DISCOVERING ZERO-DAY ATTACKS BY LEVERAGING CYBER THREAT INTELLIGENCE

by

Amirreza Niakanlahiji

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2019

Approved by:

Dr. Bei-Tseng Chu

Dr. Weichao Wang

Dr. Jinpeng Wei

Dr. Sheng-Guo Wang

ABSTRACT

AMIRREZA NIAKANLAHIJI. Discovering Zero-day Attacks By Leveraging Cyber Threat Intelligence. (Under the direction of DR. BEI-TSENG CHU)

Cyber attacks cost companies and organizations billions of dollars each year. To alleviate this issue, security professionals and companies continuously attempt to discover new vulnerabilities, and also to share their cyber threat information about ongoing attacks with other defenders and the public. However, the amount of data that is being shared by them is immense; making the problem of finding useful information tantamount to looking for a needle in a haystack.

In this dissertation, I present a new framework utilizing various machine learning, text mining and natural language processing techniques to automatically extract cyber threat intelligence, in special Indicators of compromise, from public data sharing platforms such as social media, discussion forums, and text sharing websites. I also present two systems to predict malicious IP addresses and to detect phishing URLs. These systems can be integrated with the presented framework and consume its output to improve their results. Moreover, I introduce a new class of vulnerabilities that arises from conflicting requirements in modern operating systems. To show its feasibility, I reveal one of such vulnerabilities in Microsoft Windows operating systems and based on that propose a new stealthy lateral movement that cannot be detected by existing state-of-the-art detection systems.

To extract useful threat information from public data sharing platforms, I, first, present a reputation model to identify credible cyber threat intelligence sources. Only streams of data published by such sources are tracked. Although the identified sources publish threat information in general, they may also post about other topics such as personal matters. Hence, I devise another model to filter-out non-threat information from the observed data streams. Next, I introduce an IoC extraction tool to extract

and combine IoCs from the filtered streams. The output of this framework in given to two predictive models to validate the IP addresses and URLs associated with the resulted IoCs. The confirmed IoCs can further be used to train these system. In this dissertation, I focus on Twitter and Pastebin as exemplars of social media and text-sharing platforms respectively. However, the presented work can be adapted to other similar platforms without requiring significant effort.

ACKNOWLEDGEMENTS

I am thankful to my advisor, Prof. Bei-Tseng Chu, for his professional and personal mentorship. I am also grateful to my doctoral committee members, Prof. Weichao Wang, Prof. Jinpeng Wei, and Prof. Sheng-Guo Wang, for their feedback and contribution. I also would like to thank Cybersecurity Analytics and Automation (CCAA) for supporting part of this dissertation. Moreover, I would like to thank my beloved wife, Lida, my parents, and my wonderful sisters for their love during these stressful years. I would also like to thanks Mir Mehedi Pritom, Md Rabbi Alam, and Reginald Harper for their valuable contribution in my work. And last, but not least, I would like to thank my beloved friends at UNC Charlotte, especially Haadi Jafarian, Hossein Hemati, Abdullah Farooq, Ghaith Husari, Abhinav Mohanti without whom this long journey would have been even harder.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API An acronym for Application Programming Interface.

APWG An acronym for Anti-Phishing Working Group.

AV An acronym for antivirus.

C&C An acronym for Command and Control.

CDN An acronym for Content Delivery Network.

CSRF An acronym for Cross-Site Request Forgery.

CTI An acronym for Cyber Threat Intelligence.

CTPH An acronym for Context Triggered Piecewise Hash.

DOM An acronym for Document Object Model.

EV An acronym for Extended Validation.

IANA An acronym for Internet Assigned Numbers Authority.

IoC An acronym for Indicator of Compromise.

LOC An acronym for Line of Code.

PSL An acronym for Public Suffix List.

RTT An acronym for Round Trip Time.

TLD An acronym for top level domain.

XSS An acronym for Cross-site Scripting.

CHAPTER 1: Introduction

1.1 Motivation

In recent years, cyber attacks have consistently grown in terms of volume, sophistication, coordination, and pervasiveness. Such attacks impose billions of dollars loss to companies and government entities annually. Protecting systems against cyber attackers is a cat and mouse game: on the one hand, defenders deploy a set of defense mechanisms to enforce their security policies and prevent attackers from penetrating their systems, and on the other hand, attackers attempt to find ways to break or bypass these defense mechanisms to take advantage of the target systems. Upon learning new attacks, defenders introduce new defense mechanisms to protect themselves and this loop repeats endlessly.

However, in general, attackers have a key advantage that makes their attack much cheaper and more successful than defense plans. Attackers have the opportunity to study their targets and crafting attack plans accordingly, while defenders must fortify their systems against all possible attacks without truly knowing about the actual attackers or their plans. This information asymmetry, or knowledge gap, between attackers and defenders makes the defending task significantly more resource demanding and highly susceptible to failure.

Broadly speaking, defenders have two different approaches to take in order to defend against unknown attackers. In the first approach, which is traditional, defenders attempt to identify as many as unknown vulnerabilities in their environments and preemptively isolate them in addition to patching all known vulnerabilities. In the second approach, defenders share cyber threat intelligence attacks targeting their

environments with other defenders. The shared CTI information is then analyzed by defenders to learn about potential attackers and their ways of operation.

Sharing cyber threat intelligence about ongoing attacks can significantly alleviate information asymmetry between defenders and attackers as many cyber attackers tend to reuse or share the network infrastructure, techniques, tactics, and procedures across multiple attacks. As a result, sharing Indicators of Compromise (IoCs), such as malicious IP addresses, malware hashes, and malicious URLs, as well as new attack method is a key part of a modern cyber defense strategy. For example, most enterprises check an IP blacklist at their network perimeter to identify potentially malicious traffic. Such traffic is often blocked and, depending on their policy, additional actions may be taken. For instance, a host sending information to blacklisted IPs may be investigated for zero-day infection.

The amount of cyber threat information that is being shared on the Internet is immense. Countless security professionals and security companies devote their time and effort on hunting cyber threats and sharing such valuable information to the public in hope to halt the criminal activities as soon as possible; thus protecting their users and the public. However, as there is no widely adopted standard way of sharing information such an effort cannot reach its full potential as it may not reach to the audience within a reasonable time. Most of threat intelligence data are not published in a well-structured format, nor through a well-defined API. Commonly such information is shared through social media, discussion forums, text sharing platforms, and mailing lists. Such published information often need to be interpreted, correlated with other information in order to derive actionable threat intelligence.

In this dissertation, I present a set of new frameworks and tools utilizing various machine learning, text mining and natural language processing (NLP) techniques to locate and ingest cyber threat intelligence, in particular indicators of compromise, from the wealth of data available on public data sources such as social media. In-

dicators of Compromise (IoCs) are defined as network or systems artifacts such as IP addresses, domain name, and file hash that are observed during cyber attacks. Such information is valuable because they lead to immediate actions (*e.g.*, to block traffic at network perimeter, and to initiate take down actions). Moreover, I introduce a new class of vulnerabilities in existing operating systems that can be exploited by attackers. As a showcase, I present a new stealthy lateral movement based on a vulnerability in this class.

1.2 Background

In this section, I define the terms and concepts that are frequently referred to in this dissertation and are essential for understanding the contribution of this work.

1.2.1 Cyber Threat Intelligence

In this work, one of my primary goals is to extract cyber threat intelligence from publicly available data sources including social media and cyber threat repositories. Rob McMillan [3], a Gartner analyst, defined threat intelligence as "evidence-based knowledge, including context, mechanisms, indicators, implications and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard". This definition can be used to define cyber threat intelligence (CTI) as threat intelligence related to computers, networks, and in general information technology (IT) systems.

1.2.2 Indicator of Compromise (IoC)

Indicators of Compromises(IoCs) are network or system artifacts that are observed during a cyber attack. IoCs can be categorized in different ways; a common way to do so is based on the granularity of data represented by IoCs [4]. In this categorization, IoCs are divided into three groups: atomic, computed, and behavioral IoCs. Atomic

IoCs, such as ip addresses, domain names, registry keys, and process names, represent network or system artifacts being observed during a cyber attack. Computed IoCs are the ones that are calculated from data observed during the attack such as hash values of malware instances. The behavioral IoCs are the ones that are a combination of the other IoCs such as a malicious docx file X with a hash value of Y is hosted on server Z, upon opening the docx, a malware W will be executed on the victim machine.

1.2.3 Advanced Persistent Attackers

Advanced Persistent Threats (APTs) are organized, well-supported, and well-planned cyber attacks against governments and companies with high values [5]. APT attackers use multiple attack techniques and tactics conducted meticulously to avoid detection, so that they can maintain their access to the target for a long time. They also amend their techniques and tactics over time to cope with changes on the target networks and to further extend their footholds [6]. Such attackers cost companies and government agencies billions of dollars in financial losses annually. An exemplar of such groups is Lazarus, also known as APT38. Since 2014, this group has attempted to steal over 1.1 billion dollars from financial institutions worldwide, including the recent 81-million-dollar heist of Bangladesh's central bank [7].

1.2.4 Cyber Kill Chain

A kill chain, a military term, is a step-by-step process to identify, prepare to attack, engage, and destroy a target [8]. This concept is adapted by Lockheed Martin, called the cyber kill chain, to describe the steps that advanced persistent threat (APT) attackers take to attack their targets. In the cyber kill chain, an APT attacker must perform reconnaissance to identify the target, and develop suitable payloads to compromise and bypass a trusted perimeter. Once inside the attacker would take

actions toward the objective by laterally moving inside the environment. At every new location, the attacker may repeat this process to identify new potential targets, compromise them, and expand her intrusion inside the environment [9]. To be specific, the cyber kill chain proposed by Lockheed Martin is comprised of the following seven steps: reconnaissance, weaponization, delivery, exploitation, installation, command and control (C2), and actions on objective.

- Reconnaissance: is an observation phase in which attackers assess the environment to discover targets and select tactics for launching the attack. They can use a variety of techniques, ranging from actively scanning network to social engineering exploits.

- Weaponization: crafting an attack such as creating a malicious document paired with a phishing page or a self-replicating malware

- Delivery: sending the weaponized payload to the target environment, through network, email attachments, or USB flash drive.

- Exploitation: executing the delivered attack code to exploit a vulnerability in the system. The execution may be triggered automatically via a vulnerability or manually by luring the user to do so.

- Installation: installing a backdoor on the exploited target in order to maintain a foothold inside the environment.

- Command and Control (C2): establishing outbound connections for command and control to the adversary. This allows the adversary to have a remote presence inside the target environment, turning them into an insider.

- Actions on Objectives: after achieving an insider access, the attacker can execute actions to achieve their objectives. This objective could be data exfiltration, or sabotaging the integrity and availability of the target. Alternatively,

the attacker may decide to use the new compromised target as a foothold for compromising additional systems; moving laterally inside the target network.

1.3 Aim and Objectives

The aim of this dissertation is two-fold. First, to provide cyber defenders with new frameworks and tools to fully utilize available cyber threat intelligence about ongoing attacks. Second, to report a new class of vulnerabilities in conventional operating systems that can be exploited by attackers.

My first goal is to help defenders to locate and process public cyber threat intelligence reports describing recent attacks in order to gain knowledge about techniques, tactics, and procedures in addition to network resources used in these attacks. Obtaining such knowledge about the potential attackers can help defenders to significantly reverse the existing knowledge gap between defenders and attackers. Defenders can synthesis such knowledge to enhance the configuration of their defensive mechanisms to prevent zero-day, unknown, attacks destined to their networks. My next goal is to help operating system designers improve the implementation of isolation mechanisms in modern operating systems. Isolation is a fundamental security design principle for enforcing application security.

The main objective in this dissertation is to devise a set of new tools that enable defenders to, first, efficiently discover reliable sources of information publishing cyber threat information about ongoing cyber attacks through public data sharing platforms. Then, to monitor such sources to extract useful information for predicting network infrastructures, and attack techniques, tactics, and procedures that presumably will be used by attackers to target new victims. The secondary objective is to develop systems that can ingest discovered cyber threat intelligence in order to dismantle unknown attacks such as blocking zero-day phishing web page or unknown malware. Another key objective in this dissertation is to reveal a fundamental design

flaw in Linux and Windows operating systems that enable attackers to misuse resources such as sockets on these platforms. Based on this design flaw, a new stealthy lateral movement is introduced which cannot be detected by existing state-of-the-art detection systems.

1.4 Contribution

My main contribution in this dissertation is as follows:

- I propose a novel framework that enables defenders to accurately and efficiently extract IoCs from cyber threat intelligence reports published through public data sharing platforms such as social media, discussion forums, and text-sharing portals. This framework can help them to retrieve fresh IoCs about ongoing attacks that are published by cybersecurity professionals.

- I offer a novel approach to predicate malicious zero-day IP addresses that no malicious activity have been publicly reported by considering the network activity of malware instances. Through evaluation, I show that it can accurately predict such IP addresses while having a small false positive.

- I develop a feature-rich machine learning framework for detecting zero-hour phishing web pages in real time. I offer a set of salient features that can accurately cut-off phishing web pages from the benign ones. These features can be computed efficiently and independently from external systems, which makes it a scalable approach.

- I also present a new natural language processing-based framework to analyze cyber threat reports written in unstructured text in order to identify techniques, tactics, and procedures (TTPs) used by attackers. I use this framework to identify trending TTPs used by APT attackers.

- Last but not least, I explain a design flaw in modern operating systems that leads to a new class of vulnerabilities. To demonstrate its feasibility, a novel stealthy lateral movement strategy is introduced that APT attackers can employ to laterally move between system while evading existing state-of-the-art detection mechanisms.

1.5 Dissertation Organization

The rest of the dissertation is organized as follows:

Chapter 2 presents a novel approach to proactively identify zero-day malicious IP addresses based on the network activities of recently identified malware instances. The underlying idea in this approach is to identify soft targets, the systems that can be acquired by attackers with minimal effort, in the vicinity of known malicious IP addresses. Such targets with high probability have already acquired or soon will be overtaken by malicious actors.

Chapter 3 presents a new scalable feature-rich machine learning framework to identify zero-hour phishing attacks. To do so, I introduce a set of new salient features that can collectively be used to distinguish between phishing and benign web pages with a high level of accuracy.

Chapter 4 presents a new Natural Language Processing (NLP)-based framework to analyze threat reports written in unstructured text regarding cyber attacks to identify TTPs used by attackers. Such information can be further analyzed to prioritize these TTPs, which can help defenders in better budgeting the resources to improve cyber defense. I use this framework to study a large set of APT reports to understand the current trends in attack techniques.

Chapter 5 presents a new framework that help defenders to locate cyber threat intelligence sources in public data-sharing platforms such as social media, discussion forums, and text-sharing websites. It also demonstrates how the approaches

introduced in previous chapters can be combined together to fully take advantage of information shared by such resources.

In chapter 6, a novel lateral movement scenario based on a design flaw in conventional operating systems is presented. This lateral movement strategy enables attackers to deep penetrate a network without being discovered by existing state-of-the-art defensive mechanism.

I conclude this dissertation, in chapter 7, by giving a concise overview of my presented ideas and their results. In this chapter, I also suggests a few directions on possible future research based on my presented work.

CHAPTER 2: Predicting Zero-day Malicious IP Addresses

2.1 Introduction

Sharing Indicators of Compromise (IoCs), such as malicious IP addresses, malware hashes, and malicious URLs, is a key part of a modern cyber defense strategy. For example, most enterprises check an IP blacklist at the network perimeter to identify potentially malicious traffic. An obvious limitation with blacklists is that they only offer a rear mirror view of the threat landscape. Attackers can easily bypass an IP blacklist by using new IP addresses that have not been employed in malicious activities. Previous research works have attempted to predict IoCs that may be associated with new malicious activities, e.g., short DNS record TTL [10], a recently registered domain [11], and a misspelled domain name that are atypical to normal businesses [12]. In addition, others have shown that infrastructures that are used by attackers to launch their malicious activities tend to cluster in certain "neighborhood" (e.g. same hosting network) [13, 14].

