and $\alpha > \frac{c_i}{l}$), then the Attacker and Defender start to randomize their choice of strategy because no pure-strategy Nash equilibrium exists.

6. Assume that the detection rate (α) is within the same range as given in point 5 (i.e. $\alpha > \frac{g-c_m}{g}$ and $\alpha > \frac{c_i}{l}$). Provided that everything else is constant, higher α will make the Attacker incline more to post benign ads (from Equation (3.1) in Section 3.3.2, we can get that x increases when α increases), and make the Defender incline more to not inspect the ads (from Equation (3.2) in Section 3.3.2, we can get that y decreases when α increases).

7. Assume that the detection rate (α) is within the same range as given in point 5, and the Defender has knowledge of the Attacker's average gain (g) resulting from each successful delivery of malicious ad. Provided that everything else is constant, higher g will make Defender incline more to inspect (from Equation (3.2) in Section 3.3.2, we can get that y increases when g increases).

8. Assume that the detection rate (α) is within the same range as given in point 5, and the Attacker has knowledge of the Defender's average loss (l) resulting from each undetected malicious ad. Provided that everything else is constant, higher l will make Attacker incline more to post benign ads (from Equation (3.1) in Section 3.3.2, we can get that x increases when l increases).

3.5 Related Work

Researchers have proposed complete defense systems to counter malvertisements as well. Ford et al. [45] developed a tool that can automatically analyze Flash advertisements to identify malicious behavior. Li et al. [46] presented MadTracer, a malvertising detection system based on machine learning techniques that learn and identify prominent features from malicious advertising nodes and their related content delivery paths. MadTracer can automatically generate detection rules and utilize them to detect malvertising activities. Rastogi et al. [47] developed a framework for analyzing the app-web interfaces in Android applications and successfully analyzed 201 ad networks and their associated ad library packages and 600,000 apps in the Google Play store and identified hundreds of malicious files and scam campaigns. Their scheme involves triggering of the app-web interfaces, detection of malicious

content, and provenance to identify the responsible parties. Arshad et al. [48] proposed an in-browser approach called Excision to automatically detect and block malicious third-party content inclusions as the user's browser loads web pages or executes browser extensions. They claimed that their approach does not rely on the inspection of the resources' content; rather, it relies on analyzing the sequence of inclusions that leads to the resolution and loading of a final third-party resource.

Researchers have previously applied the game theoretic approach to combat other similar malicious threats. Njilla et al. [49] proposed a game theoretic framework to model the security and trust relationship in cyberspace among users, service providers and attackers. The authors formulated a three-player game and analyzed different solutions obtained from Nash equilibrium that can benefit the service providers in decision making. Kamhoua et al. [50] proposed a game-theoretic approach for testing for hardware Trojans in digital circuits where the testing is modeled as a zero-sum game between malicious manufacturers or designers who want to insert Trojans, and testers whose goal is to detect the Trojans. The resulting solution involves multiple possible mixed strategy Nash equilibria that can provide guideline for optimum test sets for identifying and preventing hardware Trojans. Similar game theoretic approaches have been used in [51, 52, 53, 54].

3.6 Summary

Malvertising has posed serious security threats to the Internet, and caused losses to Internet users and ad networks alike. In this work, we formulated the malvertising inspection problem with a game theoretic model, and introduced a normal form game between the malvertiser and the ad network. To the best of our knowledge, this is the first attempt to apply game theory to model this problem. We computed pure-strategy and mixed-strategy Nash equilibria for the two players, and derived several useful insights from analysis of the game. Our findings can provide guidelines for ad networks to best utilize their resources to mitigate the problem of malvertising.

In the future, we aim to extend our game theoretic model to consider the repeated Bayesian game between the malvertiser and the ad network. The main characteristic of a Bayesian game is that one or both of the players have incomplete information about the type of the other player, which will allow us to model the scenario when the ad network has incomplete information to determine whether the advertiser belongs to the benign type or the malicious type. Moreover, repeated game will allow the players to incorporate the information they learned in previous games into the playing of future games.

CHAPTER 4

MAXIMIZING ACCURACY IN MULTI-SCANNER MALWARE DETECTION SYSTEMS

4.1 Problem Formulation

In this section, we have formulated the problem of maximizing accuracy in a multi-scanner detection system using appropriate formal notations. Table 4.1 lists some of these notations used in the formulation. Formally, the problem of maximizing accuracy in a multi-scanner detection system can be stated as follows:

Given N scanners along with their respective (true positive and false positive) detection rates or probabilities P_i (where $1 \leq i \leq N$) and binary detection results (either true or false) for a given sample obtained from these N scanners., how can we find the optimum value of T ($1 \leq T \leq N$) where T is the threshold to decide maliciousness of that given sample. Here, we assume the N is a finite number and we only have the detection rates or probabilities associated with each scanner that can be calculated from past detection history of the scanners.

Table 4.1 Notations

Symbol	*Description*
I	Input to the multi-scanner system
O_i	Output of i^{th} scanner (0 or 1)
N	Total number of scanners
Q	Optimum number of scanners to achieve maximum accuracy
T	Threshold to decide the maliciousness of an object
P_i	Detection probability of i^{th} scanner
P^T_i	The probability of classifying a malicious object as malicious by i^{th} scanner
P^F_i	The probability of classifying a benign object as malicious by i^{th} scanner
$CP(t)$	Combined detection probability when $T = t$

The problem can be extended further to answer the following questions:

1) Assuming that N is the total number of scanners that we can use and Q is the optimum number of scanners to achieve maximum accuracy, what is the relationship between N and Q? Is $N = Q$ always holds, or $Q < N$ can also be true? In other words, does adding another scanner always improve accuracy?

2) If M is the size of a subset of all N scanners, how do we select these M scanners to achieve maximum accuracy that is possible for any subset of scanners of size M. In other words, given that there can be $\binom{N}{M}$ of combinations possible, how can we rank all the scanners to select the best M scanners such that it will provide maximum accuracy among all these combinations possible?

4.2 Combined Probability Model (CPM)

In this section, we will explain the development of the *Combined Probability Model (CPM)* in detail. As mentioned earlier, we have devised a set of formula to construct the model. In the formulas, we used certain symbols and notations to denote various terms. Table 4.1 lists these notations. To help better understand the model, we will start with a small scaled model consisting only 3 scanners. Then, we will extend the small scaled model to a more generalized version.

4.2.1 3-Scanner CPM

We start with a simple 3-scanner model (N=3) to better illustrate and explain the method of developing the generalized model. The most generic multi-scanner system consisting 3 scanners should be a parallel system of scanners, depicted as in Figure 4.1. A parallel system of scanners is a system of scanners where each input sample is fed to all the scanners in parallel and at the same time. We

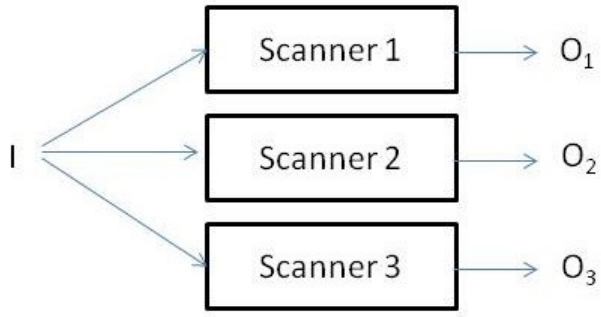


Figure 4.1 A 3-scanner parallel system.

assume here that all the scanners are binary scanners, i.e. they produce an output of either 1 or 0, where a 1-output means the sample is detected as malicious and 0-output means the sample is detected as benign.

To decide maliciousness of an input object, we have 3 choices here. We can label the object as malicious if (i) all three scanners label it as malicious, (ii) any two of them label it as malicious, or (iii) any one of them labels it as malicious. This is equivalent to considering the value of T as 3, 2 and 1 respectively.

Now, there are two distinct probabilities associated with each scanner – P^T and P^F. P^T is used to calculate the overall true positive probability and P^F is used to calculate the overall false positive probability. For the sake of generality, we will only use the notation P to denote a particular probability here.

To understand how we can come up with the equations, we have to break down each case into smaller parts. For example, if we consider $T = 1$, this means that if any single scanner detects the sample, we can consider that sample as detected and label it as malicious. Now, let us assume X denotes the random variable that is defined as the number of scanners that detect a given sample as malicious. Then, for $T = 1$, the combined probability can be derived as

$$CP(1) = P\{X \geq 1\},$$

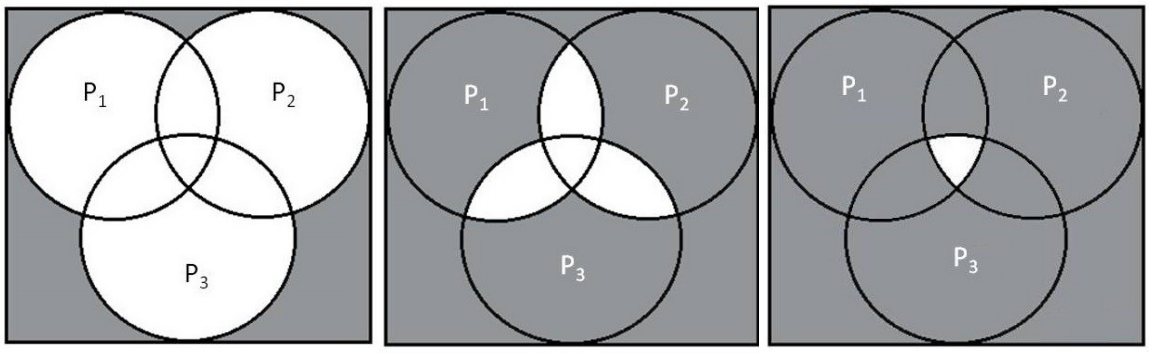
which in turn can be written as

$$CP(1) = P\{X = 1\} + P\{X = 2\} + P\{X = 3\}.$$

In other words, the probability of a sample being detected by at least one scanner is a summation of the probability of that sample being detected by exactly 1, 2 and 3 scanners. This is also depicted in Figure 4.2(a), where we can see the total white region consists of three types of smaller regions which depict three components of the summation in the above equation. Therefore, we can generalize this equation for $T = t$ (where $1 \leq t \leq 3$) as

$$CP(t) = \sum_{i=t}^{3} P\{X = i\}. \qquad (4.1)$$

Now, we have to find out how to calculate the probability $P\{X=i\}$. Let's start with $P\{X=1\}$. This means, we have to calculate the probability that exactly one scanner will detect the sample. We have the individual detection probabilities as P_1, P_2, and P_3 for *scanner 1*, *scanner 2* and *scanner 3* respectively.



(a) (b) (c)

Figure 4.2 Venn diagrams for the three cases.

P{X=1} can be described as the summation of the probabilities that only *scanner 1* detects the sample, only *scanner 2* detects the sample and only *scanner 3* detects the sample. Now, according to the rules of probabilities, we can say that the probability that only *scanner 1* detects the sample is $P_1(1-P_2)(1-P_3)$. Similarly, for *scanner 2* and *scanner 3* the probabilities will be $P_2(1-P_3)(1-P_1)$ and $P_3(1-P_1)(1-P_2)$ respectively. Therefore, we can write

$$\begin{aligned} P\{X=1\} = \; & P_1(1-P_2)(1-P_3) \\ & + P_2(1-P_3)(1-P_1) \\ & + P_3(1-P_1)(1-P_2). \end{aligned} \qquad (4.2)$$

Following similar reasoning, we can write

$$\begin{aligned} P\{X=2\} = \; & P_1P_2(1-P_3) \\ & + P_2P_3(1-P_1) \\ & + P_3P_1(1-P_2). \end{aligned} \qquad (4.3)$$

and

$$P\{X=3\} = \; P_1P_2P_3. \qquad (4.4)$$

The reasoning behind these equations is also illustrated in Figure 4.2. Replacing the values from equations (4.2), (4.3) and (4.4) into equation (4.1), we can easily calculate the combined probability (*CP*) for a given $T = t$.

4.2.2 N-Scanner CPM

In the previous section, we limited our discussion to only 3 scanners for ease of understanding. Now, we can extend this 3-scanner model to an *N*-scanner model. Figure 4.3 shows an *N*-scanner system.

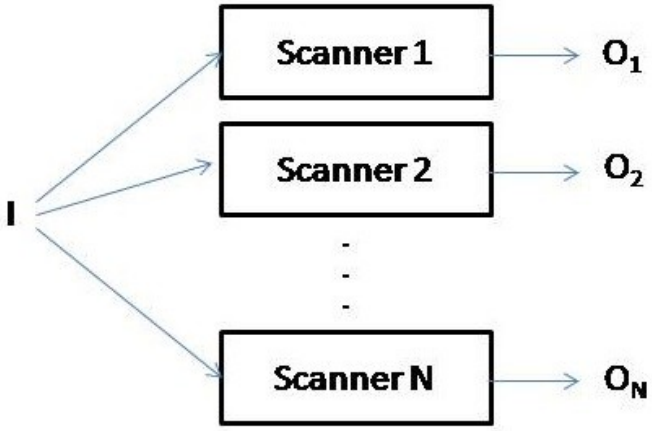


Figure 4.3 An *N*-scanner parallel system.

For an *N*-scanner model with $T = t$ (where $1 \le t \le N$), equation (4.1) becomes

$$CP(t) = \sum_{i=t}^{N} P\{X = i\}. \qquad (4.5)$$

Based on equations (4.2), (4.3) and (4.4), we can come up with a generalized *N*-scanner equation for the probability *P{X=i}* as

$$P\{X = i\} = \sum_{j=1}^{\binom{N}{i}} \prod_{k=1}^{i} P_k^j \prod_{l=i+1}^{N} (1 - P_l^j), \qquad (4.6)$$

where P_k^j is the probability of the scanner with index k $(1 \le k \le i)$ in j^{th} combination in $\binom{N}{i}$ and P_l^j is the probability of the scanner with index l $(i+1 \le l \le$

N) in all the other scanners that are not in j^{th} combination. Substituting the value of *P{X=i}* from equation (4.6) into equation (4.5) we get

$$CP(t) = \sum_{i=t}^{N} \sum_{j=1}^{\binom{N}{i}} \prod_{k=1}^{i} P_k^j \prod_{l=i+1}^{N} (1 - P_l^j). \qquad (4.7)$$

Equation (4.7) can be used as the generic *N*-scanner equation for combined detection probability when $T = t$.

4.2.3 CPM for Other Multi-Scanner Systems

So far we have considered only parallel system of scanners. In this section, we will discuss other types of multi-scanner systems such as the serial system and the mixed system and show how they only are special cases of the parallel system of scanners.

4.2.3.1 Serial System

A serial system of scanners is a system of scanners where all the scanners are connected serially, as depicted in Figure 4.4. The input sample is fed into the first scanner and the output from the first scanner is fed into the second scanner and so on. Again, we consider only binary outputs from the scanners. Therefore, by feeding the output into the next scanner, we mean that if the sample is detected as malicious (a 1-output), the sample is passed onto the next scanner to be scanned. This process goes on until the scanner is utilized and only if all the

scanner detect this sample as malicious, it is finally classified as malicious. On the other hand, if the sample is detected as benign (a 0-output), the sample is not passed onto the next scanner and all the subsequent scanners automatically report that sample as benign, eventually classifying the sample as benign.

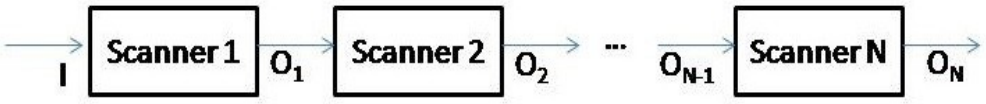


Figure 4.4 An *N*-scanner serial system.

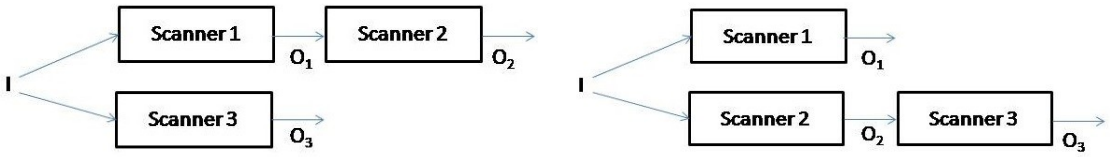
If we compare this system with the parallel system of scanners, we can easily see that this serial system of scanners is nothing but a special case of the parallel system of scanners, where the threshold value T is fixed at the total number of scanners N. This means, only when all the scanners detect a specific sample as malicious, the sample is classified as malicious. In all the other cases, the sample is classified as benign. Therefore, we can use equation (4.7) by just substituting t with N and calculating $CP(N)$.

An alternative version of the serial system is also possible where instead of passing the sample to the next scanner when it is detected as malicious and blocking it when it is detected as benign, we can block it when it is detected as malicious and pass it to the next scanner when it is detected as benign. In this

case, the sample will be detected benign only when all the scanners have detected it as benign and it will be detected as malicious if a single scanner detects it as malicious. Again, if we compare this alternative serial system with the parallel one, we find that this is nothing but a special case of the parallel system where the threshold value *T* is fixed at the value of 1. Therefore, we can use equation (4.7) to calculate *CP(1).*

4.2.3.2 Mixed System

So far, we have seen only pure parallel and serial system of scanners. There also can be a third type of multi-scanner system, where there are both parallel and serial parts in the system. We can call them a mixed system. Consider the systems depicted in Figure 4.5 for a 3-scanner system. The system shown in



(a) (b)

Figure 4.5 Two variations of a 3-scanner mixed system.

Figure 4.5(a) has *scanner 1* and *scanner 2* connected serially, and *scanner 3* is parallel to the serial system of *scanner 1* and *scanner 2*. This system is in fact a parallel system of scanners where one line in the parallel system is a serial

system, which is also a special type of parallel system as we established in section 4.1.3.1. Therefore, we can say that the mixed system is a parallel system consisting of other smaller parallel systems. This means, we can use the same equation (4.7) that we derived for parallel systems to derive the equation for a particular mixed system. Figure 4.5(b) shows another variation of 3-scanner mixed system. For an *N*-scanner system, obviously there can be many more variations possible.

4.3 Greedy Heuristic Based Models

4.3.1 Greedy Approximation Model (GAM)

Instead of deriving a mathematical formula, the *Greedy Approximation Model (GAM)* applies the greedy heuristic to approximately calculate the combined probability *CP(t)* for a given threshold t. Here, the greedy heuristic is to start by combining the highest *t* individual detection probabilities and moving along in a decreasing order doing the same until less than *t* probabilities available. An example would better explain the approach. Let's say we have P_1, P_2, P_3 ... P_N individual detection probabilities available sorted in a decreasing order, that is, $P_1 \geq P_2 \geq P_3 \geq ... \geq P_N$. To calculate *CP(t)*, we initialize *CP(t)* to 0 and calculate $P_1 \times P_2 \times P_3 \times ... \times P_t$ and add to *CP(t)*. For the next iteration, we calculate *1 - CP(t)* and multiply it with $P_2 \times P_3 \times P_4 \times ... \times P_{t+1}$ and add the result to *CP(t)*. This goes on till

we add $P_{N-t+1} \times P_{N-t+2} \times P_{N-t+3} \times \ldots \times P_N \times (1 - CP(t))$ to $CP(t)$. The final value of $CP(t)$ is our desired combined detection probability. We developed the Greedy Approximation algorithm based on this approach, as shown in Figure 4.6. Here, the parameters L_p and t refer to the list of individual detection probabilities and threshold respectively and the resulting combined probability is denoted by CP_t.

Algorithm: GreedyApproximation(L_p, t)

```
1: L_p ← sort(L_p)  //sort list of probabilities L_p
2: CP_t ← 0 //initialize combined probability CP_t
3: CP_c ← 1
4: for i in range(0, N - t + 1) do
5:       m ← CP_c
6:       for j in range(i, i + t) do
7:             m ← m × L_p[j]
8:       end for
9:       CP_t ← CP_t + m
10:      CP_c ← 1 - CP_t
11: end for
12: return CP_t
```

Figure 4.6 The Greedy Approximation algorithm.

4.3.2 Complementary Greedy Approximation Model (CGAM)

The *Complementary Greedy Approximation Model (CGAM)* applies a similar greedy heuristic approach. But instead of applying it on the detection probabilities, it is applied on the complements of the probabilities and again complemented to find the desired combined probability. To understand the reasoning behind this approach, we have to refer back to the Venn diagrams in Figure 4.2. In Figure

4.2(a), we can clearly see that the combined probability of P_1, P_2 and P_3 is shown by the total white region. The area of this white region can be calculated in another way also, that is, by subtracting area of the total grey region from the area of the rectangle. Here, the area of the rectangle represents 1, since this is the universal set, and the area of the grey region is the combined probability of the complements of the probabilities, namely, *(1-P_1)*, *(1-P_2)* and *(1-P_3)*. Figure 4.7 shows the Complementary Greedy Approximation algorithm.

Algorithm: ComplementaryGreedyApproximation(L_p, t)

1: L_c ← [] //initialize complementary list L_c
2: **for each** p in L_p **do**
3: L_c.append(1 - p)
4: **end for**
5: L_c ← sort(L_c)
6: CP_c ← GreedyApproximation(L_c, N - t + 1)
7: CP_t ← 1 - CP_c
8: **return** CP_t

Figure 4.7 The Complementary Greedy Approximation algorithm.

4.4 Accuracy Metrics

The simplest metric is called *Accuracy (ACC)* or *Fraction Correct (FC)* [55]. It measures the fraction of all instances that are correctly categorized and is defined by

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

where *TP*, *TN*, *FP*, and *FN* refers to true positive, true negative, false positive, and false negative respectively. In our experiments, we only calculate *TP* and *FP*. But *TP* and *FN* together make the total number of malicious samples. Similarly, *TN* and *FP* together makes the total number of benign samples. Therefore, we can easily calculate *FN* and *TN* from *TP* and *FP*.

Another useful metric is the *F1 score* [56]. It considers both precision and recall of the test to compute the score and is defined by

$$F1 = \frac{2TP}{2TP + FP + FN} .$$

A third metric, called the *Matthews Correlation Coefficient (MCC)* [57], is used in machine learning as measure of quality of binary classifications. It is generally regarded as a balanced measure and is defined by

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} .$$

4.5 Ranking of Scanners

To identify the best subset of scanners for a given size *M* out of *N* ($1 \leq M \leq N$), we need to rank the scanners based on a suitable criteria that can help in achieving the maximum accuracy and select the top *M* scanners. But the only information about the scanners is their detection rates. Therefore, we need to create an

individual scoring system based on the true positive and false positive detection probabilities for each scanner. Here, we propose to use the accuracy formula (ACC) from section 4.4. Then, individual score for scanner i should be,

$$S_i = \frac{P_i^T + 1 - P_i^F}{2} \quad (8)$$

Based on this score, we can sort all the N scanners in a descending order. Then, to get M best scanners, we can select top M scanners from the ordered set of N scanners.

4.6 Numerical Simulation

To verify the accuracy of our models and to answer the questions mentioned in section 4.1, we performed several numerical simulation experiments. We used Python to develop small programs that can simulate the scanning of a set of samples by a set of anti-virus scanners. In this section, we will describe the setup of these experiments and their results in detail.