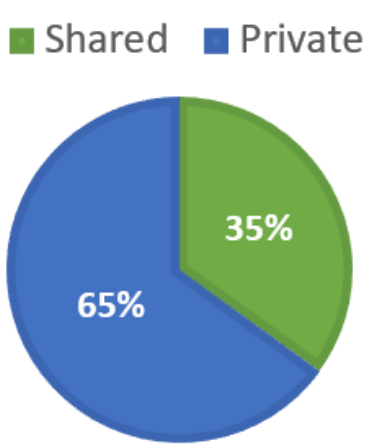
This chapter presents an approach to predict IP addresses that are likely to be used for malicious activities based on Cyber Threat Intelligence (CTI) data sources. I start with the observation that attackers tend to find *soft targets* on the Internet to deploy the infrastructures such as drive by download, command and control (C&C), and web hosting, necessary for conducting their operations. All such infrastructure services require a public IP address, hence by blocking the IP addresses of *soft targets*, one can preemptively disrupt the attackers' operations without knowing about their attack plans. *Soft targets*, in this chapter, refers the systems that are least *costly* for attackers. *Cost* is defined here more broadly to include both low price for purchasing

hosting service or lax security measures in the following areas (1) low cost of exploiting existing resources such as a web server, or hijacking a domain name (2) low cost of renting new resources such as registering a domain name, (3) low risk of attrition due to lax verification of credentials, and (4) low risk of prosecution.

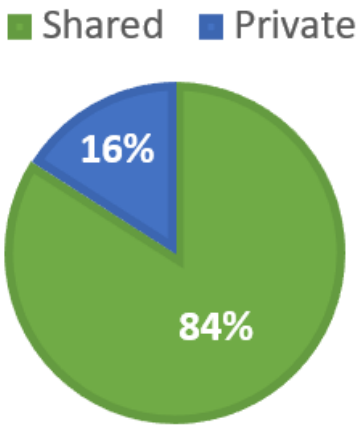
Based on my observation, I hypothesize that shared hosting services, where the services are shared by multiple independent entities, on the Internet are good candidates for soft targets and hence the probability of observing a shared hosting service involved in a malicious activity is considerably higher than a private hosting service. This hypothesis is based on the following reasons. First preliminary investigation suggests that the cost for shared hosting is significantly lower than private hosting. For example, I found a service provider advertising $99 hosting for life. Second, low-cost hosting providers typically offer little security service. Websites using such hosts may be easy targets for attackers so that attackers can acquire the IP address for *free*.

By resolving IP addresses for all .com and .net top level domains, I found that 35% of .com and .net domains run on shared hosts, where the same IP address is used for multiple unrelated domains. This is often provided by a hosting provider that adopts shared hosting as a business strategy to reduce cost. The rest, 65%, of .com and .net domains use private hosting services, where the IP address for the domain is not shared as shown in Figure 2.1a. In contrast, 84% of outbound malware traffic refers to shared hosts as shown in Figure 2.1b. For malware analysis, I used the GT Malware Passive DNS Daily Feed dataset (GT Malware dataset for short). GT Malware publishes a daily feed of DNS requests with about 250,000 malware instances.

Obviously, not all websites using shared hosting are malicious, and some shared hosting service providers offer effective security service. Another observation is that responsible businesses will choose online service providers with better security services. The networks operated by such providers exhibits less malicious activities.



(a) Among all service providers



(b) Among malicious service providers

Figure 2.1: Percentage of shared vs private hosting providers among (a) all service providers (b) malicious service providers on the Internet

In this chapter, I present a new approach that is based on the previous two observations. The rest of the chapter provides empirical evidence to support this method of identifying zero-day malicious IP addresses. I show that the presented approach can detect 88% of zero-day malwares missed by the following AV software: Kaspersky, McAfee, AVG, Avast, and Symantec. It can also block 68% of phishing URLs before they are reported by Phishtank. I will also provide substantial evidence that this approach will not considerably impact the normal business needs of an enterprise.

2.2 Predicting Zero-day malicious IP Addresses

The first task is to identify IP addresses that are engaged in the shared hosting behavior. Verisign's top level domain (TLD) zone files for .com and .net are used to identify shared hosting providers. On a daily basis, Verisign TLD files are fetched and IP addresses for .com and .net domains are resolved to identify IP addresses serving multiple domains owned by different organizations. According to [15], about 47 percent of all registered domains use .com as TLD. Therefore, this mapping is a good sample representation of all domain name to IP address mappings on the

Internet.

Large enterprises often own multiple domains and point them to the same server. For example, both t.com and twitter.com are owned by Twitter. Domains that belong to a single organization must be treated as aliases. I use WHOIS registrant organization names for top domains on the Internet to identify such aliases.

The second task is gathering IP addresses associated with malicious activities from cyber threat intelligence data sources. In the prototype implementation, data is collected from GT Malware and Phishtank. GT Malware dataset contains DNS names requested by malware instances and the corresponding IP addresses. It contains approximately 250,000 new malware instances every day. Each malware instance is identified by its hash. On average there are 5,000 unique IP addresses reported as being associated with reported malware instances each day. I refer to this list of IP addresses as GT-IP-List. Phishtank is a community based website for sharing and validating phishing URLs. Users submit phishing URLs and other users check the URLs and vote to determine whether a URL is a valid phishing URL. Every day about 2,400 URLs are submitted by users on Phishtank.

Researchers have shown that an unreported IP address in a network that has many malicious IP addresses tends to be malicious than a network with a few such IP addresses[13, 14]. My intuition is that shared hosting can play a vital role along with a /24 bit IP subnet block to predict IP addresses for future malicious activities. In my proof of concept implementation, IP address X is reported as to be likely engaged in malicious activities if it satisfies all the following conditions:

- X is hosting multiple websites operated by multiple entities (i.e. shared hosting)
- X has not previously known to be malicious
- X is in a /24 bit subnet that has as least one IP address in GT-IP-List over the past N days, where N is the time-window size.

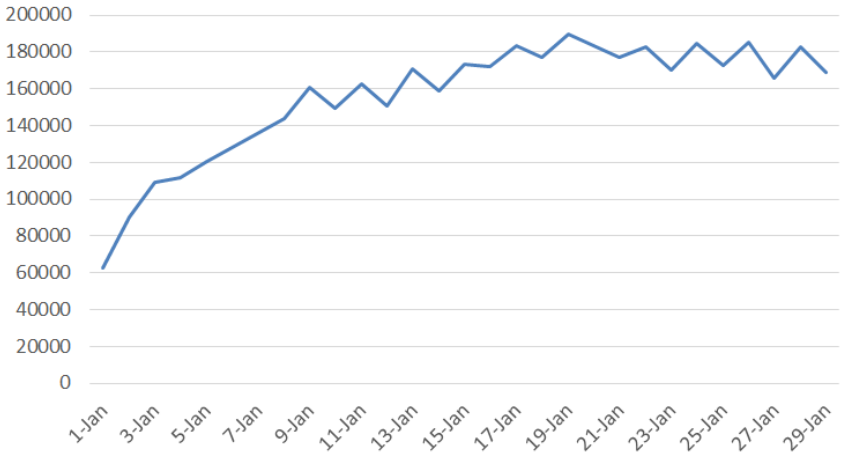


Figure 2.2: The size of IP Blacklist created based on GTMalware Dataset

In summary, to predict the list of potentially malicious IP addresses, it determines the /24 subnets that encompass reported malicious IP addresses and then for each of these subnets, it enumerates all the IP addresses that 1) were not appeared in the blacklists, and 2) is a shared host. These IP addresses are considered as potentially malicious IP addresses.

A time window (seven days) is also used to account for actions taken by service providers to "clean up the neighborhood". An IP address stays on the predicted list for only seven days if no new malicious activities are reported for rest of the IP addresses in the /24 subnet neighborhood. On average, 158,000 IP addresses exist in the blacklist per each day after the first seven days. Figure 2.2 shows the graph for the number of Blacklisted IP addresses throughout January 2017.

2.3 Evaluations

To evaluate my approach, I seek to answer the following research questions. First, how effective is this approach for preventing malicious activities? I choose to look at the detection of zero-day malware infections and blocking of phishing websites. In both cases, I benchmark my results against measures widely used by industry and show that my approach is better at detecting zero-day infections and blocking phishing

websites. Second, how much impact would my predicted blacklist have on normal business functions? Third, what is the most effective time window for prediction?

2.3.1 Zero-day malware infections

I use GT Malware to evaluate the effectiveness of the described approach in preventing zero-day malware instances. I also use VirusTotal as an oracle to determine the maliciousness of hashes in GT Malware. VirusTotal is a public online file scanning service that determines whether a file is a malware. In addition to scanning a binary file, one can query the VirusTotal database by giving the hash of a binary file. VirusTotal provides results from more than 60 antivirus (AV) products. AV products can mistakenly identify a binary as malicious (false positive). As an oracle, I use the the results of the following five high ranked antivirus products that are commonly used by today's businesses: `Kaspersky`, `McAfee`, `AVG`, `Avast`, and `Symantec`. To be more specific, a hash is labeled as malicious if it is regarded as malicious by at least one of the five AV vendors. To automate this process, I use VirusTotal Public API which was limited to 5,000 queries daily.

During Jan 2017, I randomly selected 5,000 unique hashes from GT Malware every day. I queried VirusTotal with each of those selected hashes immediately after GT Malware data becomes available. The line labeled "clean" in Figure 2.3 shows the daily number of hashes that were recognized as "benign" by all five AV products on that day. A significant subset of these "benign" hashes are predicted by presented approach as malicious because they contacted IP addresses in the malicious IP prediction list. The number of daily predicted malicious hashes that evaded the five AV products are represented by the line "predicted" in Figure 2.3.

To evaluate the accuracy of the prediction, I asked VirusTotal to rescan the all "benign" hashes again in March 2017, two months after the initial query. In the intervening period, these AV products have changed the "verdict" for some of the

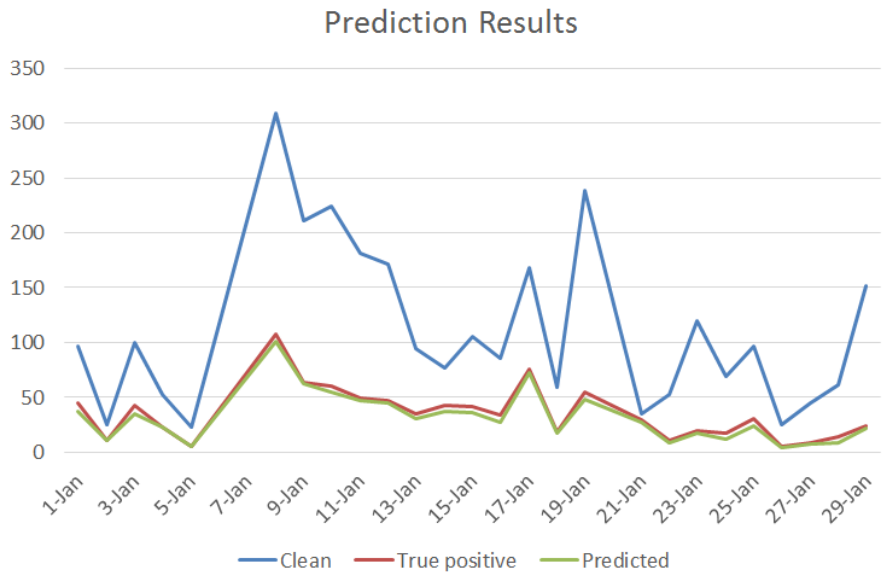


Figure 2.3: Zero-day malwares undetected by top 5 anti-viruses and predicted by the presented approach

hashes regarded as "benign" earlier. Hashes identified as malicious in the new scan by at least one of the five AV vendors were represented by the line "true positive" in Figure 2.3. On average, my method predicted 88% of zero-day malware instances missed by all five AV vendors.

A practical application scenario for my approach might be as follows. A zero-day malware got past AV and infected a machine in an enterprise. As soon as the malware starts to generate DNS traffic, the predicted IP list will be able to detect this infection and timely quarantine the infected machine.

Next, I evaluate the robustness of the suggested approach. Robustness, here, means how many different malware families this approach is able to detect. One can imagine a situation that a specific malware family uses shared hosting as part of its infrastructure. It has many variants, and the proposed approach may only be effective at detecting variants of this malware family.

For each malware instance the system successfully predicted, I queried VirusTotal for its identity. VirusTotal would return multiple answers, each provided by a different

Table 2.1: Diverse Variants of Malwares Detected

Trojan.ADH.2	PUA.Gen.2	Packed.NSISPacker!g4
Ransom.Cry	Downloader	PUA.Downloader
Trojan.Gen.2	Trojan.ADH	SecurityRisk.gen1
Infostealer	Trojan.Gen.8	Infostealer.Limitail
Trojan Horse	Trojan.Gen	Trojan.Gen.8!cloud
Trojan.Gen.6	Backdoor.Trojan	SecurityRisk.Downldr
PUA.DriverPack	PUA.InstallCore	Packed.Vmpbad!gen35
Ransom.Kovter	PUA.OpenCandy	Trojan.Zeroaccess!g3
PUA.Softonic	SMG.Heur!gen	PUA.ICLoader!g2
		ML.Attribute.High-Confidence

AV vendor. For this evaluation, I used results from Symantec. Table 2.1 lists 28 malware family names for malware hashes detected by the described approach during the period of evaluation (January 2017). The proposed approach appears to apply to a significant number of malware families.

2.3.2 Zero-day phishing websites

Phishtank is used to evaluate the effectiveness of the presented approach in preventing zero-day phishing attacks. Phishtank is a community-based phishing dataset. It accepts reports of phishing URL. Phishtank allows users to vote to determine whether posted URLs are indeed phishing sites. This process is time-consuming and hence many published URLs in Phishtank are not verified as phishing URLs by the users.

Moreover, VirusTotal is used as an oracle to determine the maliciousness of unverified URLs in Phishtank. If an unverified URL is identified as a phishing URL by at least two sources in VirusTotal, it is considered as a phishing URL.

70,953 phishing URLs published on Phishtank during July 2016 were collected. User voting results were also collected 30 days after each URL is first published on Phishtank. Based on user votes on Phishtank, 11,308 out of the published URLs were valid phishing URLs, and 319 URLs were not valid ones. 54,724 out of the remaining

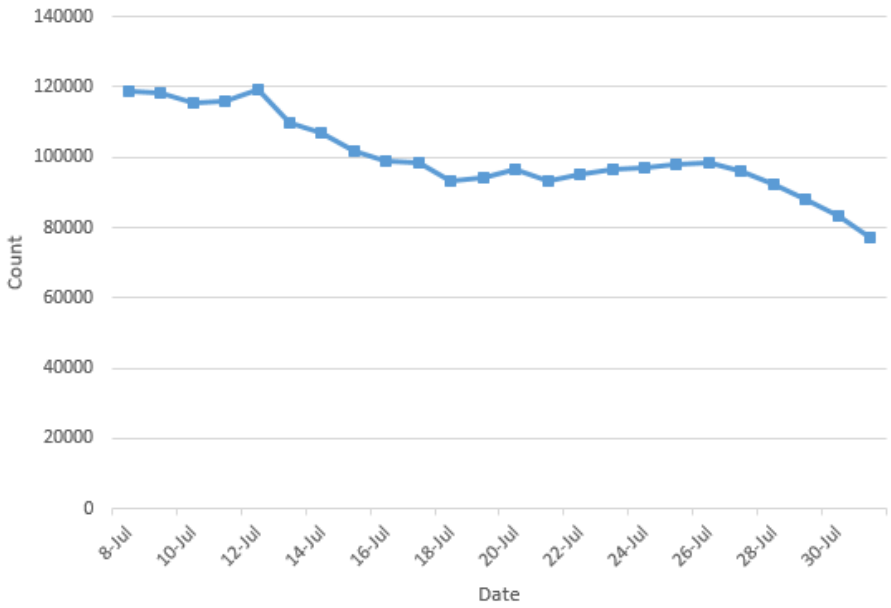


Figure 2.4: Number of zero-day IP addresses based on Phishtank dataset

URLs were reported as phishing URLs by at least two VirusTotal sources. These URLs were added to the dataset of valid phishing URLs.