4.6.1 Simulation of the Models

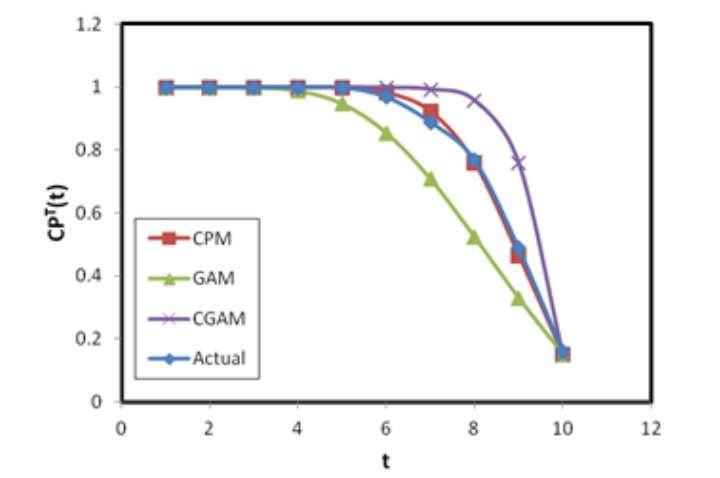
We defined a hypothetical set of 1000 malicious and 1000 benign samples and 10 anti-virus scanners. We randomly decided whether a particular sample is detected as malicious or not by a particular anti-virus scanner. Then, we calculated the true positive rate and false positive rate for each anti-virus scanner

as well as the combined true positive rate and false positive rate for all the threshold values ranging from 1 to 10. We did the same using our models as well. Then, we calculated the accuracy values both for the actual case and for our models based on three metrics of evaluation, as described in section 4.4.

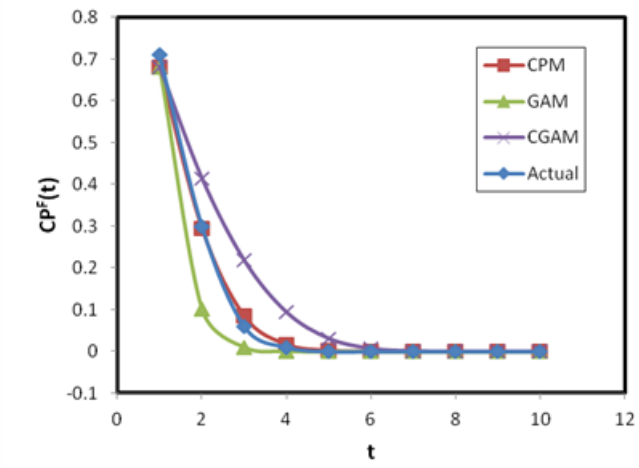
As mentioned earlier, we randomly decided whether a sample is detected as malicious or not by an anti-virus scanner. To create different test sets with different detection rates for the anti-virus scanners, we enforced different maximum values so that all the anti-virus scanners will have a detection rate that is below the maximum value for that test set. This means, for example, if the maximum value is 90, all the anti-virus scanners (10 in our experiments) will have a maximum detection rate of 0.9 or 90%. We varied the maximum value to create all the test sets spanning all possible detection rates. The range of maximum values for true positive rates was from 50 to 95 and the range of maximum values for false positive rates was from 5 to 50.

To better illustrate our simulation results, we show the graphs of one specific test case, where the true positive rate was limited to 80% and the false positive rate was limited to 10%. Figure 4.8(a) shows the graphs of combined true positive rates generated from the actual case and the models for different threshold values ranging from 1 to 10. Similarly, Figure 4.8(b) shows the graphs

combined false positive rates calculated from actual case and our models for different threshold values.



(a)



(b)

Figure 4.8 Graphs of combined detection probabilities against different threshold values.

Figure 4.9 shows the comparison of accuracy values resulting from the actual values estimated using the actual optimum threshold and also using the

threshold calculated from our models for the example test case using three different evaluation metrics. We have also included the minimum accuracy levels to show how our model predicated accuracy values perform against them. The graph clearly indicates that all of the model predicted accuracy values are very close to the actual maximum accuracy values.

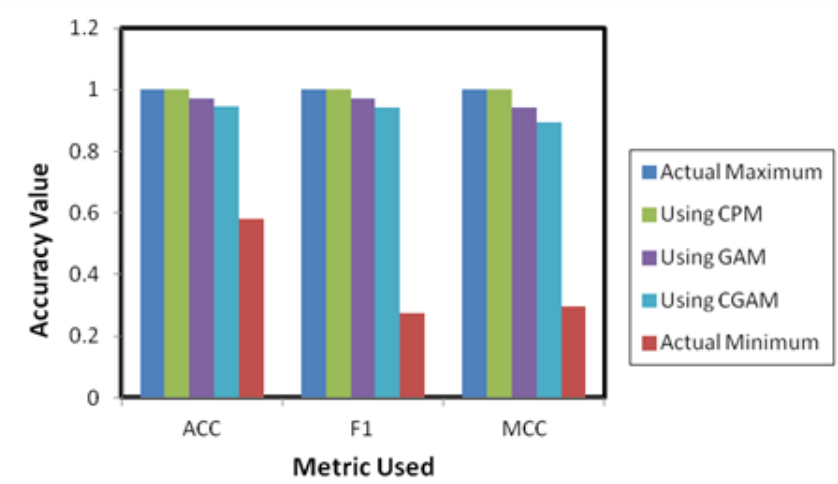


Figure 4.9 Comparison of accuracy values using three evaluation metrics based on simulation results.

To evaluate how our models perform against the actual cases, we varied the limiting maximum values for randomization and created different test cases. As mentioned earlier, the range of limiting maximum values for true positive rates was from 50 to 95 and the range for false positive rates was from 5 to 50. We varied the values with a step size of 5, creating total 10 × 10 = 100 test cases. Table 4.2 shows average deviation from the actual maximum accuracy value for all

three models based on three evaluation metrics we used. Results from Table 4.2 indicate that CPM performs best among the models.

Table 4.2 Average Deviation from Maximum Accuracy

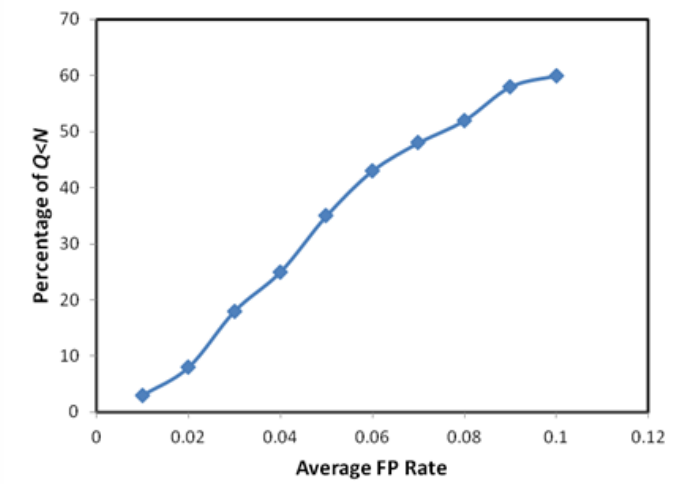
Metric Used	*CPM*	*GAM*	*CGAM*
ACC	0.03	0.1	0.14
F1	0.04	0.12	0.15
MCC	0.06	0.18	0.26

4.6.2 Simulation of Optimum Size for Scanner Set (Q)

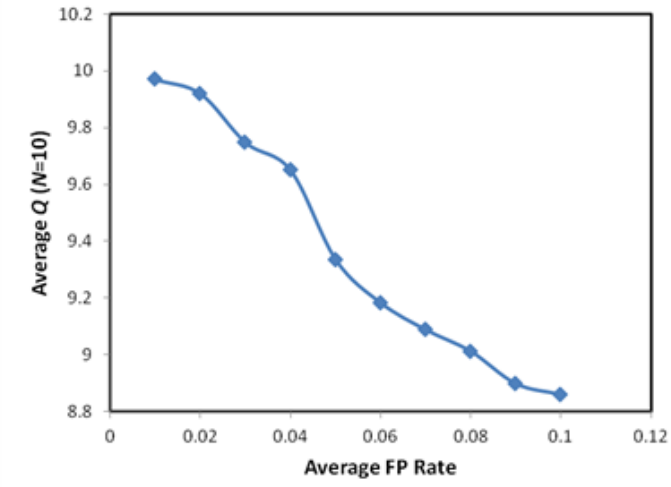
The optimum size of the scanner set refers to the number of scanners in a scanner set that achieves the maximum accuracy value among all available N scanners. We have denoted it here as Q. Here, the goal of our simulation test is to determine whether adding new scanners to a multi-scanner system can always improve or maintain the maximum accuracy. In other words, if we have a total of N scanners available, we want to answer the following question - should we use all of them to achieve maximum accuracy ($Q = N$), or is it possible to reduce the number of scanners needed to achieve maximum accuracy by removing some scanners from the set ($Q < N$)?

In the simulation test, we vary the average false positive detection rate of the scanners and calculate the value of Q. The value of N is selected as 10 like before. The value of average false positive rate is varied from 0.01 to 0.1 with a

step size of 0.01. We run the tests for each average false positive rate value 100 times to get an average estimate. Figure 4.10 (a) shows the percentage of times Q is less than N out of all instances as we increase the average false positive rate of scanners.



(a)



(b)

Figure 4.10 Trends of changes in Q vs. average false positive rate.

We can see from the graph that the increase is almost linear and it increases up to more than 50% when the average false positive rate is increased up to 0.1. Figure 4.10 (b) shows the calculated average values of Q when N is 10, as we increase the average false positive rate. The graph shows that the average value of Q almost linearly decreases with the increase in average false positive rate. Both of these graphs in Figure 4.10 verifies the fact that if the false positive rate of the scanners are high enough, the number of scanners that will yield the maximum accuracy can be lower than the total number of available scanners. In other words, with a high enough false positive rate, it is not always beneficial to add new scanners to the set of scanners in a multi-scanner system.

4.6.3 Simulation of the Ranking Approach

In section 4.5, we proposed a ranking system based on the accuracy score of individual scanners to rank all the scanners and take top M to create a subset of scanners. We performed simulation experiments to test how the performance of this ranked subset fit into the range of maximum accuracy values achieved by any M scanner subset.

Figure 4.11 shows the graph for a sample simulation test done to compare the maximum accuracy values achieved by best combination, worst combination and the combination consisting of top ranked scanners. The individual scanner

true positive and false positive detection rates were randomized like the previous simulation tests and were limited to a highest value. In this test case, true positive rates were limited to 80% and false positive rates were limited to 5%. We can see from the graph that our ranking approach does much better than the worst combination selected and performs almost at the same level as the best combination for higher *M* values.

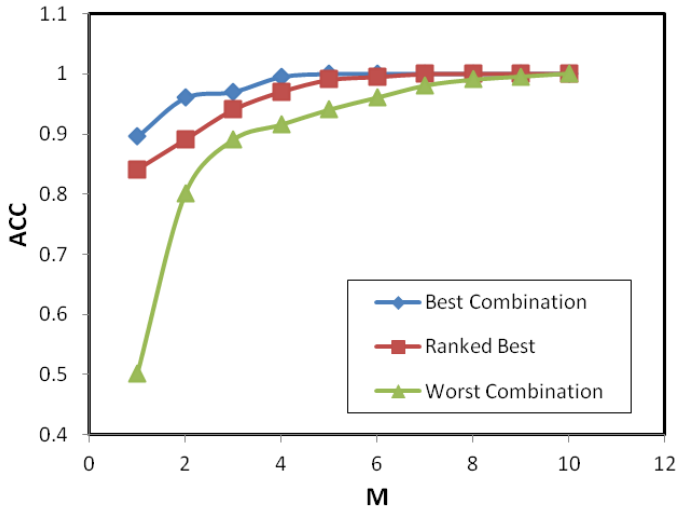


Figure 4.11 Comparison of maximum accuracy by best, worst and ranked best combinations based on simulation results.

We executed similar simulation test 100 times to get an average estimate of how our ranking approach performs. We found that on average our ranking approach provides a combination that achieves a accuracy value that is 0.0195 lower than the maximum accuracy achieved by the best combination and 0.0655

higher than the maximum achieved by the worst combination. Here, we have only included the evaluation results done using the first metric (*ACC*). Similar evaluation could be done using the other two metrics as well.

4.7 Experimental Evaluation Using Real Data

4.7.1 Malware and Goodware Dataset

We collected a large data set of malware samples from VirusSign [58], which generously provides with a significant amount of malware samples everyday in return of a small payment. Our malware dataset consisted 38,789 malware samples in total. Our goodware dataset consisted of 21624 benign portable executable (PE) binary files collected from SourceForge [59]. We downloaded these files by crawling the SourceForge website in order of user rating to ensure they are not malicious. Table 4.3 lists the details of each of the malware and goodware datasets.

Table 4.3 Malware and Goodware Dataset

Name	*Source*	*Number of Samples*	*Period of Collection*
Malware Dataset	VirusSign	38,789	April 26 to April 29, 2014
Goodware Dataset	SourceForge	21,624	July 20 to July 31, 2015

We divided both the malware and goodware dataset further into training and test sets. The training datasets are used to calculate individual true and false detection probabilities (P^T and P^F) for each anti-virus scanner. These values are used by our models to calculate combined detection probabilities ($CP^T(t)$ and $CP^F(t)$) according to our CPM formula (equation (4.7)) and GAM and CGAM algorithms. Then, the test datasets are used to calculate the actual combined detection probabilities ($CP^T(t)$ and $CP^F(t)$) for each threshold t. Table 4.4 lists the division of malware and goodware dataset into corresponding training sets and test sets. We used multiple test sets of varying sizes by dividing the full test set to add diversity into the experiments.

Table 4.4 Training and Test Sets

Name	*Number of Samples*
Malware Training Set	28,789
Malware Test Set	10,000
Goodware Training Set	11,624
Goodware Test Set	10,000

4.7.2 Experimental Setup

We used online multi-scanning service VirusTotal for our experiments. VirusTotal generates scanning reports based on scanning performed by at most 55 anti-virus scanners (at the time of the writing). But not all the reports contain

the same anti-virus scanners all the time. This is why we had to identify a set of anti-virus scanners that are common to all the generated scanning reports. We found that 21 anti-virus scanners (listed in Table 4.5) were common to all the scanning reports.

Table 4.5 List of Anti-virus Scanners

Kaspersky	ESET-NOD32	VBA32
Antivir	GData	VIPRE
Agnitum	Ikarus	TrendMicro-HouseCall
Avast	K7GW	BitDefender
AVG	McAfee-GW-Edition	Emsisoft
Comodo	Malwarebytes	NANO-Antivirus
DrWeb	Sophos	Panda

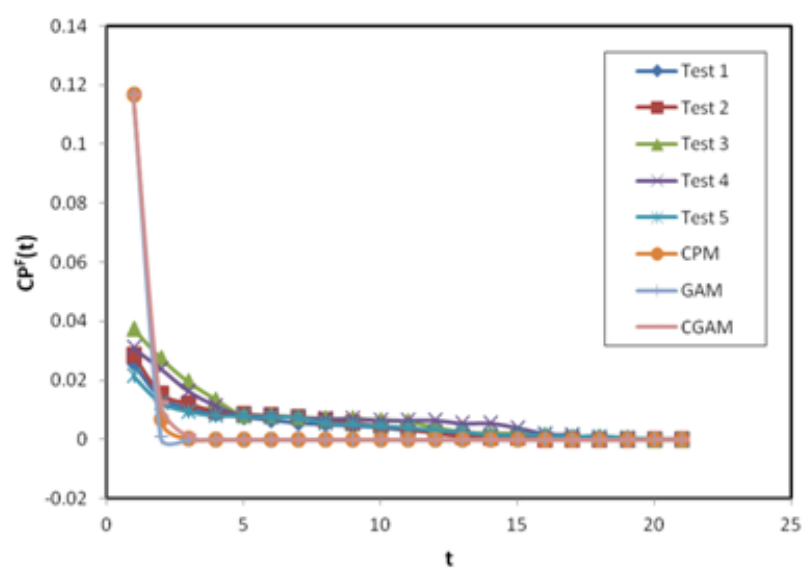
To implement the experiment, we developed a small program in C#.NET that is based on the VirusTotal API to generate the scanning reports from VirusTotal and another small program in Python to parse and calculate our desired combined detection probability and accuracy values from them. We also implemented our models using Python.

4.7.3 Results and Analysis

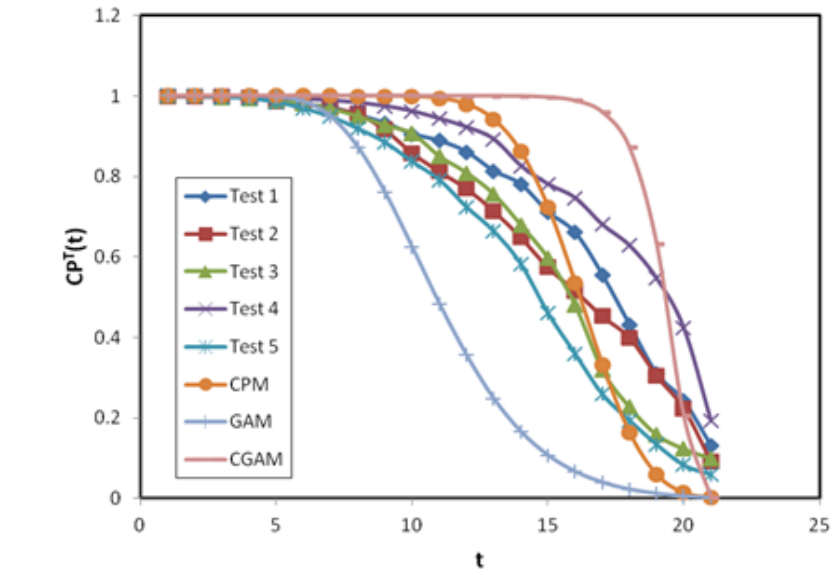
Figure 4.12(a) shows the graphs of combined true positive detection probability ($CP^T(t)$) against threshold values (t) from 1 to 21. Here, we have divided the full test set (both malware and goodware) into 5 test sets containing 2000 samples

each. From the graphs, we can see that the actual combined true positive detection rate varies from test to test. Among the graphs generated from the models, CPM shows least amount of deviation from the actual trend. The other two (GAM and CGAM) graphs deviate further in opposite directions. A similar trend can be found in Figure 4.12(b), which demonstrates the graphs of combined false positive detection probabilities ($CP^F(t)$) against threshold values (t) from 1 to 21.

We use these combined true and false positive detection probabilities to calculate accuracy values according to three evaluation metrics from section 4.4 and use them to determine the optimum threshold. To add diversity in test sizes, we created 3 test sets from the malware and goodware test set according to Table 4.6. Figure 4.13 shows the comparative graphs for these accuracy values for each test set. The accuracy values calculated using the models are actually the actual accuracy values for the model predicted optimum thresholds. Figure 4.13(a), 4.13(b) and 4.13(c) presents the comparative accuracy values for test set 1, test set 2 and test set 3 respectively. We can see that for all the test cases, the model predicted accuracy values are very close to actual maximum accuracy values. We also see that there is a very small difference in accuracy values among CPM, GAM and CGAM, where CPM and GAM perform better in comparison to CGAM.

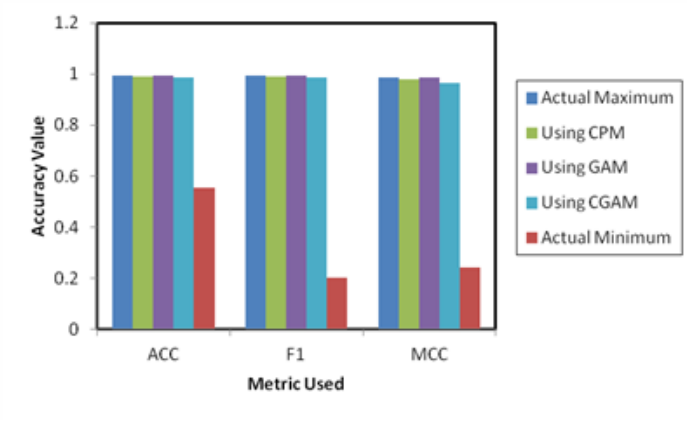


(a)

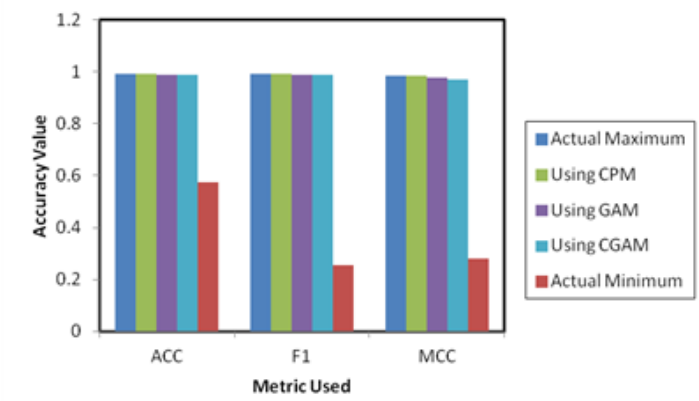


(b)

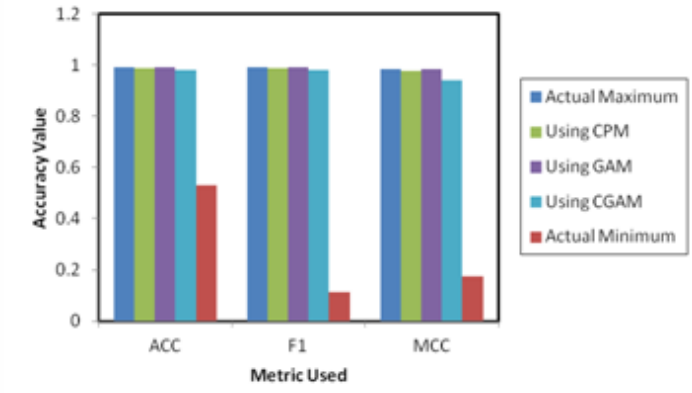
Figure 4.12 Comparison of graphs of combined detection probabilities against threshold values generated from actual test cases and our models.



(a)



(b)



(c)

Figure 4.13 Comparison of accuracy values using three evaluation metrics based on real world (a) test set 1, (b) test set 2, and (c) test set 3.

Table 4.6 Distribution of Test Sets for Combined Accuracy Test

Test Set	*Number of Malware Samples*	*Number of Goodware Samples*
1	4000	4000
2	4000	2000
3	2000	4000

Next, we perform all combination tests where we take a subset of M scanners from all N scanners and calculate maximum accuracy values for the best combination, the worst combination and the combination from top ranked scanners. Figure 4.14 shows the graphs for this experiment done only on the test set 1 from Table 4.6. The results for test set 2 and 3 also yield similar results and omitted for space constraints. In Figure 4.14, we see that the ranking approach yields accuracy values that are very close to the maximum accuracy values achieved by the best combination and much higher than the maximum accuracy achieved by the worst combination. We have also calculated an average among all 3 test sets to find out the average difference of the accuracy values for the combinations. We found that on average the maximum accuracy value calculated using the ranking approach is lower than the maximum accuracy for the best combination by 0.00164 and higher than the maximum accuracy for the worst combination by 0.05468.