I applied the presented approach on the collected phishing dataset to determine the number of phishing URLs it could have predicted. In this experiment, instead of relying on GT-IP-LIST to mark /24 subnets on the Internet, it considers the IP addresses associated with valid reported phishing URLs. Figure 2.4 shows the number of predicted IP addresses on each day during July 2016. On average about 100K IP addresses will be added to the list of reported IP addresses on each day. Figure 2.5 shows the total number of reported phishing URLs and the number of URLs that presented approach could have predicted on each day based on the resulted blacklist during July 2016. In my experiment, the system could have blocked about 68 percent of phishing URLs before they are reported to Phishtank.

I also checked whether the predicted phishing URLs belong to multiple phishing campaigns. To do so, I randomly selected a small number of phish links from predicted ones and manually examined them to determine which companies were the target of

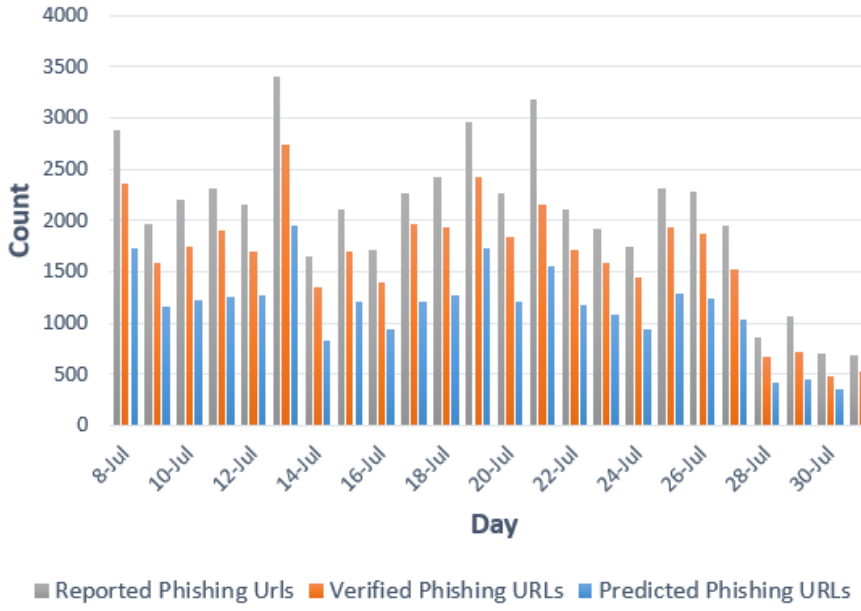


Figure 2.5: Number of predicted phishing URLs

these phishing links. Based on the observation, the predicted URLs targeted different companies, which shows that the presented approach can block a broad range of phishing attacks.

2.3.3 Impact on normal business functions

In this section, I evaluate the impact of my approach on normal business function. Clearly not all IP addresses predicted are malicious. I start by evaluating my hypothesis that responsible businesses tend to have better cybersecurity and that attitude is reflected in the selection of hosting providers. I used Alexa top 1,000 websites as a proxy for responsible businesses. Over the period of January 2017, only the IP addresses of the following four of Alexa top 1,000 websites appeared in the predicted blacklists by the described zero-day malware prediction approach: wordpress.com, wp.com, yandex.ua, 163.com, two of them were hosting WordPress contents. Note that WordPress sites are often blocked by large enterprises for poor security.

This evaluation suggests that vast majority of reputable businesses are not using

service providers that may have higher security risks. Additionally, the average size of our predicted IP blacklist (160,000 IP addresses) is a very small fraction of the Internet (.004% of IPv4 space). To minimize the impact on normal business, one may distinguish human initiated browsing traffic vs. automated traffic. For example, visiting to a Wordpress site may be okay for human initiated browsing. Outbound traffic to likely malicious IPs that is not generated by human browsing may be blocked to minimize the risk of malware infection. Human browsing exceptions may be made utilizing commonly available safe browsing features in major browsers.

2.3.4 Prediction time window

Presented approach predicts malicious IP addresses by considering observed malicious activities during a specific period of time. As the time window size increases, the number of predicted IP addresses increases as well. However, the size of time window must be limited as service providers often clean up malicious websites in response to reports. In this section, I evaluate the impact of the size of this time-window on the effectiveness of prediction using data from GTMalware.

Let $P_n(T_x)$ denote the percentage of malicious IPs (in GTMalware) predicted by my approach on day n for time-window T_x, where x is the size of time window. $P_n(T_x)$ is

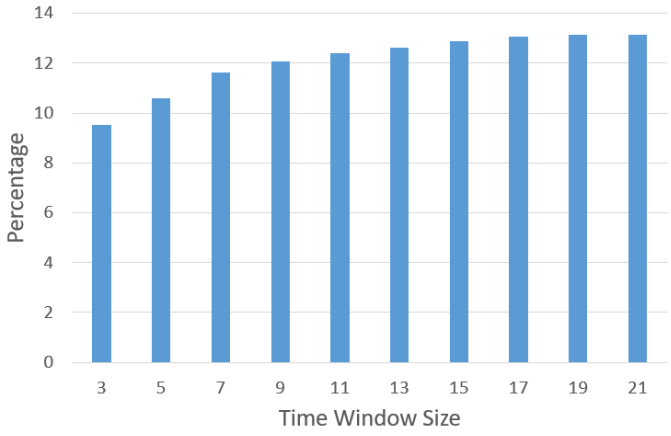


Figure 2.6: Avg. IP prediction percentages for different time-window size

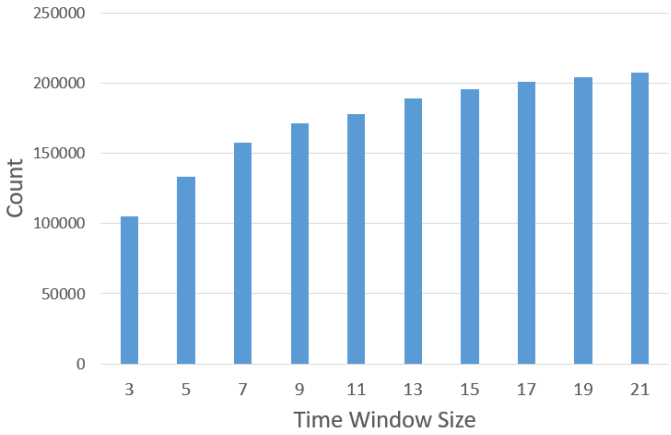


Figure 2.7: Change in Blacklist size over different time-window

calculated according to Eq. 2.1 where $BlockedIP_n(T)$ is the number of IP addresses predicted by my approach and $NewIP_n$ is the number of new unique IPs in the GT Malware dataset on the n-th day.

$$P_n(T_x) = \frac{BlockedIP_n(T_x)}{NewIP_n} \tag{2.1}$$

Let $\overline{P(T_x, N)}$ denote the average malicious prediction rate for a given time window T_x over N days (N = 30 days, January 2017) as shown in Eq 2.2.

$$\overline{P(T_x, N)} = \frac{1}{N} * \sum_{i=1}^{i=N} P_i(T_x) \tag{2.2}$$

$\overline{P(T_x, N)}$ is calculated for a number of time windows ranging between 3 to 21 days as shown in Figure 2.6.

It is evident that IP predication rates increases with time window size. However, the rate of increase decreases quickly. I also calculated the impact of time-window on daily blacklist size. Figure 2.7 shows that the average blacklist sizes for different time window sizes over the same one month (January 2017) time period. From Figure 2.7, it is clear that blacklist size increases with the time-window, but the rate of increase is decreasing similar to the average prediction percentages. One possible explanation

is that attackers constantly acquire new IP addresses in order to circumvent IP-blacklisting. As bad IP reports age, the chance that attackers still resides in the same subnet decreases. Moreover, as it is mentioned earlier, service providers clean up the malicious domains once they are reported. Therefore the chance of predicting another IP addresses as malicious on the same subnet after the time-window will also decrease. As an evidence supporting this hypothesis, I observed in GTMalware that on average over 70% of the malicious IPs overlap in the same /24 subnet between two consecutive days. That overlap drops to 30% for two days apart.

Based on the results in Figure 2.6 and 2.7, selecting a time window between 7 to 21 days is reasonable depending on how much time one might want to give service providers to take down malicious activities.

2.4 Related Work

IP blacklisting is a well-established practice in the security community, and many companies are relying on blacklists to defend against attackers. Traditionally, IP blacklists are created by compiling cyber threat intelligence reports from different sources. Researchers have proposed ways of using blacklists to enable network firewalls to mitigate different types of attacks. Zhang et. al [16] proposed Highly Predictive Blacklisting (HPB), which is a PageRank-like algorithm to rank attack sources based on threat intelligence sources. Soledo et. al. [17] has an Implicit Recommendation System that extends HPB by considering temporal patterns of cyber attacks to prioritize attack sources.

Although compiling a blacklist from a set of threat data sources can be beneficial for cyber defense, such blacklists only offer a rear mirror view of the threat landscape. In recent years, many researchers have attempted to tackle this problem by identifying features that are shared among cyber threats that can be examined on incoming network traffic to determine whether they should be blocked.

Several research works, e.g. [14, 18, 13], have shown that malicious activities are not uniformly distributed over the Internet. In other words, malicious activities tend to cluster together and form high risk communities [13]. The goal in such works is to identify high risk networks that host such malicious activities. Collins et. al. [14] presented the idea of spatial and temporal uncleanliness in network to predict botnet IP addresses. Stone-Gros et. al. [18] presented FIRE, FInding Rogue nEtworks, to identify ISPs that are responsible for the most malicious activities. Moura et. al. [13] coined the term Internet Bad Neighborhood. They showed that spamming activities tend to be clustered in bad neighborhoods by analyzing spammer activities on the Internet. In such works, a network is considered as high risk if enough number of malicious activities (above some predefined threshold) reported by cyber threat intelligence sources are reside in that network.

Other research works such as [19, 20, 10] have suggested features that can be calculated on an incoming network request to determine whether it is maliciousness without requiring a collection of threat reports. McGrath et. al. [19] proposed several features such as number of IP addresses with a domain, number of ASs that these IP addresses reside in, and DNS record TTL that can be used to determine whether a domain name is using a fast flux technique that is commonly used by phishers. Moghimi and Varjani [20] proposed another set of features including the number of dots in URL, SSL certificate, URL length, blacklisted keywords to identify phishing URLs. Bilge et al. [10] proposed a system, EXPOSURE, to detect malicious domains. EXPOSURE consider four different sets of features: time-based features, DNS answered based features, TTL value-based features, and Domain name based features.

During the course of this research, I found that some of the proposed features are not effective in predicting zero-day IP addresses based on GT Malware data. For example, many research works such as [19] have reported a very short domain

TTL is a good indicator for detecting malicious domains; however, I found that a significant number of reputable domains including Alexa top domains also have very short domain TTLs possibly due to the use of load balancers, or content delivery networks (CDNs). In this chapter, I introduce a new salient feature, shared hosting, that is strongly correlated with malicious activities.

Mine is a hybrid approach in which cyber threat intelligence data sources as well as shared hosting are used to identify potentially high risk network neighborhoods. The described approach has a lower threshold for the number of observed malicious activities to identify high risk network neighborhoods as I am not solely rely on cyber threat intelligence sources to predict zero-day malicious IP addresses. This approach is robust in that can be used to pro-actively identify a variety of infrastructures such as command and control servers, drive by download servers, and phishing web sites that are used by attackers to launch their attacks.

CHAPTER 3: Detecting Zero-hour Phishing Webpages

3.1 Introduction

Phishing, a type of social engineering attack, is one of most common attack types used by cyber attackers to lure unsuspecting users to disclose their sensitive information, such as user credentials, credit card information, or social security numbers. According to the Anti-Phishing Working Group (APWG), more than 1.2 million phishing attacks are documented in 2016, a 65% increase over 2015 [21]. Moreover, these attacks have evolved over time and become increasingly more advanced as phishers attempt to make the look of their phishing webpages and corresponding URLs as similar as possible to target websites while utilizing various evasion techniques to circumvent existing phishing detection mechanisms.

Phishers use various techniques to convince unsuspecting users by using valid SSL certificates [22], utilizing URL hijacking techniques such as typosquatting [23], and scraping information from target webpages. Phishers also use techniques to mislead existing phishing detection systems. This includes techniques such as using the image of the target webpage instead of reusing its HTML content, using old registered domains, and moving from one domain to another on a regular basis.

In recent years, many phishing detection systems have been proposed to combat the increasing number of phishing threats. These systems rely on a combination of features extracted from various sources such as search engines [24, 25], public blacklists [26], DNS records [24], URLs [27, 28], HTML documents [29, 28], and SSL certificates [1]. Despite achieving high accuracy, existing phishing detection systems suffer from several shortcomings that limit their applicability: 1) dependency on third

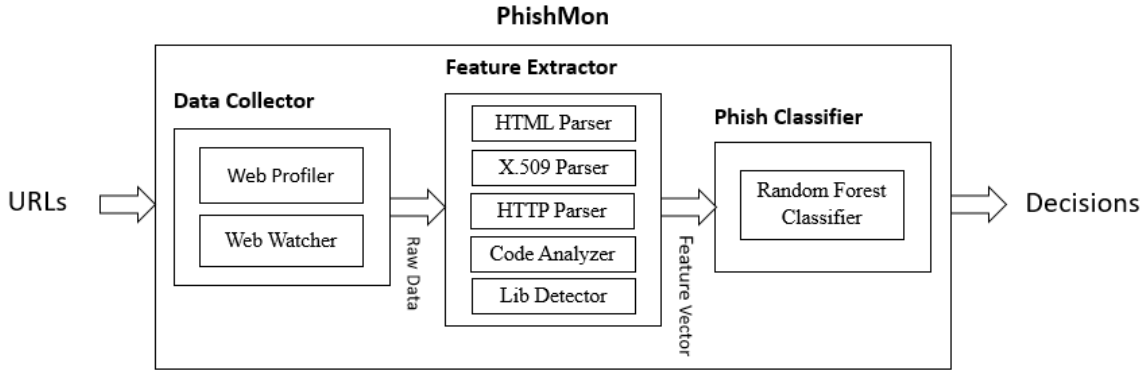


Figure 3.1: PhishMon Architecture.

party services such as search engines [29, 30] and WHOIS servers [30] introduces concerns of availability, cost, privacy, and performance 2) limited detection scope due to comparing the URLs or webpages under investigation with a whitelist of potential targets [31] or a blacklist of known phishes [32] 3) language dependency as they rely on features extracted from textual content of webpages [29] or URL [25].

In this chapter, I present `PhishMon`, a new scalable feature-rich machine learning framework for detecting zero-hour phishing attacks in a real-time fashion. It relies on a set of twenty salient features, seventeen of which are new features that characterize how the webpage is put together, such as certificate validity scope, number of external script blocks, that can be collectively used to discern phishing webpages from the legitimate ones with a high degree of accuracy. `PhishMon` exploits features extracted from HTTP requests/responses, SSL certificates, HTML documents, and JavaScript files when a given URL is loaded by a web browser. These features collectively reveal the characteristics of technology used to build and host a webpage. Such characteristics add significant cost obstacles for phishers if they want to evade detection. In summary, the presented system has the following properties:

- **System independence.** `PhishMon` does not depend on any third-party system to make a decision. It decides whether a URL is a phish based on features easily extracted from a webpage loaded by a web browser. Relying on third-party ser-

vices such as search engines can impose some limitation on the applicability of a phishing detection system. First, it can prolong the decision process; thus making the approach unsuitable for online detection. Second, using a third-party system can be costly in case of requiring a subscription. Third, sometimes finding the right provider is impossible, for example, not all domain registrar provides WHOIS records. Forth, using a third-party system can raise privacy concerns as the queries leak some information about the page under investigation.