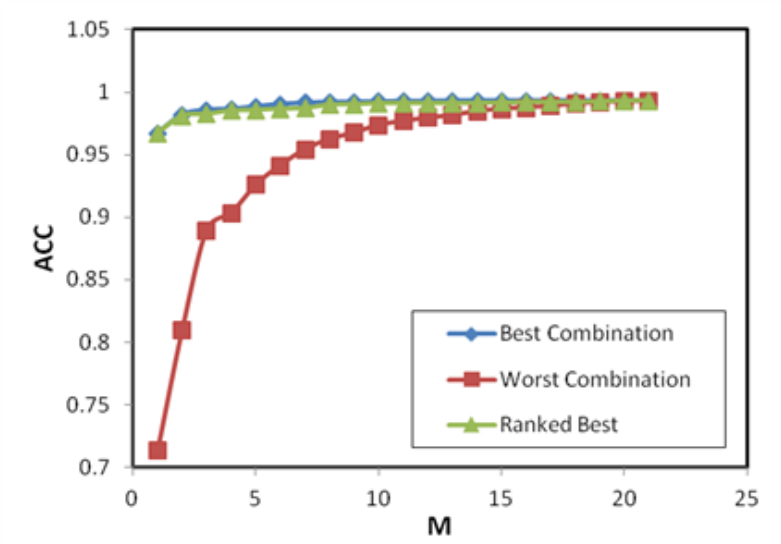


Figure 4.14 Comparison of maximum accuracy by best, worst and ranked best combinations based on real world dataset.

Another important observation from Figure 4.14 is that the accuracy values tend to always increase with the increase of *M*. This is because the average false positive rate for all the scanners is 0.00864 which is lower than 0.01. This also verifies our simulation results from section 4.6.2, where we have seen that for very low average false positive rates; *Q* is almost equal to *N* and the probability of *Q* being lower than *N* is very low.

4.7.4 Runtime Analysis and Comparison of the Models

A comparison of the models in terms of runtime analysis is given in Table 4.7 and as you can see, CPM is far worse than both GAM and CGAM based on this criterion. The main reason behind this is obviously the combinatorial component

in the formula for CPM. We have calculated actual execution time from our experiments for each model as well, which is also listed in Table 4.7. The execution time has been calculated in an Intel Core i3 2.10 GHz laptop for a scenario where N was assigned 20. We see that CPM takes almost more than 6 minutes to execute, whereas GAM and CGAM takes about 1 millisecond. This means, CPM is not the best choice in terms of scalability and the greedy approximation algorithms provide a good alternative. If we want to reduce the execution time even more, we can consider using a subset of M scanners instead of all N scanners, where M < N. If we want to make the best tradeoff between scalability and accuracy, GAM should be our best choice.

Table 4.7 Comparison of the Models

Criteria	*CPM*	*GAM*	*CGAM*
Runtime Complexity	$O(N^2\binom{N}{N/2})$	*O(NlgN)*	*O(NlgN)*
Actual Execution Time	402.55 Seconds	0.00099 Seconds	0.001 Seconds

4.8 Related Work

4.8.1 Multi-scanner Architecture

Very few research papers have been published that focus solely on combining multiple scanners to achieve higher accuracy. Morales et al. [60] investigated

whether a single anti-malware program is sufficient to detect and clean all malware present on a system. They experimentally showed that a single anti-malware program is not sufficient. Their experiments used a combination of 3 well known anti-malware programs in different permutations and they followed a serial architecture. Though in a limited fashion, their results showed that combining multiple anti-malware programs achieve better recall and false negative rates. Oberheide et al. [61] presented a new model for malware detection on end hosts based on providing anti-virus as an in-cloud network service. Their model used multiple, heterogeneous detection engines in parallel, a technique termed as `N-version protection'. They claimed that their approach provides several benefits including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and improved deployability and management. To verify their model, they constructed and deployed an in-cloud antivirus system called CloudAV. CloudAV includes a lightweight, cross-platform host agent and a network service with ten anti-virus engines and two behavioral detection engines. They evaluated the performance, scalability, and efficacy of the system using data from a real-world deployment lasting more than six months and a database of 7220 malware samples covering a one year period. The results showed that CloudAV provides 35% better detection coverage against recent threats compared to a single anti-virus engine and a 98%

detection rate across the full dataset. Cukier et al. [62] presented empirical evidence that detection capabilities are considerably improved by diversity with AVs and their findings also showed that none of the single anti-virus software achieved perfect detection rate.

4.8.2 Collaborative Malware Detection

There has been some research on the collaborative approach in detecting malware. Schmidt et al. [63] presented a collaborative malware detection approach to reduce false negative rate for Android-based malware detection by performing static analysis of executables and sharing detection information among neighboring nodes. Fung et al. [64] presented a collaborative decision making approach for malware detection systems. They proposed a decision model called RevMatch [65], where collaborative malware detection decisions are made based on the scanning history with multiple anti-virus systems. They claimed that the experimental evaluation of their model shows significant improvement over any single anti-virus engine. RAVE [66] is a centralized collaborative malware scanning system for email infrastructures where email correspondence is used to contact multiple agents for malware scanning and a voting mechanism is used to make the final decisions. Marchetti et al. [67] presented a distributed peer-to-peer architecture for collaborative malware and intrusion detection focusing more on dependability and load-balancing issues.

Similar approach was proposed by Colajanni et al. [68]. Lu et al. [69] presented SCMA, a distributed malware analysis system with the goal of better collaboration and scalability.

4.8.3 Multi-AV Scanning Services and Software

There are free online public services that provide scanning reports from multiple anti-virus scanners. VirusTotal [8], a Google subsidiary, is the most prominent among these services. VirusTotal uses the command-line versions of 55 anti-virus scanners (at the time of writing) to scan a single file and include the results returned by each scanner into an aggregated report. In addition to telling whether a given anti-virus solution detected a submitted file, it displays the exact detection label returned by each engine. This service is mainly useful to the anti-virus vendors and to those private users who wants a second opinion. Among other such services, there are Jotti [70], VirSCAN [71], File2Scan [72], and Metadefender [73], where File2Scan and Metadefender are paid services. There are also multi-AV scanning client tools such as HerdProtect [74], HitmanPro [75], SecureAPlus [76], and Multi-AV [77].

4.8.4 Commercial AV Scanners

Most of the anti-virus vendors use their own proprietary malware detection engine which usually includes a signature database, a heuristic-based detection

engine, and a reputation-based detection system. A few of them, namely Emsisoft [78] and G Data [79], use a dual-engine technology where each scan passes through two engines.

4.9 Summary

With the ever increasing amount of activities in the Internet and the world moving into an era of cloud computing, the protection from malicious content remains a top priority of cyber security. And the first step in this protection mechanism is detection of malware and other malicious content. In this chapter, we provided a new set of guidelines in achieving the optimum detection capabilities of malware using multiple anti-virus scanners. We have presented three theoretical models to capture the behavior of a multi-scanning malware detection system based on only the individual detection capabilities or ratings of the member scanners in the system. These models help us in finding the optimum threshold to achieve maximum accuracy in an *N*-scanner system, which our experimental evaluation verifies. Furthermore, we discovered that with high enough false positive rates, addition of new scanners might be disadvantageous and ranking the scanners based on accuracy scores is a good approximation for finding a best subset of scanners. All of these findings along with our models together make up a set of important guidelines for any multi-

scanner detection system consisting of only third-party anti-virus scanners where very little information is available about them, such as VirusTotal.

In future, we anticipate further extending this work into other areas of malicious content detection, such as intrusion detection and anti-spam filtering. Our models do not take into account any specific detail of a single scanner or filter, rather take them as black boxes and only take into account their detection probabilities based on prior detection history. Even the past detection history does not have to be available at hand. Only an approximate or calculated detection rate or quality score is necessary. Therefore, incorporating the intrusion detection or anti-spam filters instead of an anti-virus scanner into a multi-filter system is quite straight forward. The only difficulty here is that there is no existing multi-filter system of intrusion detection or anti-spam filters currently available like VirusTotal or other multi-AV scanning services. We intend to include an extensive experimental evaluation of our models based on popular intrusion detection and anti-spam filters in future.

CHAPTER 5

DETECTION OF HTTP-BASED BOTNET C&C TRAFFIC

5.1 Introduction

In this chapter, we introduce an anomaly detection based approach to detect HTTP-based botnet C&C communication which focuses on how to prevent the botnet from upgrading itself to avoid detection. That means, we want to make it very hard for the botmaster to mimic the legitimate HTTP communication and hide C&C activities. Our approach is based on identifying anomaly in client generated HTTP request packets as well as DNS server generated response packets for the same HTTP communication. Based on some initial analysis of both legitimate and botnet C&C HTTP traffic, we have selected some statistical features that are suitable for detecting anomaly in a large set of captured HTTP traffic. These features are based on patterns emerging from HTTP request packets, more specifically, the URL string that is used to fetch data from an HTTP server. Using these features we primarily run an unsupervised anomaly detection algorithm to distinguish between HTTP request packets generated by human actions and HTTP request packets generated by a software bot, both

legitimate and malicious. Then, to further narrow down the isolated packets, we extract the primary domain names involved in those packets and run a semi-supervised anomaly detection algorithm using a selected set of features based on the DNS server response packets that particularly contain resolved IP address list (A or AAAA record). Eventually, we are left with a list of domain names that are highly probable to be involved in malicious C&C communication.

5.2 Details of Methodology

HTTP botnets try to hide their C&C communication in the massive HTTP traffic generated and transmitted over the Internet everyday by mimicking the behaviors of a legitimate Web communication. Our idea is to find the features that are very hard for the botnets to mimic and use those features to effectively isolate the C&C traffic. Therefore, the first step in our method is to select the feature set. We have selected a feature set based on HTTP request URL field and DNS response packet fields. Then, we apply anomaly detection algorithms on the feature set in unsupervised (for HTTP request URL) and semi-supervised (for DNS response) fashion. There are two stages in the anomaly detection part. In the first stage, our goal is to isolate the software-agent-generated HTTP packets from the browser-generated HTTP packets resulting from human browsing activities. For this purpose, we focus on the HTTP request URL patterns. Here,

the motivation of our approach is that human browsing activities tend to generate diverse and noisy HTTP traffic, whereas the software-agent-generated automated HTTP traffic tends to follow certain algorithms written by the software developer. In other words, browser-generated HTTP traffic can be regarded as human-generated manual traffic, where the human user effectively types or clicks through the URLs; on the other hand, the software-agent-generated HTTP traffic can be regarded as non-human-generated bot-like traffic, where the software agent acts like a bot. Here, we should mention that the browser itself can also act like a software agent or bot and generate bot-like traffic and we have considered this into our approach. In the second stage, the goal is to isolate the botnet C&C domains from the legitimate Web domains. There are two steps in this stage. In the first step, we extract the primary domains from all the IP addresses. The concept of primary domain is discussed later in this section. In the second step, we extract the DNS response features from the dataset for each domain. Then we apply one anomaly detection algorithm (Chebyshev's inequality) to this set, along with our training dataset, in a semi-supervised fashion. Figure 5.1 shows the steps in our method and we describe the details in the next subsections.

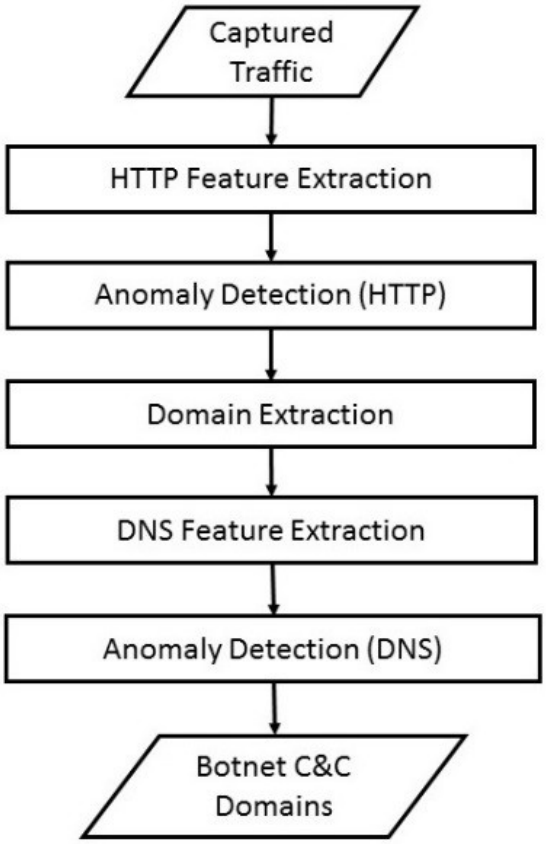


Figure 5.1 The main steps in our detection process.

5.2.1 Feature Selection

1) HTTP Request URL Features

The HTTP request URL features are used to isolate the human-generated manual HTTP traffic and non-human-generated automated HTTP traffic.

a) Total number of distinct URLs

Automated HTTP traffic usually has a lower value for this feature, unless they either generate a distinct URL every time or use many dummy URLs that effectively point to the same set of original URLs. In the latter cases, the value can be too high. Human users usually visit many distinct URL for the same website, which means the value should be high but within a certain limit. Using this feature in our detection method, we can make the botmaster to work a bit harder to mimic normal traffic and hide their activities.

b) Frequencies of request URLs

It is hard to come up with a frequency pattern that mimics human browsing activities. Normally a software agent either will reuse the same URL over and over again or generate a distinct URL every time. We use the mean and the standard deviation values for the set of frequencies as features into the anomaly detection algorithm.

c) Lengths of request URLs

To make it even harder for the botmaster to generate pseudo-browsing pattern that resembles human browsing pattern, we take the request URL lengths into account. A website usually has a hierarchy of web pages with distinct names, which makes all the request URLs different in length. On the other hand,

software-agent-generated URLs generally have the same length, although they can be distinct (for example, if the URLs are encrypted). A botmaster has to randomize not only the URLs, but also the URL lengths to pass this test. We use the standard deviation of all the observed URL lengths as the feature.

d) Order of the request URLs

We take into account the predictiveness of the request URLs by calculating the information entropy of the order of the occurrence of the URLs. We assign to each URL an increasing number starting from 1 and generate a numeric sequence string that denotes the order of occurrence of the URLs. Then, we generate a signed differential number string from the sequence string that shows movement between consecutive URL numbers in the sequence. The following example will better illustrate the process: Suppose we have the numbers 1 through 9 to represent 9 distinct URLs. Then for an example URL sequence string 1231345231, the differential string will be +1+1+1-2+2+1+1-3+1-2. That is, it starts with an initial value of 0 and calculates the difference from the first number in the URL sequence string. Then, it will append the difference between the second number and the first number, append the difference between the third number and the second number, and continue appending until all the numbers are used. To calculate the entropy of this string, we use Shannon's formula, as given by

Equation (5.1), where X is a discrete random variable with possible values $\{x_1, \ldots, x_n\}$ and $H(X)$ is the entropy.

$$H(X) = -\sum_i P(x_i) log_2 P(x_i) \qquad (5.1)$$

2) DNS Response Features

The DNS response features are used to further isolate the legitimate software-agent-targeted domains and botnet C&C domains.

a) Number of distinct IP addresses per response

Botmasters try to evade detection of C&C domains. Therefore, they tend to use IP flux and domain flux techniques. That means, the IP addresses associated with a domain can vary highly as well as there can be many domains for the same C&C server. Although the total number of distinct IP addresses associated with a single domain might be large, the number of IP addresses per DNS response packet can be lower. On the other hand, large load-balancing Web domains tend to have a fixed high number of IP addresses per DNS response packet.

b) Total number of distinct IP addresses

We need the total number of IP addresses to check reuse of IP addresses per domain.

c) Mean TTL (Time to Live) value

The mean TTL value is used to check the frequency of change between IP addresses for a domain.

d) Total number of distinct ASN

Large load-balancing Web domains should have the IP addresses in a more concentrated distribution, whereas IP flux techniques force the botnet domain IP addresses to be sparsely distributed. We distinguish them by calculating the number of Autonomous System Numbers (ASN) for the whole IP address set. A legitimate Web domain should have most of the IP addresses in a single autonomous system, whereas a malicious domain using IP flux techniques should have the IP addresses distributed over many different autonomous systems.

5.2.2 Feature Extraction

We calculate the HTTP request features per source-destination IP address pair, where the source IP address is the client IP address and the destination IP address is the server IP address. We call a single source-destination IP address pair and the corresponding properties and features a Conversation. Since the features are statistical in nature, we need to have at least a minimum number of HTTP request packets per conversation to calculate the true value of each

feature. We set this minimum value to 20. After the first stage is complete, we extract the primary domain name from the hostname field for each conversation and merge them to discard duplicates. The domain name extraction process is discussed in the next subsection. Then, for each domain we find all the DNS response packets and extract the DNS features from them.

5.2.3 Domain Extraction

In this work, a *primary domain name* refers to a domain name with all the subdomains after second or third level domain name stripped. For example, the primary domain name for my.example.com will be example.com, whereas the primary domain name for my.example.co.uk will be example.co.uk. This technique is used in both the steps in the second stage of our method, where we extract primary domain names from each conversation and also from each DNS response packet.

5.2.4 Anomaly Detection Methods

We have two different stages where we need to use anomaly detection. In the first stage for HTTP request features, we use three different anomaly detection methods independently in an unsupervised manner to compare between them. In the second stage for DNS response features, we only use the first anomaly

detection method based on Chebyshev's inequality in a semi-supervised manner. A brief overview of all three techniques is given below.

1) Chebyshev's Inequality

The anomaly detection method based on Chebyshev's inequality can be used in both unsupervised and semi-supervised manner. This technique is particularly suitable when (1) the distribution of the available data is unknown or an experimenter does not want to make assumptions about its distribution, and (2) it is expected that the observations are independent from one another. The formula for Chebyshev's inequality is

$$\Pr(|X - E(X)| \geq k\sigma) \leq \frac{1}{k^2},$$

Where X is a random variable, $E(X)$ is its expected value and $k > 0$ is a parameter. This formula establishes an upper bound for the percentage of data points with value more than k standard deviations away from the population mean. As proposed by Amidan et al. [80], we use a two-stage approach to detect outliers. In the first stage, we use an upper bound of 0.1 (k = 3.16) to find more obvious outliers. Then in the second stage, after discarding the outliers from the first stage, we select a much smaller upper bound of 0.01 (k = 10) to fine tune the detection process. Following their approach, we generate the upper bound and lower bound for the outlier detection value (ODV) for each feature. But we

observe that the lower bound of the ODV values always become negative according to the formula. To maintain better symmetry, we include the inverse values of the existing features into the feature set and calculate ODV values for them as well. An instance is considered outlier when at least one of the feature values falls outside of the ODV bounds.

2) One-class Support Vector Machine

One-class Support Vector Machines (SVM) are a semi-supervised version of traditional Support Vector Machines. We use the extended version of the semi-supervised one-class SVM such that it can be used for unsupervised anomaly detection as proposed by Amer et al. in [81]. Instead of implementing it from scratch, we use the implementation by RapidMiner Studio [82] that follows the same method. It generates an outlier value greater than 1 for outliers in a dataset.

3) Nearest Neighbor based Local Outlier Factor

This anomaly detection algorithm calculates an outlier score based on the local outlier factor (LOF) implementation proposed by Breunig et al. [83].Like the previous one, we use the implementation by RapidMiner Studio [82] for this one as well. Here also a normal instance has an outlier value of approximately 1, while outliers have values greater than 1.

5.2.5 Detection Process

Our detection method is a cumulative process on the captured packets as they are accumulated at a network point such as an ISP router. That means, we don't discard anything as completely benign and the relevant information from all the packets is retained. As illustrated in Figure 5.1, the packets go through the steps into the next stages. If the packet is malicious and belongs to a C&C communication, it should go through all the steps and finally get detected. If the packet is from a benign and legitimate HTTP communication, at some point in the steps it will stop going to the next stage, but still the extracted information will be retained as part of the training set. Note that there is the possibility of false negative in the current round, in which the packet is malicious and belongs to a C&C communication, but does not get detected because of insufficient feature values. We want to point out that in this case the conversation along with its feature values is still retained for future review and not ruled out completely. As we capture more similar packets from the same C&C communication, it will eventually get picked by our detection process. This approach ensures that no C&C communication will be able to bypass our detection scheme completely all the time. The detection might be delayed but eventually the malicious communication will be captured. Note that the storage requirements to retain the conversations are significantly less since we are not storing the entire packet.

5.3 Experimental Evaluation

5.3.1 Implementation

All the processes in the flowchart of Figure 5.1 can be considered as separate modules in our implementation of the overall scheme. We implemented the feature extraction modules in Java using the jNetPcap packet parsing library [84]. The HTTP packet parser is already supported by jNetPcap, but we had to develop our own DNS packet parser on top of the existing support for packet parsing from jNetPcap. The domain extraction module is also part of the same Java code. One of the DNS features involves calculating the number of distinct ASN. We used the Team Cymru IP to ASN lookup [85] service for this purpose. For anomaly detection modules, we implemented three anomaly detection methods, namely, Chebyshev's inequality, one-class support vector machine, and nearest neighbor based local outlier factor algorithms.

Table 5.1 RapidMiner Implementation Configurations

Algorithm	***Description***	
	Parameter	***Value***
1-class SVM	SVM Type Kernel Type Beta Epsilon Auto Gamma Tuning	Eta 1-class RBF 0.3 0.001 True
NN-LOF	K-min K-max Measure Types Mixed Measure	10 20 Mixed Measures Mixed Euclidean Distance

The first one is implemented from scratch by us as part of the same Java code we developed. The other two machine learning algorithms were implemented using RapidMiner Studio [82] by RapidMiner which provides support for many popular machine learning algorithms. Table 5.1 lists the parameters used for each of the algorithm implementations.

5.3.2 Data Collection

The first part of our anomaly detection experiment, namely the anomaly detection in HTTP request traffic, is unsupervised in nature. Therefore, we needed a huge amount of unlabeled real world HTTP traffic for our experimental evaluation. We used a partial dataset from Clemson University campus network traffic [86] that was collected from May to June in 2013. This dataset consisted of general day-to-day Web browsing traffic captured for 7 days and filtered to remove probable malicious traffic from well-known suspicious domains. The total size of the dataset is 271 GB and it contained over 9 million HTTP request packets. The traffic was anonymized and HTTP payloads were truncated for privacy reasons, but we only needed the HTTP request headers. Therefore, this dataset was perfectly suitable for our experiment.

The second part of our anomaly detection, namely the anomaly detection in DNS response traffic, is semi-supervised in nature. That means, we needed a training data set of DNS response packets that will help construct the model

representing normal behavior. Our goal is to distinguish between large load-sharing Web domains and botnet C&C domains. Therefore, we generated the normal DNS response traffic by crawling the top 500 websites in the world according to the Alexa ranking [87]. We crawled for 3 hours every day for a month to remove any kind of bias in the dataset. To reduce the size of the dataset we only retained the DNS traffic. The final dataset contained 97468 DNS response packets.