- **Broad coverage.**. `PhishMon` can detect zero-hour phishing campaigns masquerading as previously unknown targets or brands as it makes a decision based on intrinsic features shared among phishing attacks and not based on features that measure the similarity of a given webpage with a curated list of legitimate or phishing webpages.
- **Language agnostic.** Classifiers that derive features from textual content are bound to a specific language such as English in which they were trained and are not effective against phishing attacks targeting websites in other languages such as Chinese. None of the features used by `PhishMon` are derived from the textual content of the webpage; hence making the approach language agnostic. In other words, `PhishMon` internal classifier can detect phishing attacks irrespective of the written language.
- **High accuracy.** `PhishMon` achieves high accuracy even when it is trained on a small training set. Curating a large training set containing phishing and non-phishing instances can be highly challenging.
- **Computation efficiency.** `PhishMon` decision-making process is highly efficient as it only relies on features that can be efficiently derived based on the webpage. I use the Random Forest algorithm to construct the classification model; the resulted model is also fast in making a decision.

In the rest of chapter, first, the overall system architecture of `PhishMon` is presneted in Section 3.2. Next, the underlying features that `PhishMon` utilize to detect

phishing webpages are enumerated and discussed in Section 3.3. Then, in Section 3.4, a comprehensive evaluation of `PhishMon` are provided. In Section 3.5, PhishMon is compared with existing phish detection systems.

3.2 System architecture

`PhishMon` is a novel phish detection system that employs a feature-rich machine learning framework to detect phishing instances. Figure 3.1 depicts its overall architecture. It receives a list of URLs as input and produces a list of binary decision values as an output indicating which of the input URLs point to phishing webpages. In a nutshell, `PhishMon` loads each input URL in a web browser and records the generated web traffics. From the collected data, it extracts twenty features, seventeen of which are new, and feeds them to its internal classifier to decide whether the webpage pointed by the input URL is a phish.

`PhishMon` consists of the following main components: `Data Collector`, `Feature Extractor`, and `Phish Classifier`. It operates in two modes: training and operational modes. In training mode, PhishMon analyzes the labeled input URLs and train its internal classifier. In the operational mode, it utilizes the trained classifier to detect phishes.

Although in this chapter I describe `PhishMon` as a standalone system, it can be implemented as an add-on for existing modern browsers since all the data necessary for obtaining the features are readily accessible through browser APIs. PhishMon can collect the features and decide while a webpage is loading without sending requests to third-party systems which can significantly prolong the decision making process.

3.2.1 Data Collector

`Data Collector` is responsible for collecting required information regarding the input URLs. It has two subcomponents: `Web Profiler` and `Web Watcher`. `Web Profiler`

is a headless browser, a fully functional web browser without a graphical user interface. In the current implementation, it is developed on top of .NET WebBrowser class. `Web Watcher` is a background daemon that monitors various public data sources, such as PhishTank and Alexa, to obtain new phishing and legit URLs. The collected information is used for training the PhishMon when it is in training mode.

`Web Profiler` loads each URL in a new web browser instance running in a separate thread and captures the whole HTTP(S) traffic exchanged between the embedded browser and the external web servers while loading the URL. This includes the HTTP response headers, the SSL certificates, the HTML documents, and other related files such as image, CSS, and JavaScript. `Web Profiler` waits for 30 seconds before ending a browsing session since some of the features proposed in section 3.3 such as `HTML version` measure dynamic properties of the webpage. It is worth noting that in case of integrating PhishMon with a web browser, it will not wait for any second as it will decide about the webpages on the fly while the user navigates on the Internet.

The main task of `Web Watcher` is to gather phishing and legitimate URLs from various data sources including Phishtank and Alexa on a regular basis. It utilizes web profiler to collect the web contents of these URLs. The collected information is saved on a local data store and used for training PhishMon. In addition to collecting fresh URLs, `Web Watcher` attempt to validate the label of curated URLs, whether they are phish or legit, by checking the URLs with public data sources after two weeks from their collection dates.

3.2.2 Feature Extractor

`Feature Extractor` extracts all the features described in Section 3.3 from the data passed by `Data Collector` . To do so, it utilizes several lightweight parsers to extract data from X.509 certificate files, HTTP headers, HTML documents, and JavaScript codes. It also employs two utility applications namely `Lib Detector` and `Code`

`Analyzer` to derive more abstract features. `Lib Detector` examines JavaScript files and determines whether the files are part of any open-source JavaScript library. `Code Analyzer` analyzes JavaScript code and extract code complexity related features. I explain these two applications in more details in section 3.3.2. After extracting the features by these tools, `Feature Parser` passes the resulted feature vector to the Phish Classifier component.

3.2.3 Phish Classifier

Similar to several phishing detection systems such as [30, 25], PhishMon formulates the problem of detecting phishing URLs as a binary classification problem in which the task is to determine whether a given URL is phish or benign. To predict the class of an input URL, PhishMon represents the URL and its related information as an n-dimensional feature vector and feeds this vector to its internal classification model to make a decision. This classification model is constructed during the training mode by applying the Random Forest classifier on the dataset collected by `Web Watcher` on a regular basis. I use the Random Forest implementation provided by scikit-learn machine learning library [33] in the current implementation of `PhishMon`.

3.3 Feature Selection

In this section, I describe the twenty features, seventeen of which are new, that `PhishMon` uses to decide whether the webpage pointed by a given URL is a phish. I start with the observation that the average lifetime of a phishing webpage is short, measured in hours [34, 35, 36]. Phishers need to constantly create new webpages before getting blacklisted either based on URL or content. To reduce cost, phishers tend to focus their efforts on webpages' appeal to victims and do not pay much attention to web technologies and infrastructures used to develop and host their web applications. In contrast, legitimate business websites are increasingly becoming more technology

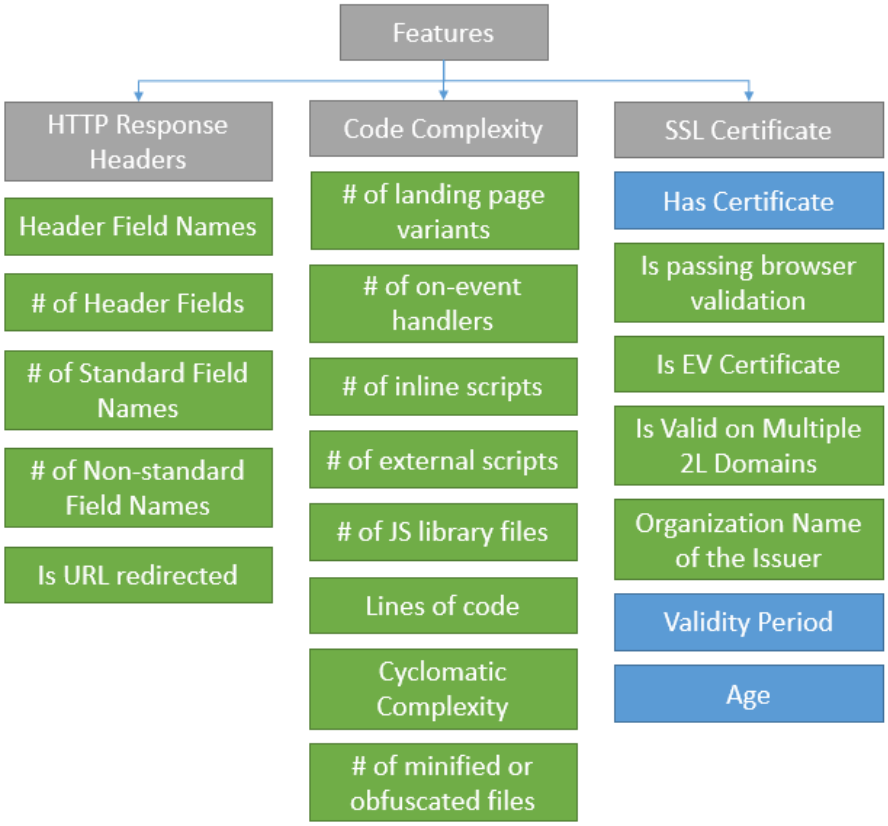


Figure 3.2: PhishMon feature set. The green-colored features are the new ones proposed in this proposal for detecting phishes. The blue-colored features are used in other research works [1].

savvy and pay attention to issues such as security of web applications, quality of service, and following best development practices to enhance code maintainability.

I have identified the followings are often used by business websites targeted by phishers: (I) security techniques to protect users from client-side attacks such XSS and CSRF, (II) security techniques to prevent web scraping, (III) security techniques to protect users traffics from sniffing, (IV) tracking techniques to monitor user activities, and (V) techniques to reduce the loading time of webpages. In addition, website developers use best practices, such as separation of JavaScript code from HTML content, to improve the maintainability of their applications.

Based on these observations, I introduce three groups of features, depicted in Figure 3.2, to characterize the techniques, mechanisms, and technologies that are used by

websites. These features indirectly measure the developmental efforts and technological investments associated with websites. They would require substantial resources for phishers to mimic.

To ensure the effectiveness of proposed features in distinguishing phishes from legit websites, I conduct a series of experiments on the dataset that I constructed from 2,064 phishing and 17,508 legitimate webpages. This dataset is a part of the larger dataset described in Section 3.4.1; it only has phishing instances reported in September and October.

3.3.1 HTTP Features

Websites use HTTP headers to pass additional information to web browsers. Exchanged HTTP headers can reveal information about the underlying web application and its technology infrastructure. In fact, web application fingerprinting tools such as Wappalyzer [37] analyze such information to identify back-end technologies for a given website. I have observed that the underlying web technology stacks are considerably different between phishing and legitimate websites. Their owners have different objectives that influence their spending on technology and infrastructures. As a result, PhishMon attempts to utilize returned HTTP headers as part of the fingerprint for phishing sites.

In addition, many commercial websites use JavaScript and Ajax to dynamically change the appearance and content of websites. Many phishers use redirection techniques to hide the actual URL of a phishing webpage. To handle these cases, PhishMon employs a headless web browser based on the Internet Explorer rendering engine. This browser captures the original HTML document and its related documents. PhishMon monitors a webpage for 30 seconds for the page to "settle" before saving the active URL, the rendered HTML documents, and all the related documents. In this way, it can detect redirection or page content changes via JavaScript and Ajax

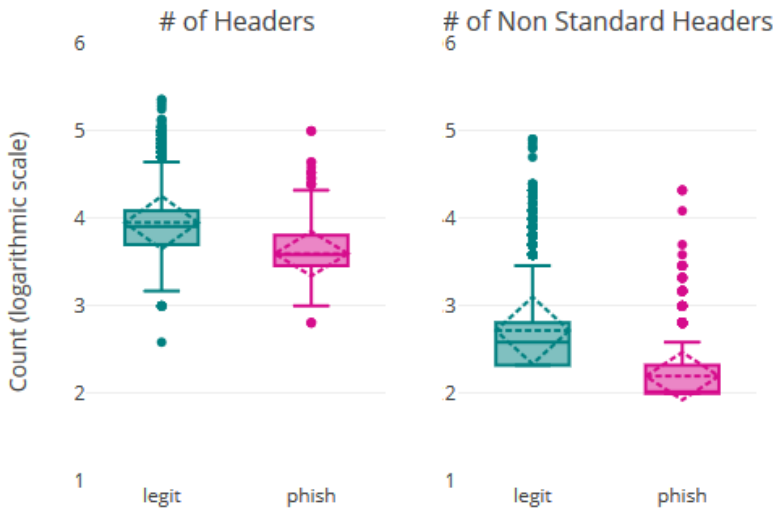


Figure 3.3: Comparison of HTTP header features between legitimate and phishing websites

calls.

HTTP header field names. PhishMon considers all the header names appeared in an HTTP response as a feature. To represent this feature, a vector space model [38] is used. In this model, an HTTP response is represented as a vector. Each dimension in this vector corresponds to a specific header name. If a header name appears in an HTTP response, its value in the vector will be the length of the corresponding field value measured in bytes; otherwise, its value will be -1. In my test dataset, there are 2,123 distinct header names.

Number of header fields. PhishMon counts the number of headers appeared in the header section of the HTTP responses. As figure 3.3 depicts, on average, the number of header fields in a response of legitimate websites is greater than the phishing ones.

Number of non-standard header fields. PhishMon computes the number of non-standard header fields in the header section. These headers are not on the list of standard HTTP header names provided by IANA [39]. As of writing this chapter (April, 2018), this list contains 329 permanent header names.

3.3.2 Code Complexity Features

The front-end code of a phishing webpage focuses on simulating the appearances of its target. It does not contain other features of the target site. This difference is reflected in the code complexity of the webpages. In this section, I explain a set of features that `PhishMon` extracts from a webpage to capture the code complexity of the underlying web application. These features attempt to capture the fact that a full-fledged website offers a variety of functionalities to its users, some of which are triggered by users' actions, and the rest are triggered by other events such as timeout. Phishers, on the other hand, mainly concern about the visual aspect of the webpages by mimicking the target UI as much as possible to mislead the unsuspecting users. As a result, the complexity of the JavaScript code included in phishing webpages is significantly less than the target.

Researchers have proposed various metrics including lines of code, number of functions, number of variables, and cyclomatic complexity to predict less maintainable [40] or more vulnerable applications [41, 42, 43]. However, in this work, I utilize such measurements to determine the codes that offer more functionalities to end-users. It is noteworthy that before computing some of these features, such as lines of code, PhishMon must perform code formatting on the input since in many web applications, JavaScript files are minified; during minification process, all the unnecessary characters are removed from the code while preserving its semantics. `PhishMon` employs jsbeautifier library to format JavaScript source code to reverse code minifications.

Many modern web applications rely on off-the-shelf libraries, such as `JQuery`, `JQuery UI`, and `Bootstrap` to build their web user interfaces. To measure the code complexity of a webpage, PhishMon must ignore such libraries as they are implemented by third-party developers. To recognize such libraries, I collected the top 200 JavaScript libraries (8637 versions of them until Oct 2017) listed by `cdnjs` website and computed the context triggered piecewise hashes (CTPH), a.k.a fuzzy hashes, for

all of their JavaScript files using `ssdeep` library [44]. Before extracting any features from a JavaScript file, `PhishMon` computes its CTPH value and compare it with its internal list of CPTH values to ensure the file under examination is not part of any well-known library.

An entry point URL might be recursively redirected before reaching the landing URL. In such cases, PhishMon only considers the webpage point by the landing URL, the last URL in the chain, to compute the features described below.

Minified/Obfuscated. A common practice among legitimate websites is the use of code minifiers [45], or obfuscators to both reduce the page loading time and protect the front-end code against web scrapers. As mentioned by several researchers [44], usage of code obfuscation is also fairly common among some types of malicious webpages such as drive-by-download webpages; however, I observed that a large percentage of phishing websites does not use any of such techniques. PhishMon uses the following group of features to determine whether a script file is minified or obfuscated: the ratio of white spaces to all printable characters, the average length of variable names, and the average length of function names. For a webpage with several external JS files, PhishMon considers the average of these features.

Number of external script blocks. A common practice to enhance code maintainability is to store JavaScript code in separate files from the HTML documents and reference such files in HTML documents by using external script blocks. PhishMon counts the number of script blocks in the landing webpage.

Number of inline script blocks. Despite objections against usage of inline script blocks because of security reasons, they are still commonly used even in popular websites such as Google and Amazon. As in this way, developers can reduce the number of round trip times (RTTs) required for loading external script files; hence they can significantly reduce the page loading time. This feature measures the number of the inline script blocks in the base HTML document loaded by the web browser.

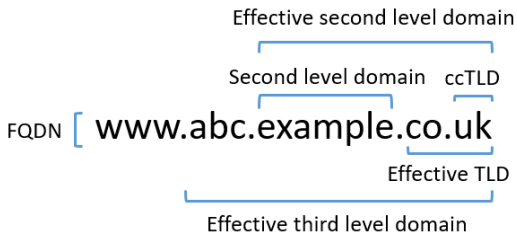


Figure 3.4: Common domain name

The following HTML code fragment depicts a typical inline script block:

Code 3.1: Inline script block

```
<script type="text/javascript" >
[JavaScript code]</script>
```

Number of DOM on-event handlers. DOM events, such as onclick and onload, allow JavaScript code to register event handlers on DOM elements, such as body and image, in an HTML document. In this way, the handlers will get notifications when events of interest have occurred on the specified elements. Code 3.2 shows a sample of on-event handlers for onclick event.

Code 3.2: DOM on-event handlers

```
<img onclick="[on-event handlers]" src="sample.png" />
```

Unlike inline script blocks, developer best practice refrain from using on-event handlers since placing such handlers inside HTML elements can make the code significantly less maintainable. However, on-event handlers are handy tools for rapid prototyping; hence more often used by mock or phishing webpages. In the current implementation, PhishMon separately counts the handlers registered for each of the following DOM on-events: `onclick`, `onload`, and `onchange`.

Number of JavaScript libraries. PhishMon also counts the number of JS libraries that are referenced in the HTML document of the landing webpage. As I mentioned earlier, PhishMon utilizes a JavaScript library detector that recognizes

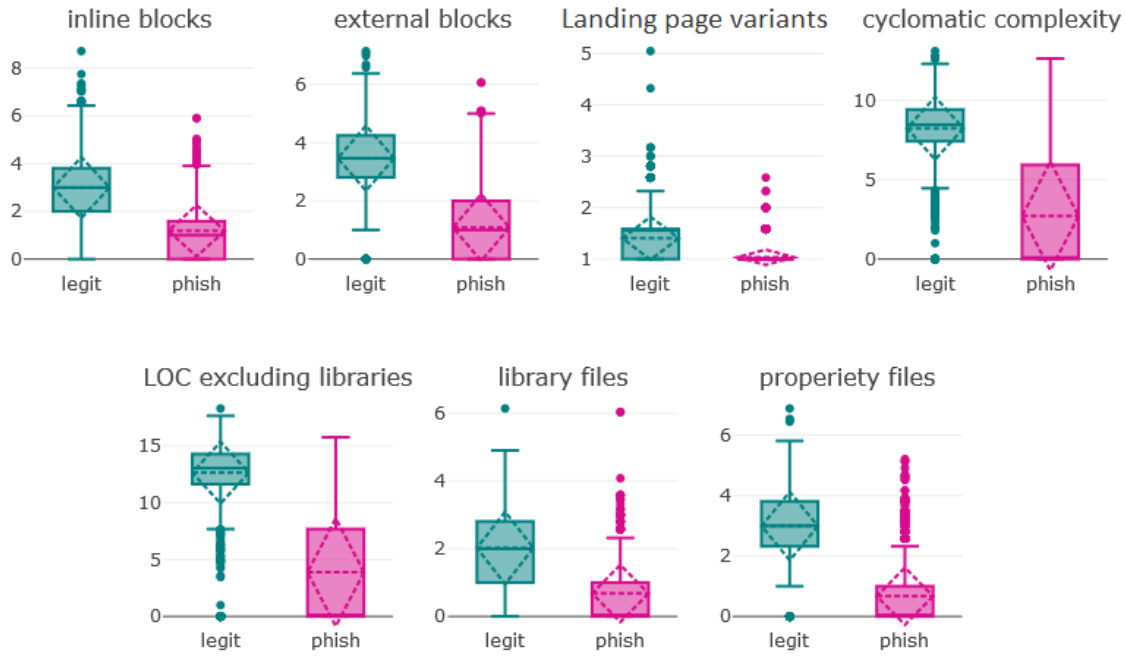


Figure 3.5: Comparison of Web UI code complexity metrics between phishing and legitimate websites. Numbers are expressed on a logarithmic scale (base 2)

more than 8,600 different versions of popular libraries.

Number of Landing Page Variants. Many commercial web applications use JavaScript and Ajax calls to dynamically change the look and the content of a webpage without refreshing the page. This feature captures the number of times the base HTML document is changed by Ajax calls during the observation period.

Is URL redirected. This feature shows whether the initial URL is redirected to other effective second-level domains during the observation period. An effective second-level domain name is a concatenation of a second-level domain name plus an effective TLD. Figure 3.4 shows different terms that are used to refer to different parts of a domain name. PhishMon relies on the public suffix list (PSL) published by Mozilla Foundation [46] to correctly recognize the effective TLD for a given domain name.

Lines of Code (LOC). PhishMon computes the average number of lines in exter-

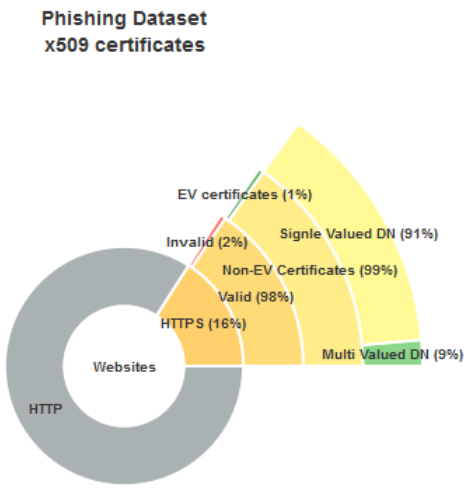


Figure 3.6: Phishtank Dataset - Types of X509 Certificates

nal JavaScript files, excluding libraries. This is intended to measure the amount of developer effort spent of building the website. To ensure that the LOC value calculated from different files reflect the number of statements, PhishMon uses a reformatter, in this work jsbeautifier, to normalize the format of the code before counting lines.

Figure 3.5 depicts a box plot for each of the numerical features that PhishMon considers for code complexity over the collected dataset. Y-axis values in this figure are on a logarithmic scale (base 2).

3.3.3 Certificate features

PhishMon extracts a set of features from X.509 certificates provided by websites hosted over HTTPS. As depicted in figure 3.6, about 16 percent of phishing websites in my phishing dataset are also hosted over HTTPS with valid SSL certificates. I compared attributes of digital certificates of phishing websites vs. commercial websites. I observed that certain types of certificates are more commonly used by legitimate websites (see Figure 3.7). Moreover, 98 percent of certificates used by phishing websites had valid certificates. One percent of these valid certificates are extended validation (EV) certificates, and about nine percent of non-EV certificates are valid on multiple domains.

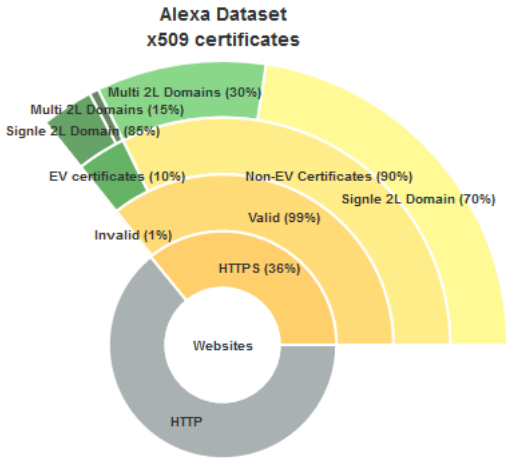


Figure 3.7: Alexa Dataset - Types of X509 Certificates

Has X.509 certificate. This feature indicates whether the website is hosted over HTTPS.

Is passing browser validation. This feature indicates whether the provided X.509 certificate is valid and is issued for the requested domain. A valid X.509 certificate must not be expired or revoked at the time of access. It also must have a valid signature. Moreover, the certificate of CA that signed the certificate must be valid. This chain of valid certificates must end with a valid self-signed certificate that is trusted by the validator. To determine the validity of a certificate, PhishMon relies on its underlying web browser.

Valid on multiple 2-level domain names. This feature shows the number of second-level domain names that this certificate is valid for. To extract this feature, PhishMon counts the number of distinct second-level domain names declared in the Subject Alternative Names (SAN) extension of an X.509 certificate. The default value for this feature is one. I observed that the majority of such certificates (about 46 percent) belongs to CloudFlare which offers Content Delivery Network (CDN), DDoS mitigation, and Internet security services to its client.

Extended Validation (EV) Certificate. This feature indicates whether the certificate is an Extended Validation (EV) certificate. To obtain an EV certificate

Table 3.1: Market share of Certificate Authorities

Phishing websites		Legitimate websites	
CA (organization name)	%	CA (organization name)	%
cPanel, Inc	41	COMODO CA Limited	29
COMODO CA Limited	23	Let's Encrypt, C	17
Let's Encrypt	23	GeoTrust Inc., C	11
GoDaddy.com, Inc.	4.1	GoDaddy.com, Inc.	7
GeoTrust Inc., C	1.2	DigiCert Inc, C	6.2

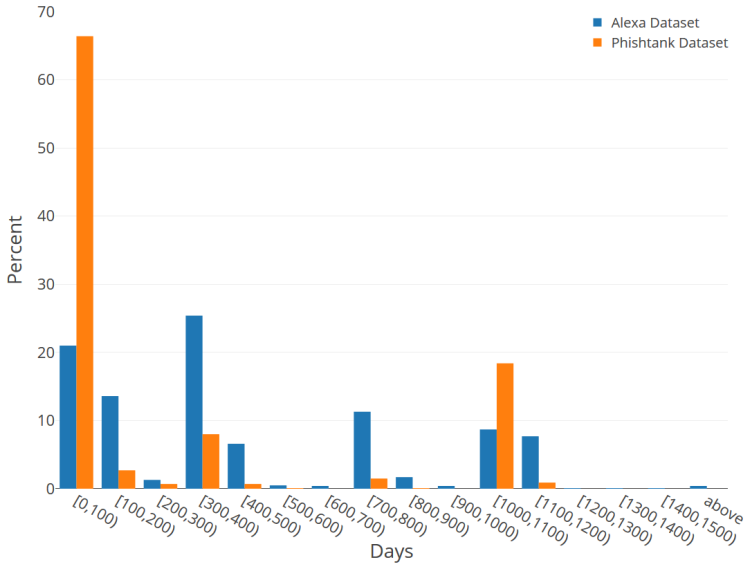


Figure 3.8: Distributions of certificate longevity period in phishtank and alexa datasets.

for a domain name, one needs to go through a standardized vetting process to prove they legally own the domain.

Name of the Issuer. This feature indicates the names of Certificate Authority (CA) that issued the certificate. Table 3.1 show the top five CAs in both Alexa and Phishtank datasets. As it can be seen, some CA are more targeted by phishers as they issue certificates with minimal cost. For example, one can obtain a certificate from X for free.

Certificate Longevity Period. This feature indicates the number of days a given certificate is valid. Figure 3.8 shows the distribution of longevity period in Phishtank and Alexa dataset. As it can be seen in this figure, although the longevity of most

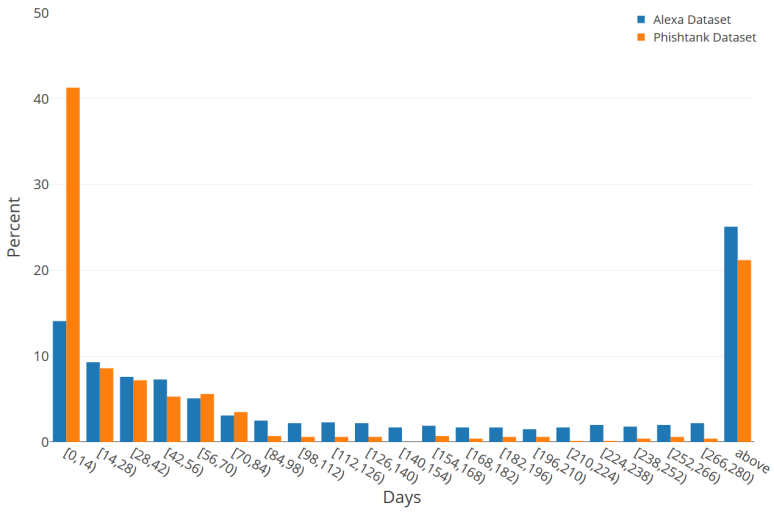


Figure 3.9: Distributions of certificate age in phishtank and alexa datasets.

Table 3.2: Comparison of binary features extracted from X.509 certificates

Feature	Bayes Factor	Is a Phish?
EV certificate	0.035	Strong rejection
Valid on multi 2L domain	0.144	Moderate rejection
Has X.509 certificate	0.44	Anecdotal

phishing certificates is less than 100 days, more than 20% phisher certs appear to have similar longevity as legit websites.

Certificate Age. This feature indicates the number of days past from the issuance date. Again, although most phishing certificates are issued in less than 50 days, more than 20% phisher certs appear to have much longer issuance dates, similar to legit websites.

To show the relative strength of proposed binary features in the presented framework, I report their corresponding Bayes factor values in Table 3.2. Bayes factor is defined as a ratio of likelihood probability of two competing hypotheses, in this work being a phishing or a legitimate URL. Bayes factor, K, is calculated by the formula 3.1.

$$K_i = \frac{Pr(F_i = true|C = phish)}{Pr(F_i = true|C = legit)} = \frac{Pr(C = phish|F_i = true)}{Pr(C = legit|F_i = true)} * \frac{Pr(C = phish)}{Pr(C = legit)} \tag{3.1}$$

Where F_i is a binary feature, C shows the type of the webpage (*i.e.*, phish or legit webpage), and Pr is a conditional probability function.

3.4 Evaluation

In this section, I present my evaluation of PhishMon, in which I seek to answer the following research questions. First, how effective is the approach in detecting zero-hour phishing webpages? Second, what is the contribution of each feature in the proposed detection model? To answer these questions, I first compare the performance of several well-known classifiers when they are trained on a large dataset consisting of both phishing and legitimate websites. Then, I examine the power of prediction for each of the proposed feature in the system. I also report the performance of the system when a subset of features is considered.

3.4.1 Dataset

To study my approach, I collected a large dataset containing $22,315$ distinct instances of phishing and legitimate webpages. In this dataset, legitimate webpages are homepages of the popular websites selected randomly from the Alexa top one million domain name list, and phishing webpages are distinct confirmed phishing instances reported by PhishTank, a community-driven website for sharing and validating phishing URLs.

In the dataset, the number of legitimate instances is $17,508$, which is about 3.6 times of the phishing ones, $4,807$ (*i.e.*, 21 percent of the dataset represents phishing instances). The main reason for making an imbalanced dataset is because, in reality,

the number of legit websites is much larger than the number of phishing ones; thus making it a better representation of reality.

To pick legitimate webpages, I divided Alexa top one million list to four sublists: sublist A contains the top 1,000 domain names, sublist B contains the top 10,000 domain names excluding the top 1,000 names, sublist C contains the top 100,000 domain names excluding the top 10,000 names, sublist D contains the remaining domain names excluding the top 100,000 names. Then, I selected 1000, 1000, 8000, and 10000 domain names from sublist A, B, C, and D respectively. In this way, I bias toward selecting more reputable websites, which have a larger audience. I assume homepages for these Alexa sites are not controlled by attackers. To ensure the validity of this assumption, I cross-checked the selected websites with the public blacklists provided by malwaredomains.com and networksec.org and filtered out any sites being blacklisted. I visited the homepage of the remaining websites with my web scraper to form the dataset of legitimate webpages. Further, I removed webpages that contain certain phrases indicating the site is under construction, not functional, or not supporting the web engine used by the customized web scraper. In this way, I obtained $17,508$ webpages from Alexa list; the size of this dataset (downloaded text content of these webpages) is about 46 GB.

I also collected live phishing webpages by monitoring PhishTank for four weeks between September and November 2017. PhishTank is a community-driven phish reporting website, in which members submit phishing URLs, and also vote for or against submitted ones to declare whether they are indeed valid phishing URLs. Unfortunately, the validation process takes time, and many submitted URLs never receive a vote from the community. A URL is considered as a phish if it is confirmed by either PhishTank or VirsusTotal within two weeks of its initial addition to PhisTank. However, the webpage contents corresponding to URLs are fetched within five minutes of the time they were added to PhishTank. In this way, $27,311$ valid phishing URLs

were collected.