Table 5.2 Collected HTTP Botnet Families

1. Alina	12. Harnig	23. Weelsof
2. Andromeda	13. Hiloti	24. Winwebsec
3. Beebone	14. Medfos	25. Xpaj
4. Carberp	15. Mirage	26. Xtreme
5. Citadel	16. Njw0r	27. Zegost
6. Cutwail	17. Pushdo	28. ZeroAccess
7. Dofoil	18. Renos	29. ZeroLocker
8. Dorifel	19. Smoke	30. ZeuS
9. Dyre	20. Ubot	31. Zwangi
10. Expiro	21. Umbra	
11. Festi	22. Vobfus	

To effectively evaluate our method, we needed a test dataset of known botnet C&C traffic. We collected binary samples (and already captured C&C traffic in some cases as well) for 31 HTTP based botnet families from various sources. Table 5.2 shows the list of botnet families. The samples were a bit old, but they were still useful since they were still generating the HTTP request packets while being executed even if the C&C servers were already down. In

some cases, the DNS server was responding with NXDOMAIN responses. We removed those packets from our experimental dataset. Therefore, the available test dataset was good enough for our experimental evaluation. There were in total 8258 HTTP request packets and 689 DNS response packets as part of the C&C communication in total.

5.4 Results and Analysis

We evaluated our method by generating the overall false negative and false positive ratios over the complete dataset of benign and malicious domains. After running our experiment, we could accurately count the number of benign and malicious C&C domains involved in the traffic dataset. Tables 5.3 and 5.4 list the results of the experiment.

From Table 5.3, we can see that Chebyshev's inequality performs best as the anomaly detection approach used in terms of false negative ratios. Chebyshev's inequality based approach detects almost 94% of all the malicious C&C domains, whereas the other two methods detect more than 80% of them. Even though it does not look very high in terms of detection rate, it is still very good considering the fact that we are detecting these small number of C&C domains amongst an extremely large number of legitimate domains. From Table 5.4, we can see that the one-class SVM based approach performs best in terms of

false positive ratios, though all three of them have quite small percentage of false positives, considering the huge number of packets and corresponding domains are being scanned.

Table 5.3 Detection Results (False Negative)

Anomaly Detection Approach Used	*Malicious (C&C) Domains*		
	Total	*Detected*	*FN (%)*
Chebyshev's Inequality	134	125	6.71
1-class SVM	134	111	17.16
NN-LOF	134	101	19.40

Table 5.4 Detection Results (False Positive)

Anomaly Detection Approach Used	*Benign Domains*		
	Total	*Detected*	*FP (%)*
Chebyshev's Inequality	7613	338	4.43
1-class SVM	7613	293	3.84
NN-LOF	7613	305	4.04

The main reason behind the slightly larger false negative ratio is that some of the malicious C&C domains did not have sufficient number of communicating packets to get them detected. As we mentioned earlier, our method has the requirement of observing a minimum number of HTTP request packets to calculate the true feature values which will accurately represent the behavior and pattern that we are looking for among the participating legitimate and malicious hosts. After some initial tests, we found that 20 is an appropriate value for this minimum number and we have used it throughout our experiments. We note

that varying this number will result in different false negative and false positive ratios. Finding the optimum threshold requires an extensive evaluation process. We intend it to be a part of our future work.

5.5 Related Work

A significant amount of research work can be found related to HTTP-based botnet detection and botnet detection in general. However, only a small portion of them focus solely on detecting C&C traffic. We can roughly divide them into two main categories: specific HTTP-based botnet detection methods and generic botnet detection methods.

To our best knowledge, there have been only a few existing works focusing solely on HTTP-based botnet detection. Ashley [88] presented an algorithm that uses repeated HTTP connections to detect botnet C&C activity. The algorithm works best if the bot polls the C&C server very frequently. Brezo et al. [89] used several supervised machine learning algorithms to develop a model capable of classifying both botnet and legitimate traffic. Chen et al. [90] combined both Web traffic and domain analysis to detect Web-based botnets with fast-flux domains. Cai et al. [91] focused on HTTP-based botnet's C&C patterns to classify network traffic into clusters. Yamauchi et al. [92] proposed a detection technique for HTTP-based botnets using Support Vector Machines

(SVM). Venkatesh et al. [93] presented a detection method based on hidden semi-Markov model using TCP-based SNMP MIB variables. Matthew et al. [94] proposed a Genetic Algorithm based layered approach to detect attacks conducted by HTTP botnets. Zarras et al. proposed BotHound [95] which uses perceivable minute differences in different implementation of the HTTP protocol to generate models for both malicious and benign requests and thereby classifies HTTP-based malware. We observe that none of the previous researchers have used an anomaly based approach to distinguish legitimate and malicious HTTP communication in order to detect HTTP-based C&C communication.

The field of generic botnet detection is too wide to discuss here in detail. Therefore, we will only briefly overview the detection techniques that are more relevant to our approach, namely, the anomaly or other machine learning based techniques. BotSniffer [96] presented anomaly based detection algorithms based on spatial-temporal correlation and similarity properties of botnet command and control activities. Appendix B of BotSniffer [96] proposed to identify HTTP C&C channels by detecting a repeating and regular visiting pattern from one single bot. BotMiner [97] used a similar approach to cluster network traffic based on similarity. BotHunter [98] used a real-time dialog correlation engine that investigates evidence of botnet life-cycle phases. Lu et al. [99] proposed to detect by clustering botnet traffic based on N-gram feature selection. Wurzinger et al.

[100] presented a system that automatically generates detection models from network traffic traces recorded from actual bot instances. Strayer et al. [101] detected botnets by examining flow characteristics such as bandwidth, duration, and packet timing for evidence of botnet command and control activity. Reiter et al. [102] proposed a method called "TAMD" which aggregates traffic flows of internal hosts of a network to find similar communication patterns to external networks. We see that all of these generic botnet detection techniques also focus solely on different types of C&C patterns and statistical features instead of legitimate communication.

There have been a few attempts to model the Web traffic to identify anomaly and thereby detect malicious traffic. Estevez-Tapiador et al. [103] used Markov chains to model the HTTP network traffic. Their approach detects attacks carried over HTTP and is not meant to detect botnet C&C traffic. Xie et al. [104] used a similar hidden Markov model technique based on inter-arrival time of HTTP requests to detect pseudo Web behavior. Their work focuses on modeling the user session correctly and thereby detects anomaly. Spectrogram [105] presented a model and sensor framework using a mixture of Markov-chains which is able to detect Web-layer code-injection attacks. The difference between these works and our approach is that, ours focuses more on distinguishing between normal legitimate Web traffic and botnet C&C traffic,

rather than detecting general anomalies in the entire Web traffic. Therefore, our technique might fail to detect other types of attacks carried over HTTP, but will be able to identify botnet C&C traffic.

5.6 Summary

In this chapter, we presented an anomaly detection based detection approach for the HTTP-based botnet C&C communication. The main strength of our approach is that it is able to exploit the limitations and weaknesses of a botnet system in our favor to reveal its presence. We believe that this approach will be able to detect not only the present day known botnets but also any future unknown botnet with better capabilities. To verify this, we plan to extend our work to include real time traffic capturing and monitoring in a live network that will include honeypots to attract bot infection.

Another possible extension of our work will be to evaluate our approach using other anomaly detection techniques currently available varying the minimum packet count requirement as mentioned earlier.

CHAPTER 6

CONCLUDING REMARKS

This dissertation provided some new directions towards revealing malicious contents hidden in the Internet. We presented an automated system for collection and analysis of malware hidden inside online advertisements, which can be detected and verified through any online multi-AV scanning services using our proposed multi-scanner model based optimum configurations with maximum accuracy. We presented a game theoretic model of the malvertising inspection problem that can provide guidelines for ad networks to best utilize their resources to mitigate the problem of malvertising. We also presented an anomaly detection based solution approach for the extremely difficult problem of detecting HTTP-based botnet command and control communication.

Through the proposed solutions in this dissertation, we have tackled important problems in four different areas of the malicious content research landscape. We believe that we have been successful in contributing significantly in furthering the progress of research in the field of network security. In future,

we plan to further extend our work by applying our solutions to other related problems of malicious content detection.

BIBLIOGRAPHY

[1] B. Johnson, "Internet companies face up to 'malvertising' threat," The Guardian, published on September 25, 2009, available at http://www.theguardian.com/technology/2009/sep/25/malvertising.

[2] L. Zeltser, "Malvertising: some examples of malicious ad campaigns," available at https://zeltser.com/malvertising-malicious-ad-campaigns.

[3] N. Bilogorskiy, "HuffingtonPost serving malware via AOL ad-Network," published on January 5, 2015, available at http://www.cyphort.com/huffingtonpost-serving-malware.

[4] J.C. Chen, B. Li, "Evolution of exploit kits," Trend Micro white paper, available at https://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-evolution-of-exploit-kits.pdf.

[5] Malvertising, published on April 6, 2016, available at https://www.enisa.europa.eu/publications/info-notes/malvertising.

[6] GeoEdge website, available at http://www.geoedge.com/.

[7] State of Search Marketing Report 2012, available at http://c.ymcdn.com/sites/www.sempo.org/resource/resmgr/members_only/SEMPO_2012_State_Of_Search_M.pdf.

[8] VirusTotal, available at https://www.virustotal.com.

[9] Requests: HTTP for Humans, available at http://docs.python-requests.org/en/latest.

[10] XML and HTML with Python, available at http://lxml.de.

[11] Adblockparser 0.2, available at https://pypi.python.org/pypi/adblockparser/0.2.

[12] EasyList, available at https://easylist.adblockplus.org/en.

[13] Adblock Plus, available at https://adblockplus.org/.

[14] Selenium WebDriver API, available at http://selenium-python.readthedocs.org/en/latest/api.html.

[15] VirusTotal Public API v2.0, available at https://www.virustotal.com/en/documentation/public-api/.

[16] A.K. Sood and R.J. Enbody, "Malvertising – exploiting web advertising," Computer Fraud and Security, Volume 2011, Issue 4, April 2011, Pages 11-16.

[17] S. Mansfield-Devine, "The dark side of advertising," Computer Fraud and Security, Volume 2014, Issue 11, November 2014, Pages 5-8.

[18] T. Zhang, H. Zhang, and F. Gao, "A malicious advertising detection scheme based on the depth of URL strategy," in proceedings of Sixth International Symposium on Computational Intelligence and Design (ISCID), October 2013.

[19] Google's anti-malvertising website, available at http://www.anti-malvertising.com.

[20] Y. M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated web patrol with strider honeymonkeys," in proceedings of the Network and Distributed System Security Symposium (pp. 35-49), February 2006.

[21] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou, "Collecting autonomous spreading malware using high-interaction honeypots," in Information and Communications Security (pp. 438-451), Springer Berlin Heidelberg, 2007.

[22] J. Nazario, "PhoneyC: a virtual client honeypot," in proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more, pp. 6-6, 2009.

[23] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: De-cloaking internet malware," in IEEE Symposium on Security and Privacy (SP), pp. 443-457, 2012.

[24] K.Z. Chen, G. Gu, J. Zhuge, J. Nazario, and X. Han, "WebPatrol: Automated collection and replay of web-based malware scenarios," in

proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp. 186-195, 2011.

[25] Y-D. Lin, C-Y. Lee, Y-S. Wu, P-H. Ho, F-Y. Wang, and Y-L. Tsai, "Active versus passive malware collection," Computer 47, no. 4 (2014): 59-65.

[26] C.H. Tseng, S. Wang, S-C. Wang, and T-Y. Juang, "Proactive malware collection and classification system: How to collect and classify useful malware samples?," in International Conference on Information Science, Electronics and Electrical Engineering (ISEEE), vol. 3, pp. 1846-1849, 2014.

[27] Annual Security Report, Cisco, 2016.

[28] Hacking communities in the Deep Web, Infosec Institute, published on May 15, 2015, available at http://resources.infosecinstitute.com/ hacking-communities-in-the-deep-web/.