I found many phishing URLs are hosted on the same domains, or their webpages having almost identical HTML documents. Such phishing URLs may belong to the same phishing campaign. I took the following steps to avoid biasing the classifier toward fitting larger phishing campaigns. First, for all phish URLs that share the same domain, I picked only one of them and ignored the rest (6,671 URLs remained). Second, I filtered duplicate phishing webpages by comparing fuzzy hash values of their HTML content (4,807 URLS remained). Two URLs are considered as duplicate if the similarity score of the fuzzy hash values for their respective landing page content is more than 95. In my experiments, `ssdeep` library is used to compute and compare fuzzy hash values.

3.4.2 Selection of Machine Learning Algorithm

I formulate the phishing detection problem as a classification problem in which my aim is to determine whether an input URL is a phish. As mentioned in Section 3.2, I use Random Forest (RF), a decision-tree based ensemble classifier, to build a classification model. To choose this classifier, I performed a series of experiments on four standard machine learning algorithms, namely CART, K Nearest Neighbors (KNN), AdaBoost, and Random Forest (RF). All of these classifiers were trained on the dataset described earlier in Section 3.4.1.

I adopted the stratified 10-fold cross-validation strategy to estimate the performance of the classifiers under evaluation. The reason that I picked this strategy is twofold. First, this strategy is commonly used by many researchers such as [47, 34], so selecting this strategy makes it easier to compare my results with related works. More importantly, as it is shown in [48], the stratified 10-fold cross-validation, in general, tends to provide a less biased estimation of the accuracy.

In this strategy, the dataset is first divided into separate groups based on the class

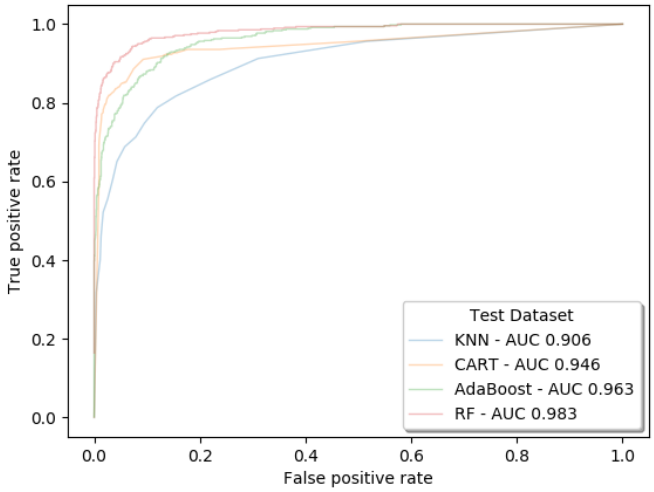


Figure 3.10: ROC curves of different classifiers trained on the collected dataset

of the instances. Each of these groups is called strata. In my case, I have phish strata and legit strata. Each strata is further divided into ten partitions, also known as folds. Then, I keep the first partitions from phish strata and legit strata for testing, and use the remaining parts for training the classifier. Next, I keep the second partitions for testing and the rest for training. I do this ten times so that all parts are used in both training and testing phase. I average the performance achieved by these trained classifiers.

The Area Under the ROC Curve (AUC) is used [49] as a performance measure to compare the classifiers. The ROC curve for a binary classifier is a plot that depicts the relationship between false-positive rate and true-positive rate when different probability thresholds are used by the classifier to make a decision. As stated in [49], the AUC of a classifier can be interpreted as the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance by the classifier. Figure 3.10 depicts the ROC curves [49] of the candidate classifiers when stratified 10-fold cross-validation approach is employed. As it is easily observable, the AUC for Random Forest classifier is larger than the other classifiers; which

Table 3.3: Random forest over all the proposed features (Accuracy: 95.4%, False Positive Rate: 1.3%)

n=22460	Predicted: Benign	Predicted: Phish	
Actual: Benign	TN=17419	FP=231	17650
Actual: Phish	FN=794	TP=4016	4810
	18213	4247	

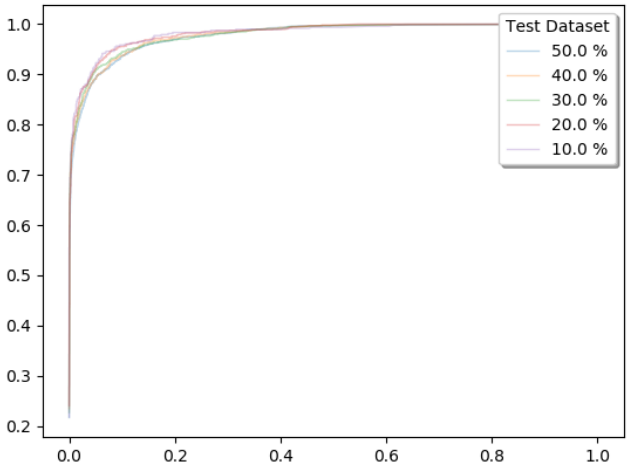


Figure 3.11: ROC curves of classifier trained on datasets with different ratio of training set to the whole dataset. The ratio varies from 10 to 50 percentage of the training dataset.

means that it can reach to a lower false-positive rate while keeping the true positive rate higher.

Table 3.3 shows the detail performance of Random Forest algorithm on the collected dataset when stratified ten-fold cross-validation is performed. It worth noting that the numbers are the summation the ten test runs. TN, FN, FP, and TP in this table stand for true negative, false negative, false positive and true positive respectively.

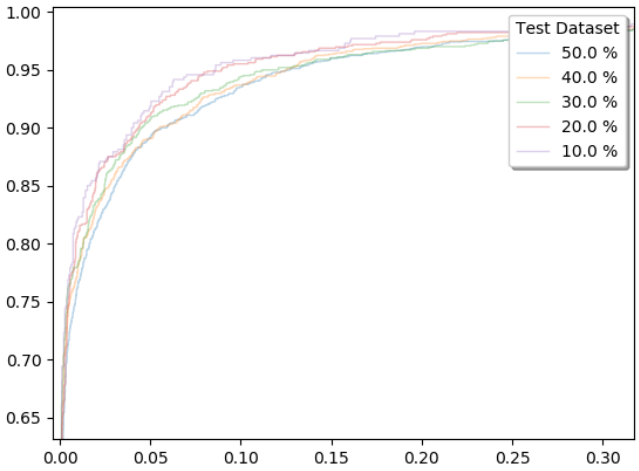


Figure 3.12: A closer look to ROC curves of classifiers trained on datasets with different ratio of training set to the whole dataset.

3.4.3 Impact of Training Size

I also evaluate the accuracy of the RF classifier when it is trained with different percentages of the dataset. Figure 3.11 depicts the results when the classifier is trained with only 50% to 90% percent of the dataset and kept the remaining 50% to 10% of the dataset for testing. As the percentage grows, the area under ROC curve increases. However, increasing the training size has a marginal effect on the performance of the classifier. For testing, 90% is used for training and 10% for testing. In the dataset, the ratio of phishing instance is about twenty percent.

3.4.4 Performance of Features

In this section, I first evaluate each category of features presented in section 3.3 individually to determine their power of prediction in detecting phishing URLs. Next, I evaluate the contribution of each feature on the overall performance of PhishMon classifier. In other words, I identify the most effective features in `PhishMon` for identifying phishes. These insights can help future feature development and also help researchers to better understand how might phishers react to evade the presented

Table 3.4: RF classifier trained on certificate features. For training and testing this classifier, websites hosted on HTTP were removed from the main dataset.(Accuracy: 92.6%, False Positive Rate: 0.6%)

n=8400	Predicted: Benign	Predicted: Phish	
Actual: Benign	TN=7035	FP=45	7080
Actual: Phish	FN=574	TP=746	1320
	7609	791	

Table 3.5: RF classifier trained on code complexity features (Accuracy: 91.2%, False Positive Rate: 4%).

n=22300	Predicted: Benign	Predicted: Phish	
Actual: Benign	TN=16755	FP=755	17510
Actual: Phish	FN=1190	TP=3600	4790
	17945	4355	

detection system.

I trained a Random Forest classifier on top of each feature category and measured the performance of these classifiers by using stratified 10-fold cross-validation approach. Tables 3.4 to 3.6 show the overall performance of three classifiers trained with certificate, code complexity, and HTTP header features. They achieved 92.6, 91.2, 93.2 percent accuracy on the dataset. These results show that these feature sets roughly have the same power in predicting phishing webpages; in other words, none of these group of features are dominant.

To evaluate the contribution of each feature on phish detection, I calculated the Mean Decrease Impurity (MDI) importance [50] of each feature in the classifier when it is trained on the dataset. Table 3.7 shows the top 15 most important features in the presented classification model.

Table 3.6: Random Forest over HTTP header features (Accuracy: 93.2%, False Positive Rate: 2.7%).

n=22300	Predicted: Benign	Predicted: Phish	
Actual: Benign	TN=17021	FP=489	17510
Actual: Phish	FN=1025	TP=3765	4790
	18046	4254	

Table 3.7: Top 15 most important feature based on MDI

No.	Feature	Feature Group
1	Avg cyclomatic complexity	Code complexity
2	LOC of external blocks	Code complexity
3	Avg number of external blocks	Code complexity
4	Set-Cookie	HTTP response header
5	Number of landing page variants	Code complexity
6	Number of HTTP headers	HTTP response header
7	Proprietary code count	Code complexity
8	Is URL redirected	Dynamic content
9	Avg number of inline blocks	Code complexity
10	Keep-Alive	HTTP response header
11	Number of library blocks	Code complexity
12	Cache-Control	HTTP response header
13	Content-Type	HTTP response header
14	Server	HTTP response header
15	X-Frame-Options	HTTP response header

3.5 Related Work

There is a large body of previous research to detect phishing websites. In this section, I briefly discuss and compare my work with some of the representative works.

Zhang *et al.* [29] proposed `CANTINA`, a content-based approach in which the key terms are extracted from a given webpage and fed into a search engine such as Google to examine whether the URL for the page appears in the top N search results. Xiang *et al.* proposed `CANTINA+` [30], a successor of `CANTINA`, which relies on fifteen distinctive features to identify phishing websites. It achieved 90% TP and about 0.4% FP on a dataset with 10% unique training phish instances. However, to achieve this level of accuracy, it relies on third-party services namely search engines and WHOIS servers to compute five of the proposed features. As mentioned in the introduction section this can cause availability, cost, performance and privacy issues. My approach does not utilize any third party system to reach a decision. Furthermore, CANTINA+ relies on some HTML-based features such as whether a page contains `bad forms`, `bad action fields` that can be easily manipulated by an attacker; for example, using images of text to avoid their text-based detectors. My proposed approach does not have these limitations.

Marchal *et al.* [25] proposed PhishStorm, which is a phish detection system that analyzes a given URL by extracting features from the words in the URL and querying `Google Trends` and `Yahoo Clues`. The system achieved a correct classification rate of 94.91% with only 1.44% false positives on a dataset consisting of 96018 phishing and legitimate URLs. Similar to the two previous works, this system interacts with external systems; thus suffering from the same shortcomings.

Chen *et al.* [31] proposed a phishing detection system that calculates the similarity score of a suspicious webpage with a list of legitimate webpages. The system compares the screenshot of the given page with the screenshots of all webpages on a whitelist; in case the similarity score with one of them is above a certain threshold, the page

will be marked as a phish for that whitelisted webpage. In their experiment, the system could reach 95% to 98% percent accuracy for different legitimate webpages. The main advantage of their system is that they are not relying on the HTML content of a webpage to determine whether it is a phish; hence evasion techniques such as using images instead of texts will be ineffective. However, to determine whether a page is a phish, the system needs to compare it with all the webpages in the whitelist which may contain millions of pages. In addition, this system can only detect phishing attacks against legitimate websites on a given whitelist; which means phishing attacks against unlisted legitimate websites can slip under the radar. In my approach, I do not compare with a list of legitimate webpages. In addition, `PhishMon` can detect a phishing page that use images instead of text to evade existing content-based detection techniques.

Chang *et al.* [51] proposed another system which utilizes Google Image Search service to identify the website identity based on the segmented website logo. The system is resilient against image-based evasion techniques and can achieve 87% TP and 30% FP on a dataset containing segmented images of webpages. The main limitation of the system is its performance as it sends a number of image queries to Google Image Search and a number of queries to Google Search Engine. In addition, due to the difficulty of logo extraction, the detection accuracy is less than the other approaches.

Dong *et al.* [1] proposed a real-time phishing detection system that can detect phishing webpage host on HTTPS-enabled web servers. It extracts 42 features from X.509 certificated to detect phishing websites and reach a recall of 95.5% in the phishing category, with a precision of 93.7 on average. The main limitation of the proposed system is that it can only detect phishing websites hosted over HTTPS. I also extract seven salient features, three of which are new ones, from X.509 certificates. However, my system extract features from other sources such as HTML and JavaScript code;

which enable it to detect phishing webpages regardless of underlying communication protocol. In this chapter, I showed a system that can achieve a high degree of accuracy in detecting zero-hour phishing webpages without relying on third-party systems. The proposed system is resistant to the current state of the art evasion techniques employed by phishers such as using images, buying old domains.

CHAPTER 4: NLP-Based Trend Analysis of APT Techniques

4.1 Introduction

Advanced Persistent Threats (APTs) are organized, well-supported, and well-planned cyber-attacks against governments and companies with high values[5]. APT attackers use multiple attack techniques and tactics conducted meticulously to avoid detection, such that they can maintain their access to the target for a long time. They also amend their techniques and tactics over time to cope with the changes on the target networks and to further extend their footholds [6]. Such attackers cost companies and government agencies billions of dollars in financial losses annually. An exemplar of such groups is Lazarus, also known as APT38. Since 2014, this group has attempted to steal over 1.1 billion dollars from financial institutions worldwide, including the recent 81-million-dollar heist of Bangladesh's central bank [7].

Since the first reports of such sophisticated attacks by UK and USA CERTs (Computer Emergency Readiness Teams) in 2005, researchers in both industry and academia have carried out significant studies on APTs, from detection, analysis, to defense [6] [52]. A large number of technical reports on real-world APTs are now available (e.g., [53]), and such threat intelligence information is vital for defenders as it can be used to generate *actionable knowledge*, which helps the defenders to make better decisions about how to improve their defensive mechanisms against current and future APTs.

Specifically, trend analysis of attack techniques can help security professionals focus their attention on those adversarial techniques that are more common among APT groups, thus allocating their limited time and resources more economically while

maximizing their security against APTs. Not all adversarial techniques are equally important all the time. For example, an attack technique can be popular among APT attackers because of a common bad security practice or lack of appropriate defensive mechanisms on target networks that make the technique more effective. However, the popularity of an attack technique can significantly decline over time as defenders become more aware of this attack and create more effective defense mechanisms against it or because of a technology shift making the attack technique less effective. Therefore, trend analysis must be done continuously.

However, continuously analyzing a stream of threat intelligence information to derive actionable knowledge in a timely fashion poses a significant challenge due to the sheer volume of such information and its unstructured nature. For example, the APTnotes repository that I study in this paper includes 445 technical reports with more than 1.9 million words; it is infeasible to manually analyze these reports in a short period of time. Moreover, these APT reports are written by different people in a natural language (e.g. English) and they do not follow a uniform format or writing style. Therefore, there is a practical need for automated document processing systems that can derive knowledge out of such reports.

In this chapter, I describe SECCMiner, a new information retrieval system utilizing various text mining and natural language processing (NLP) approaches, to analyze a set of unstructured APT reports written in a natural language in order to automatically recognize attack techniques and tactics in those documents. It can help security professionals to promptly and accurately retrieve threat reports about APT groups that utilize specific attack techniques or tactics. Besides, by using SECCMiner, defenders can swiftly overview the attack techniques or tactics employed by a specific APT group.

To demonstrate the benefits of SECCMiner, I conduct a comprehensive analysis of the state of the art of APTs, in which I use SECCMiner to analyze 445 technical

reports of real-world APTs published since 2008. I perform a popularity/trend analysis of the common APT techniques based on these reports. Then, I present some interesting observations about APTs, such as (1) using PowerShell scripts is on the rise, (2) watering hole is used frequently alongside spear phishing to target end-users, and (3) zero-day exploits have been strongly related to Adobe Flash.

4.2 Dataset

The dataset in this work consists of 445 reports from the APTnotes repository on GitHub [53]. This repository is a collection of many public documents, white papers, and articles about APT campaigns since 2008, and it is continuously growing; the latest report I study was published at the end of 2017. The reports are written mostly by analysts from reputed vendors (such as BAE Systems, Bitdefender, Checkpoint, Cisco, Dell Secureworks, ESET, FireEye, Kaspersky, McAfee, Microsoft, Norman, Palo Alto Networks, Pandalabs, Rapid7, RSA, Sophos, Symantec, and Trend Micro) and occasionally by well-known security writers such as Brian Krebs. Therefore, I have reasonable confidence that these reports are good sources of information about representative APT techniques.

Table 4.1 shows some details about the reports in the dataset, including the number of reports, the total number of pages, and the total number of words. Given the large number of words (1,902,099), it is infeasible for a defender to manually read all these reports. Therefore, an automated tool is needed to assist a human defender. As an illustration of the kind of automated analysis that can be done, I choose two analysis goals: (1) to identify the evolution and trend of APT techniques over time, and (2) to better understand the relationship among these techniques. My intuition is that the more reports mention a technique, the more popular the technique is, and the more recent a technique is used, the more relevant (and novel) it is.

Table 4.1: Number of APT Reports in the Dataset

Year	# of APT reports	# of pages	# of words
2017	87	1,489	330,677
2016	74	1,647	305,872
2015	71	1,423	274,874
2014	108	2,026	423,597
2013	55	1,406	278,564
2012	24	594	121,108
2008-2011	26	559	158,407
Total	445	9,144	1,902,099

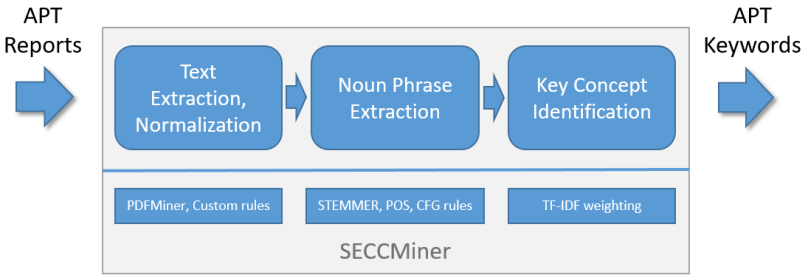


Figure 4.1: SECurity-related Concept Miner (SECCMiner) Architecture

4.3 Methodology

In this section, I present a system, SECCMiner, to automatically extract the key security concepts from APT reports. To extract such concepts, SECCMiner relies on a set of natural language processing (NLP) and information retrieval (IR) system concepts and techniques, including stemming, POS (part-of-speech) tagging, and TF-IDF weighting [54]. Figure 4.1 depicts its overall architecture. First, it converts all input PDF files to txt format. It also performs a post-processing operation on the resulted text files to correct several common issues such as the appearance of `\x1c` instead of `ffi` and fragmented text. Next, SECCMiner extracts all the unique noun phrases appeared in the corpus and then computes the TF-IDF score for each of the recognized noun phrases in each document. Finally, it records those noun phrases that have scores above a predefined threshold as *key concepts* representing each of the input documents.

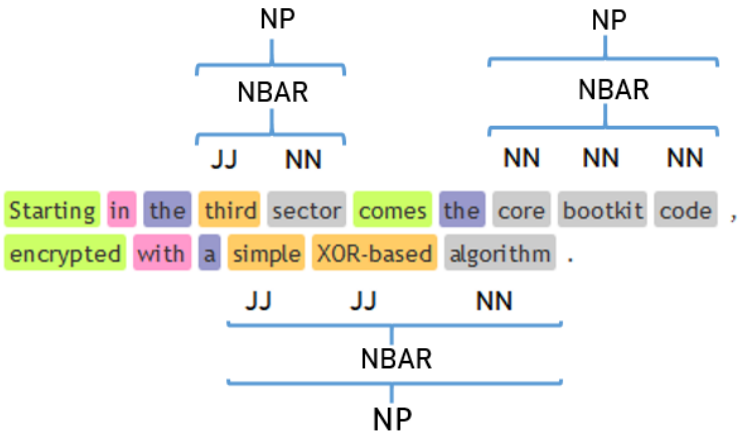


Figure 4.2: Extracting candidate phrases from a sentence using POS tagging and grammar rules

To extract the unique noun phrases in the corpus, for each document, SECCMiner first breaks the text into sentences using a regular expression rule. Then it utilizes a POS tagger to assign a part-of-speech tag like noun, verb, and pronoun to each word in the sentences. Next, it uses the CFG grammar in Code 4.1 to extract and report the noun phrases. In Code 4.1, NBAR is a collection of zero or more nouns or adjectives that ends with a noun, and NP defines a noun phrase as one NBAR or one NBAR followed by a preposition and another NBAR. Figure 4.2 depicts three candidate phrases ("third sector", "core bootkit code", and "simple XOR-based algorithm") that my tool extracts by applying the grammar rules in Code 4.1 on a sample sentence.

Code 4.1: Context-Free Grammar to Extract Noun phrases [55]

NP: {<NBAR>}{<NBAR><IN><NBAR>}

NBAR: {<NN.*|JJ>*<NN.*>}

After doing so for all documents, the system further refines the resulting phrases by filtering the phrases that do not appear in the text independently. I define an *independent noun phrase* as a noun phrase that appears at least one time in the corpus without being a part of a larger noun phrase. For example, "false positive" is an independent phrase as in several sentences it appears standalone without being

Table 4.2: Example of groups made of similar noun phrases

Group Name	Key Noun Phrases
Zero-day exploit	zero-day exploit, zero day, zero-day, unknown exploit, unknown vulnerability, 0-day
Web shell	web shell, php shell, web-based remote, web-based interface, web-based control
Video capture	phone video, recording video, video capturing, record video, security camera, cctv camera, web camera, webcam
Watering hole	watering, compromised website, compromised email, compromised smtp, compromised mail, compromised domain, compromised site
Task scheduler	task scheduler, scheduler, triggered, handler, fmgr, crontab
Mobile	android, ios, iphone
Location data	geographic location, victim location, true location, locale, keyboard layout
Flash	flash, flash player, adobe flash, flash exploit, flash code, flash file, flash object
Fileless	in-memory, in memory, fileless
Web browser	firefox, internet explorer, ie, chrome
WMI	wmi, wmic, window management
Scripts	python, bash, vbs, vb script, batch, perl, ruby
Obfuscation	obfuscation, packed, steganography

part of lager noun phrase such as “false positive rate”. However, “vector machine” is not an independent noun phrase as it is always a substring of a larger noun phrase (e.g., “support vector machine”). In this way, the system significantly reduces the number of derived meaningless noun phrases by applying the above grammar rules in the first place.

In the third step, the system calculates the TF-IDF score for all independent noun phrases appeared in each report. It filters out phrases that have a low TF-IDF score (i.e. below a predefined threshold) as they are not representative of the report content. TF-IDF is a numerical measure to capture the importance of a word or phrase in a specific document within a corpus of documents. The score is based on how often the word or phrase appears in that specific document and within the corpus. If a particular word appears only in a few documents but occurs several times in a given document, it gets a high TF-IDF score for that specific document.

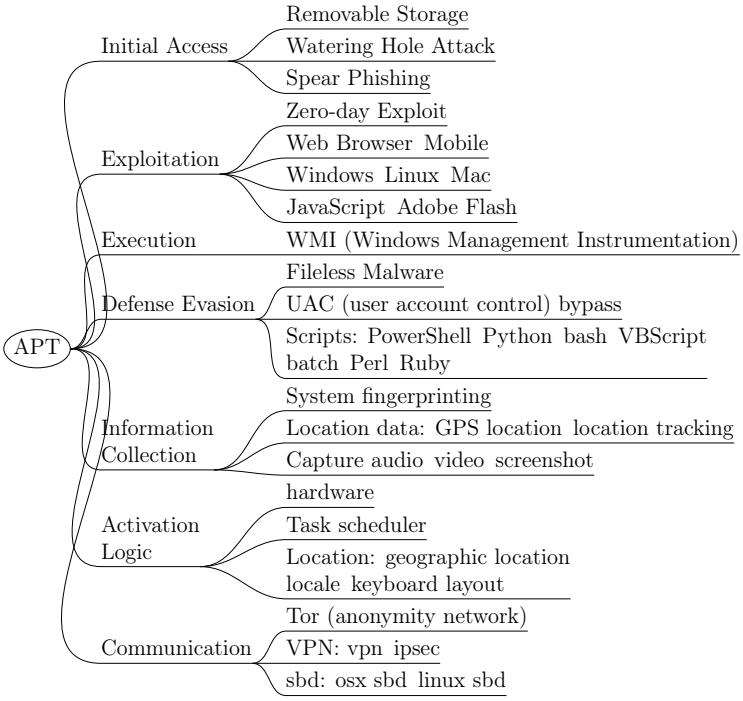


Figure 4.3: Classification of Most Common APT Techniques

I use the output of SECCMiner to develop an IR (Information Retrieval) system that enables analysts to retrieve APT reports that mention a specific technique or tactic being used by APT attackers. To do so, I define groups of independent noun phrases that represent the most common attack techniques or tactics as shown in Figure 4.3. Table 4.2 illustrates a few example groups formed from noun phrases that are conceptually related.

4.4 Evaluation of SECCMiner: Meaningful Detection of APT Techniques

I implemented a prototype of SECCMiner in Python. The Text Extraction module uses PDFminer library [56], and the Noun Phrase Extraction module leverages the NLTK part of speech tagger [57] to determine the part of speech of each word in the sentences. The prototype system analyzes the 445 documents in about 80 minutes on a PC with four CPU core (3 GHz) and 16 GB of RAM. However, it takes less than one second on average for the Information Retrieval system to find APT reports in

the dataset that mention a specific technique or tactic.

Specifically, SECCMiner reports the names of the APT notes that mention a given set of noun phrases. For example, in the following output of SECCMiner, it can be seen that the file named PWC_cloud-hopper-report-final-v4(04-03-2017).pdf contains phrases such as “wmi”, “wmic”, and “window management”, meaning that the APT may leverage WMI.

```
Noun phrases: wmi,wmic,window management
...
Year: 2017
Total reports: 3
PWC_cloud-hopper-report-final-v4(04-03-2017).pdf,
Kaspersky_Report_Shamoon_StoneDrill_final(03-06-2017).pdf,
ESET_TeleBots-Supply-chain-attacks-against-Ukraine(2017).pdf
```

With the names of APT notes provided by SECCMiner, one can associate real-world APTs with the APT techniques that they employ. Note that the association is not done manually, but facilitated by a tool like SECCMiner. Here, I give some concrete examples.

- CozyDuke/CozyBear [58] checks the presence of Anti-Virus products, such as Kaspersky, Sophos, and Comodo, by directly querying WMI, before conducting malicious activity.

- Dimnie [59] uses PowerShell scripts to download and execute its second-stage malware, and its payload can log keystrokes, take screenshots, and interact with smartcards, demonstrating several information collection techniques.

- The Trident Exploit Chain [60] uses obfuscated JavaScript to download the second-stage malware payload in the zero-day exploits produced by the NSO Group. This is an example of script based hiding technique and zero-day exploit.

- The Moonlight APT [61] used malicious scripts (e.g., VB scripts) to install additional malware, and there is a great variety of such scripts across different samples.

This APT targeted entities in the Middle East, and once successful, it can install a remote access tool called njRat.

- The TeleBots group [62] leveraged a standard backdoor that uses the Telegram Bot API in order to receive commands from, and send responses to, the C&C server. It employs heavy obfuscation of scripts, uses Tor relay to hide the C&C server, and has a ransomware component. This APT also steals passwords and Windows credentials, and uses PsExec for lateral movement.

- The Win32/Industroyer APT [63] is designed to disrupt the execution of ICS (industrial control systems) software, such as launching a denial of service attack against the Siemens SIPROTEC range. To evade detection, most of the C&C servers of Win32/Industroyer run Tor software, and its backdoor employs code obfuscation by inserting junk assembly instructions.

- The StrongPity APT [64] created watering holes to offer trojanized software installers (e.g, WinRAR and TrueCrypt), targeting mainly Italian and Belgian users. Once these malicious installers are executed, they drop malware onto the victim system to log keystrokes and steal disk content.

- An attack campaign targeting the Indian navy's submarine and warship manufacturers [65] employs a UAC (user account control) bypassing technique by hijacking the registry key $HKCU\backslash Software\backslash Classes\backslash mscfile\backslash shell\backslash open\backslash command$ (i.e., to make it points to malware), which enables malware to be silently executed in a high integrity process.

- The DarkSeoul APT [66] seeks mainly South Korean targets in several sectors, and it accomplishes this by verifying that the current locale of the victim contains "Korea." In other words, this APT has an activation logic based on locale.

- The Dustysky multi-stage malware [67] has been used by the Molerats to collect

intelligence. This APT takes screenshots, recovers saved passwords in browsers, and scans the file system for personal documents, credentials, certificates and private keys, as well as information pertaining to homeland security. It uses Windows Management Instrumentation (WMI) to get information about the operating system and check the presence of Anti-Virus. It also searches for removable storage and network drives to duplicate itself.

In all the examples above, SECCMiner offers a succinct summary (highlight) of the techniques employed by the corresponding APTs, in sets of noun phrases. This capability is very useful for a large-scale study of APT techniques across hundreds of reports, which are detailed in Section 4.5 and Section 4.6.

4.5 APT Technique Trend Analysis

Figure 4.4 shows the overall trend for 14 techniques that are more frequently mentioned than other techniques in the 2017 reports. This figure reveals several interesting findings:

- APT attackers mainly target end-users by exploiting web browsers (e.g., 33 cases in 2014 and 18 cases in 2016, detailed in Table 4.3); however, exploiting mobile phones is on the rise. Specifically, it can be seen from Table 4.3 that the number of APTs that exploited mobile devices increased from one in 2012 to 11 in 2016.

- Using PowerShell scripts is on the rise. Specifically, in 2017, 25 reports (28 percent of reports) have mentioned Powershell, which is a 625 percent increase compared with 2016 in which only 4 reports mentioned PowerShell.

- Watering hole is frequently used along with spear-phishing to target end-users. There are 18 cases of watering hole APTs in 2014, 11 cases in 2015, 11 cases in 2016, and 11 cases in 2017. Comparatively, there are 14 cases of spear phishing in 2014, 12 cases in 2015, 16 cases in 2016, and 10 in 2017.

Table 4.3: Reports of APT techniques per year

Technique\Year	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**
Exploited Browsers	7	23	33	20	18	9
Exploited Mobiles	1	4	6	7	11	4
Used Scripts	3	15	13	18	20	32
Used Obfuscation	4	6	22	12	15	14

- Various scripts have been consistently used by APTs (Table 4.3). These include Python, VBScript, Perl, batch script, and bash script. For example, Python scripts were used by the W32.Flamer's C&C servers to wipe files, free up disk space, receive stolen data, and distribute attack payload [68].

- Obfuscation has been consistently used by APTs (Table 4.3). The number of APTs that employed obfuscation has been 22 in 2014, 12 in 2015, 15 in 2016, and 14 in 2017. The concrete techniques include XOR obfuscation [69], API name obfuscation [70], PDF obfuscation [71], and steganography [72].

- Tor is used by attackers to hide their infrastructure. Example APTs include Win32/Industroyer [63] and TeleBots [62].

From Figure 4.4, one can also see the shift of the "hot" techniques per year. In 2013 the most prominent techniques were Target Browsers and Scripts, followed by Remote Desktop. In the following years till 2016, Target Browsers and Scripts continued to be the top techniques, but Remote Desktop has lost its appeal. In 2017, there has been a major shift: PowerShell emerged as a top technique while Target Browser became insignificant.