[29] J. Segura, and E. Aseev, "Operation Fingerprint: A look into several Angler exploit kit malvertising campaigns," available at https://blog.malwarebytes.com/threat-analysis/2016/03/ofp.

[30] "Just-in-time malware assembly: advanced evasion techniques," white paper by Invincea, available at https://www.invincea.com/2015/07/ white-paper-just-in-time-malware-assembly-advanced-evasion-techniques/.

[31] Massive AdGholas malvertising campaigns use Steganography and file whitelisting to hide in plain sight, available at https://www.proofpoint.com/uk/threat-insight/post/massive-adgholas-malvertising-campaigns-use-steganography-and-file-whitelisting-to-hide-in-plain-sight.

[32] A. Zaharia, "The ultimate guide to Angler exploit kit for non-technical people," published on May 18, 2016, available at https://heimdalsecurity.com/blog/ultimate-guide-angler-exploit-kit-non-technical-people/.

[33] M. N. Sakib, and C.-T. Huang, "Automated collection and analysis of malware disseminated via online advertising," in IEEE Trustcom/BigDataSE/ISPA 2015(Vol. 1, pp. 1411-1416), August 2015.

[34] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iframes point to us," in USENIX Security Symposium, 2008.

[35] 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, 2012.

[36] 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, 2013.

[37] T. Taylor, X. Hu, T. Wang, J. Jang, M.P. Stoecklin, F. Monrose, and R. Sailer, "Detecting malicious exploit kits using tree-based similarity searches," in proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (pp. 255-266), March 2016.

[38] B. Stock, B. Livshits, and B. Zorn, "Kizzle: A signature compiler for exploit kits," Technical Report MSR-TR-2015-12, Microsoft Research, February 2015.

[39] G. Wang, J.W. Stokes, C. Herley, and D. Felstead, "Detecting malicious landing pages in Malware Distribution Networks," in 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2013.

[40] G. Lu, and S. Debray, "Automatic simplification of obfuscated JavaScript code: A semantics-based approach," in IEEE Sixth International Conference on Software Security and Reliability (SERE) (pp. 31-40), June 2012.

[41] W. Xu, F. Zhang, and S. Zhu, "JStill: Mostly static detection of obfuscated malicious JavaScript code," in proceedings of the third ACM conference on Data and application security and privacy (pp. 117-128), February 2013.

[42] X. Dong, M. Tran, Z. Liang, and X. Jiang, "AdSentry: Comprehensive and flexible confinement of JavaScript-based advertisements," in proceedings of the 27th Annual Computer Security Applications Conference (pp. 297-306), December 2011.

[43] A. Dewald, T. Holz, and F.C. Freiling, "ADSandbox: Sandboxing JavaScript to fight malicious websites," in proceedings of the 2010 ACM Symposium on Applied Computing (pp. 1859-1864), March 2010.

[44] H. Mekky, R. Torres, Z.-L. Zhang, S. Saha, and A. Nucci, "Detecting malicious http redirections using trees of user browsing activity," in IEEE Conference on Computer Communications, 2014.

[45] S. Ford, M. Cova, C. Kruegel, and G. Vigna, "Analyzing and detecting malicious Flash advertisements," in ACSAC (pp. 363-372), December 2009.

[46] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: understanding and detecting malicious web advertising," in ACM Conference on Computer and Communications Security, 2012.

[47] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. Riley, "Are these ads safe: Detecting hidden attacks through the mobile app-Web interfaces," 2016.

[48] S. Arshad, A. Kharraz, and W. Robertson, "Include me out: In-browser detection of malicious third-party content inclusions," in proceedings of International Conference on Financial Cryptography, 2016.

[49] L.Y. Njilla, N. Pissinou, and K. Makki, "Game theoretic modeling of security and trust relationship in cyberspace," International Journal of Communication Systems, 29(9), pp.1500-1512, 2016.

[50] C.A. Kamhoua, M. Rodriguez, and K.A. Kwiat, "Testing for hardware trojans: A game-theoretic approach," in International Conference on Decision and Game Theory for Security (pp. 360-369), Springer International Publishing, November 2014.

[51] A. Bensoussan, M. Kantarcioglu, and S.C. Hoe, "A game-theoretical approach for finding optimal strategies in a botnet defense model," in International Conference on Decision and Game Theory for Security (pp. 135-148), Springer Berlin Heidelberg, November 2010.

[52] M.H.R. Khouzani, S. Sarkar and E. Altman, "A dynamic game solution to malware attack," in proceedings of IEEE INFOCOM (pp. 2138-2146), April 2011.

[53] B. Soper, and J. Musacchio, "A botnet detection game," in IEEE 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton), September 2014.

[54] G. Gianini, M. Cremonini, A. Rainini, G.L. Cota, and L.G. Fossi, "A game theoretic approach to vulnerability patching," in IEEE International Conference on Information and Communication Technology Research (ICTRC) (pp. 88-91), May 2015.

[55] Accuracy, https://en.wikipedia.org/wiki/Accuracy_and_precision.

[56] F1 Score, https://en.wikipedia.org/wiki/F1_score.

[57] Matthews Correlation Coefficient, https://en.wikipedia.org/wiki/Matthews_correlation_coefficient.

[58] VirusSign, http://www.virussign.com.

[59] SourceForge, http://www.sourceforge.net.

[60] J.A. Morales, S. Xu, and R. Sandhu, "Analyzing malware detection efficiency with multiple anti-malware programs," in proceedings of ASE/IEEE International Conference on BioMedical Computing, December 2012.

[61] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-version antivirus in the network cloud," in proceedings of 17th USENIX Security Symposium, 2008.

[62] M. Cukier, I. Gashi, B. Sobesto, and V. Stankovic, "Does malware detection improve with diverse antivirus products? An empirical study," in proceedings of 32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP), 2013.

[63] A-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K.A. Yuksel, S.A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on Android," in proceedings of IEEE International Conference on Communications (ICC'09), 2009.

[64] C.J. Fung, D.Y. Lam, and R. Boutaba, "A decision making model for collaborative malware detection networks," Technical Report: CS-2013-01, School of Computer Science, University of Waterloo, Canada, 2013.

[65] C.J. Fung, D.Y. Lam, and R. Boutaba, "Revmatch: A decision model for collaborative malware detection," Technical Report CS-2013-01, Department of Computer Science, University of Waterloo, 2013.

[66] C. Silva, P. Sousa, and P. Verissimo, "Rave: Replicated antivirus engine," in proceedings of IEEE International Conference on Dependable Systems and Networks Workshops (DSN-W), pages 170–175., 2010.

[67] M. Marchetti, M. Messori, and M. Colajanni, "Peer-to-peer architecture for collaborative intrusion and malware detection on a large scale," Information Security, pages 475–490, 2009.

[68] M. Colajanni, D. Gozzi, and M. Marchetti, "Collaborative architecture for malware detection and analysis," in proceedings of the 23rd International Information Security Conference, The International Federation for Information Processing (IFIP),Volume 278,pp 79-93, 2008.

[69] H. Lu, X. Wang, and J. Su, "SCMA: Scalable and collaborative malware analysis using system call sequences," International Journal of Grid & Distributed Computing, 2013.

[70] Jotti, http://virusscan.jotti.org.

[71] VirSCAN, http://www.virscan.org.

[72] File2Scan, http://www.file2scan.net.

[73] Metadefender, https://www.metadefender.com.

[74] HerdProtect, http://www.herdprotect.com.

[75] HitmanPro, http://www.surfright.nl/en/hitmanpro.

[76] SecureAPlus, http://www.secureaplus.com/Main/index.php.

[77] Multi-AV, http://multi-av.thespykiller.co.uk.

[78] Emsisoft, http://www.emsisoft.com.

[79] G Data, https://www.gdata-software.com.

[80] B. Amidan, T. Ferryman, and S. Cooley, "Data outlier detection using the Chebyshev theorem," in IEEE Aerospace Conference, 2005.

[81] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in

proceedings of ACM SIGKDD Workshop on Outlier Detection and Description, 2013.

[82] RapidMiner Studio, available at https://rapidminer.com/products/studio.

[83] M. Breunig, H-P. Kriegel, R. Ng, and J.Sander, "LOF: Identifying density-based local outliers," ACM sigmod record. Vol. 29. No. 2, 2000.

[84] jNetPcap, available at http://jnetpcap.com.

[85] Team Cymru IP to ASN Lookup v1.0, available at https://asn.cymru.com/.

[86] I. Özçelik, and R. Brooks, "Deceiving entropy based DoS detection," Computers & Security 48 (2015): 234-245.

[87] Alexa top sites, available at http://www.alexa.com/topsites.

[88] D. Ashley, "An algorithm for HTTP bot detection," University of Texas at Austin – Information Security Office, January 2011.

[89] F. Brezo et al., "A supervised classification approach for detecting packets originated in a HTTP-based botnet," CLEI Electronic Journal, Volume 16, Number 03, Paper 02, December 2013.

[90] C. Chen, M. Huang, and Y. Ou, "Detecting Web-based botnets with fast-flux domains", in Advances in Intelligent Systems and Applications, Volume 2, Springer, 2013, pp 79-89.

[91] T. Cai, and F. Zou, "Detecting HTTP botnet with clustering network traffic," in proceedings of 8th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2012.

[92] K. Yamauchi, Y. Hori, and K. Sakurai, "Detecting HTTP-based botnet based on characteristic of the c&c session using by SVM," in proceedings of 8th Joint Conference on Information Security, 2013.

[93] G. Venkatesh, V. Srihari, R. Veeramani, R. Karthikeyan, and R. Anitha, "HTTP botnet detection using hidden semi–Markov model with SNMP MIB variables," in International Journal of Electronic Security and Digital Forensics, Volume 5, Number 3–4/2013, January 2014.

[94] S. Mathew, A. Ali, and J. Stephen, "Genetic algorithm based layered detection and defense of HTTP botnet," in ACEEE International Journal on Network Security , Vol. 5, No. 1, January 2014.

[95] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of HTTP-based malware," In IEEE 12th Annual International Conference on Privacy, Security and Trust (PST), 2014.

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

[97] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in proceedings of the 17th Conference on Security Symposium, USENIX Association, Berkeley, CA, USA, 2008.

[98] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting malware infection through IDS-driven dialog correlation," in proceedings of 16th USENIX Security Symposium on USENIX Security Symposium. USENIX Association: California, 2007.

[99] W. Lu, G. Rammidi, and A. Ghorbani, "Clustering botnet communication traffic based on N-gram feature selection," Journal of Computer Communications, Volume 34, Issue 3, March 2011, Pages 502–514.

[100] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda "Automatically generating models for botnet detection," 14th European Symposium on Research in Computer Security (ESORICS'09), 2009.

[101] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control," in proceedings of the 31st IEEE Conference on Local Computer Networks, 2006.

[102] M. Reiter, and T. Yen, "Traffic aggregation for malware detection," in Detection of Intrusions and Malware, and Vulnerability Assessment, vol. 5137/2008, Lecture Notes in Computer Science. Springer: Berlin, 2008, pp 207–227.

[103] J. Estevez-Tapiador, P. Garcia-Teodoro, and J. Diaz-Verdejo, "Measuring normality in HTTP traffic for anomaly-based intrusion detection," in

Journal of Computer Networks, Volume 45, Issue 2, June 2004, Pages 175–193.

[104] Y. Xie, S. Tang, X. Huang, and C. Tang, "Modeling Web session for detecting pseudo-HTTP traffic," in Journal of Computer, Vol. 8 No. 2, 2013.

[105] Y. Song, A. Keromytis, and S. Stolfo, "Spectrogram: A mixture-of-Markov-chains model for anomaly detection in Web traffic," in proceedings of Network and Distributed System Security Symposium, February 2009.