4.6 APT Technique Relationship Analysis

Figure 4.5 shows the relationship among 28 attack techniques mentioned in the report dataset. The main purpose of this study is to find out the relationship among different attack techniques: techniques that are closely related to each other should be used

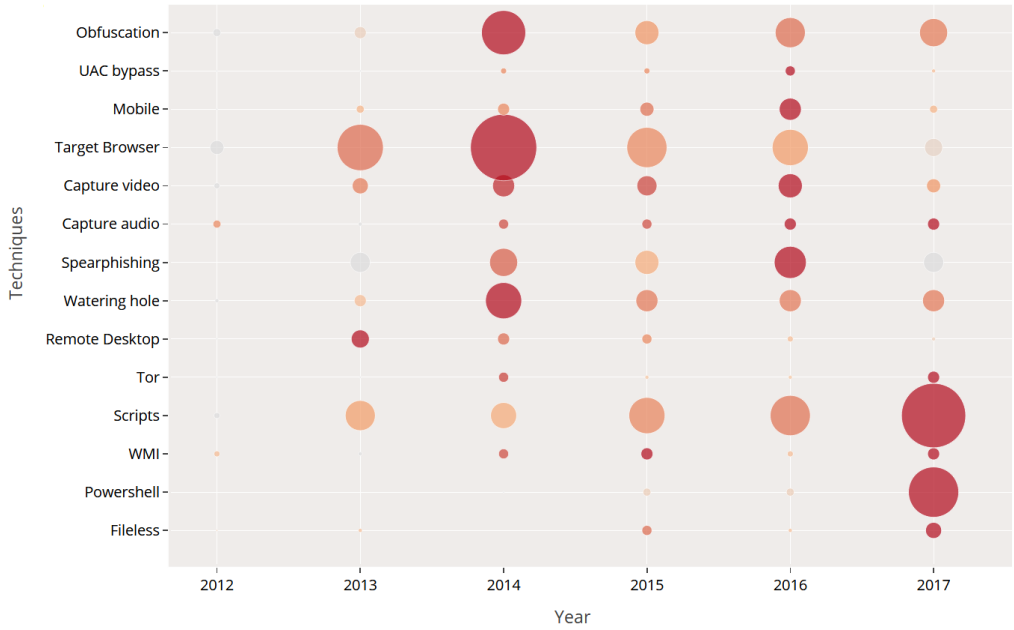


Figure 4.4: Number of reports mentioning a specific attack technique published since 2012; bigger circle means a larger number of reports.

together, and one can infer this by noting that these techniques are mentioned by the same APT report. Therefore, by counting how often each pair of techniques are mentioned together in the report dataset, one can infer how strongly they are related. Furthermore, since my goal is to discover emerging and novel APT techniques, I employ a weighted sum scheme that gives recent reports exponentially more weights. Specifically, in Figure 4.5 nodes represent APT attack techniques, an edge between two technique nodes indicates that the two techniques appear in at least one common APT reports, and the thickness of an edge represents how strongly the two techniques are related to each other. The thickness is calculated by the following formula:

$$w_{term_1,term2} = \sum_{i \in Years} |reports_i(term_1, term_2)| * 2^{i-2012} \tag{4.1}$$

In this formula, $report_i$ is a function that takes two terms (each term represents an APT technique) and returns a set of reports published in year i that mention both terms, $|s|$ is the cardinality function that returns the number of elements in a given set

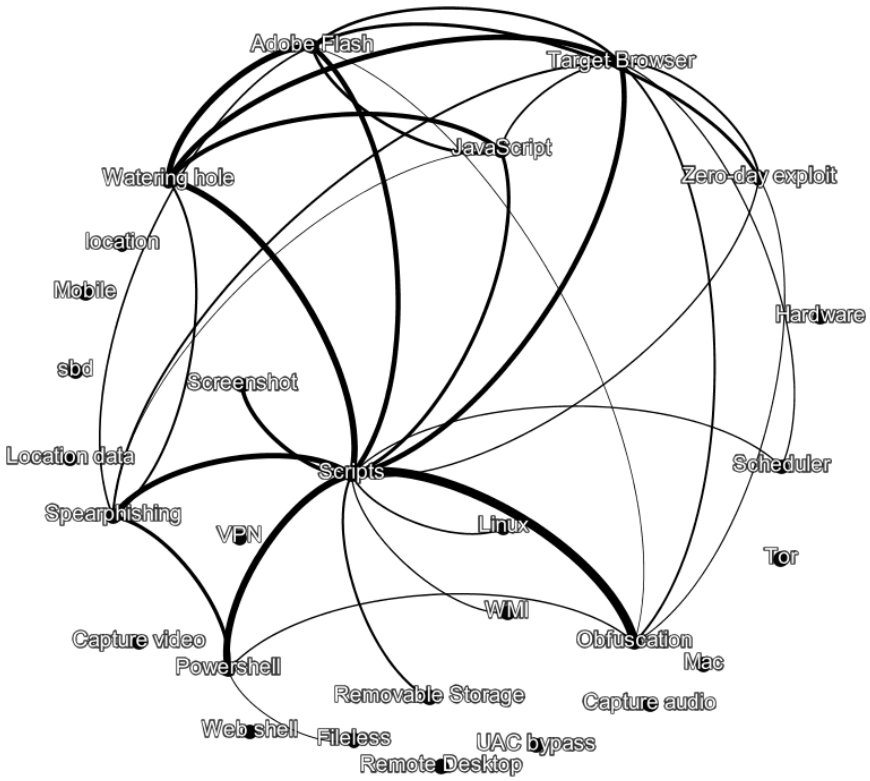


Figure 4.5: Relationship among APT Attack Techniques

s, and the weight starts from 1 in 2012 and increases exponentially over the following years. To make the graph more readable, I pruned the edges with a thickness less than 192. From Figure 4.5, several interesting observations can be drawn:

- There is a strong correlation between the Watering hole and the Target browser. Specifically, the thickness of the **{Target browser, Watering hole}** edge is 350. This makes sense because Watering hole attacks by definition are based on compromised websites that are often visited by the victims.

- There is a strong relationship between Obfuscation and Scripts (the thickness of **{Obfuscation, Scripts}** is 474), which is consistent with the observation that APTs commonly use obfuscation techniques to protect their scripts.

- Adobe flash vulnerabilities are more frequently used to target browsers in watering hole attacks than JavaScript. Specifically, the thickness of the edge **{Adobe flash,**

Target browser} is 262, while the thickness of **{JavaScript, Target browser}** is 229. Moreover, the thickness of **{Adobe flash, watering hole}** is 373, and the thickness of **{JavaScript, watering hole}** is 325.

- Zero-day exploit has the strongest relation with Adobe Flash (the edge thickness is 281) and the second strongest relation with Target Browser (the edge thickness is 242). It is also related to Removable Storage (the edge thickness is 143).

- Capture Audio and Screenshot are related (the edge thickness is 147), which is not surprising as APT attacks that collect intelligence often record audio and take screenshots.

4.7 Related Work

Chen et al. [6] conducted a comprehensive study on APT attacks. They first characterized APT attacks and compared them with traditional attacks, and next they surveyed the techniques and tactics used in APTs during each phase of intrusion kill chain introduced by Lockheed Martin [9]. They also studied four APTs in details and mapped their techniques and tactics into attack phases. In this work, instead of manually extracting such knowledge, I present a system to automatically identify techniques and tactics used by APTs. I use this tool to analyze 445 APT reports published since 2008 in order to identify trending APT techniques.

MITRE ATT&CK [73] is an attempt to create a knowledge base of known attack techniques that are used by cyber attackers. For each attack technique, they specify the kill chain phase(s) in which it can be used (i.e., for what purpose the attacker uses that specific technique), how a defender can detect the usage of such a technique, and how it can be mitigated. The goal is to create an actionable knowledge base. MITRE ATT&CK Matrix enumerates all the known techniques utilized by attackers during different phases of the attack kill chain. By analyzing the public cyber threat

intelligence reports, one can measure the occurrence frequency of these techniques and in this way, help defenders to prioritize the threats based on the commonality of techniques among APT groups.

Recently, several research works have been proposed that utilize text mining and natural language processing techniques to extract different types of information from publicly available cyber threat intelligence reports. Neuhaus et al. [74] utilized a topic modeling approach to analyze Common Vulnerabilities and Exposures (CVE) reports in order to semi-automatically study prevalent vulnerability types and identify new trends. Liao et al. [75] proposed iACE, a solution for extracting Indicators of Compromise (IOCs), such as IP addresses of C&C servers, and their contexts from blog posts in a fully automatic way. iACE utilizes a topic classifier to filter out non-IOC blog posts. To extract IOCs, it relies on a fixed set of context terms commonly used to describe IOCs in technical reports and a set of regular expressions. Sabottke et al. [76] examined Twitter data for early detection of exploits that are being used in the wild (i.e. before details about a vulnerability are officially announced by the vendor). Based on their experimentation, they introduced a set of techniques utilizing supervised machine learning for detecting such exploits. Husari et al. [77] presented TTPDrill, which is a system that uses information-theoretic approach to identify TTPs in cyber threat reports. It can recognize TTPs that are described in MITRE ATT&CK dataset. SECCMiner does not rely on any external datasets including MITRE ATT&CK and can detect new techniques that are not seen in such datasets.

CHAPTER 5: Extracting IoCs from Social Media

5.1 Introduction

Sharing information about recent cybersecurity incidents can considerably reduce the existing knowledge gap between defenders and attackers as the techniques, tactics, and procedures used in one cyber attack can be reused to attack other organizations with similar environments. In addition, the system and network infrastructures used by attackers to target a victim are commonly reused in other attacks. In recent years, many security companies have emerged that are specialized in collecting and characterizing cyber incidents and sharing extracted intelligence with other companies or the public to prevent such reuses. For example, abuse.ch have several trackers for tracking command and control (C&C) servers of famous botnets such as Zeus. Network defenders can consume their Zeus tracker feed to block network traffics destined to Zeus C&C servers; thus neutralizing Zeus bots.

Despite the achievements of such cyber threat intelligence companies, they still suffer from the following two problems. First, their coverage of cyber threats is far less than ideal, which means clients need to aggregate from many of such companies to cover a good percentage of ongoing threats. Second, there is a significant delay between the time of receiving threat signals to the time of identifying and publishing the threat reports. To address these two problems, I introduce IoCMiner, a framework to extract cyber threat intelligence, in particular IoCs, from public information-sharing platforms, such as social media, discussion forums, and text sharing websites, where a large number of individuals and companies share their findings of ongoing cyber attacks. It relies on concepts and techniques borrowed from graph-mining, text mining,

and machine learning.

IoCMiner is a lightweight online framework with no dependency on external systems. As a result, it can scale well with the amount of information published on popular data-sharing platforms. It is online (*i.e.*, it processes the input data in near real-time) as the threat landscape is continually evolving; which, on average, cause threat information to expire shortly after being reported by cybersecurity professionals on these platforms. Due to the sheer amount of published information on data sharing platforms, instead of directly examining published information, IoCMiner continuously attempts to identify reliable cyber-threat intelligence sources on the target platforms. Only contents published by such sources is analyzed to extract cyber-threat intelligence.

The current prototype of IoCMiner supports Twitter and Pastebin; however, it can be extended to support other data-sharing platforms. Due to the nearly ubiquitous adoption of social media platforms such as Twitter, IoCMiner can significantly improve the coverage problem and complement data available through traditional channels. My experimentation with IoCMiner on Twitter also confirms that a large volume of fresh threat intelligence information is shared on this platform by cybersecurity professionals.

5.2 Problem Statement

This chapter addresses the problem of extracting cyber threat intelligence, in particular indicators of compromise (IoCs), from data shared on information sharing web platforms. To be more specific, given the stream of data published on such a platform, I want to: 1) recognize reliable cyber threat-related data, 2) extract atomic IoCs from them, and 3) aggregate the resulted atomic IoCs to obtain a more comprehensive picture of the associated threat.

Without loss of generality, I focus on: 1) Twitter, a micro-blogging social network,

as it is one of the most famous social networks with more than 330 million active users sharing over 500 million tweets on a variety of topics per day, and 2) Pastebin, a text sharing platform, with more than 95 million text documents.

Due to the immense number of tweets published every day, and the existence of a significant imbalance between the number of security and non-security related tweets, any viable online solution must adopt a strategy to avoid processing each and every tweet published on the platform. As mentioned earlier more than 500 million tweets are tweeted every day, which means on average 3500 tweets per minute are streamed to subscribed applications. It is worth noting that Twitter only publishes one percent of the whole tweets through its streaming API. Without any filtration, a system must be able to process 3,500 tweets per minute. Almost all of these tweets are not related to cyber threats; thus most of the resources and time will be wasted on tweets that are not relevant to its goal. More importantly, because of this significant imbalance between threat and not-threat tweets, even a minuscule inaccuracy in differentiating between the two can make the system useless as it may render too many false positives.

To address these issues, instead of examining all tweets, IoCMiner monitors tweets posted by a set of users who have shown interest in tracking cyber threats and publishing their IoCs. In this way, IoCMiner receives a significantly lower number of tweets required to examine per minute; thus can perform more process-intensive operations. Moreover, the ratio of tweets containing IoCs increases drastically. This enables me to employ a classifier to identify IoCs without getting too many false positives or negatives. Most of the related research work on social media solely focus on finding influential users, the ones who have significant effects on other users' behavior. Despite its importance and relevance to this work, the primary goal of this work is not to identify such users. Instead, the focus is on identifying tweets that contain valuable information about ongoing cyber attacks. This information may or may not be originated or disseminated by influential users on social networks.

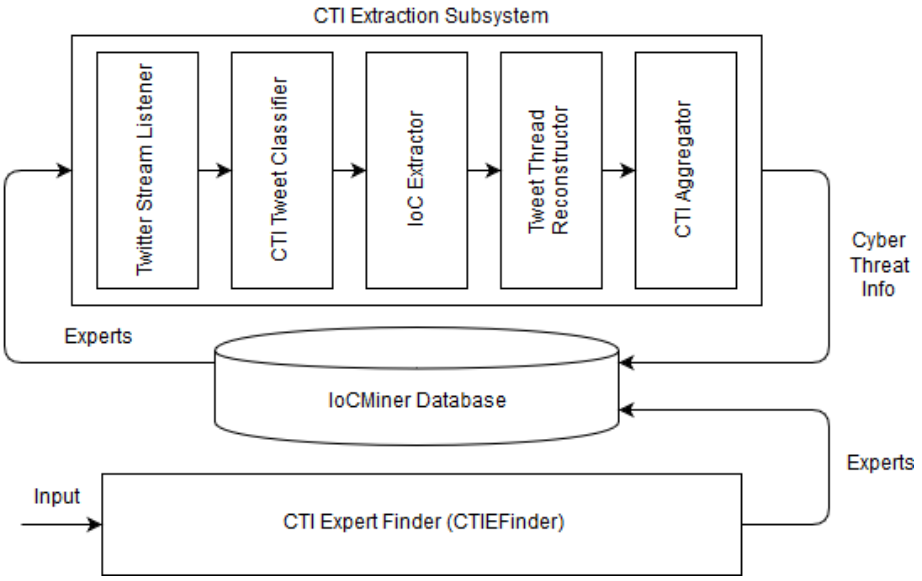


Figure 5.1: IoCMiner Architecture

5.3 IoCMiner Architecture

Figure 5.1 depicts the overall architecture of IoCMiner, which is consist of two separate subsystems, namely the CTI Expert Finder (CTIEFinder) and CTI Extraction subsystems. CTIEFinder, periodically, examines potential users on Twitter to identify CTI experts who consistently publish high-quality information about ongoing attacks. CTI Extraction subsystem continuously monitors tweet stream containing tweets posted or shared by identified CTI experts and extract useful cyber threat information.

To discern cyber threat intelligence experts, who are willing to share their knowledge, from other users, CTIFinder analyzes users' tweeting history and measures the amount of cyber threat information that each of them has already shared with the public. The underlying assumption is that the probability of publishing IoCs by users with good track records is significantly higher than others. However, one must also ensure that the identified users are credible sources of information. To evaluate the credibility of such users, CTIEFinder considers a range of features extracted from their tweet history, their relationships with other users, and also the lists that they

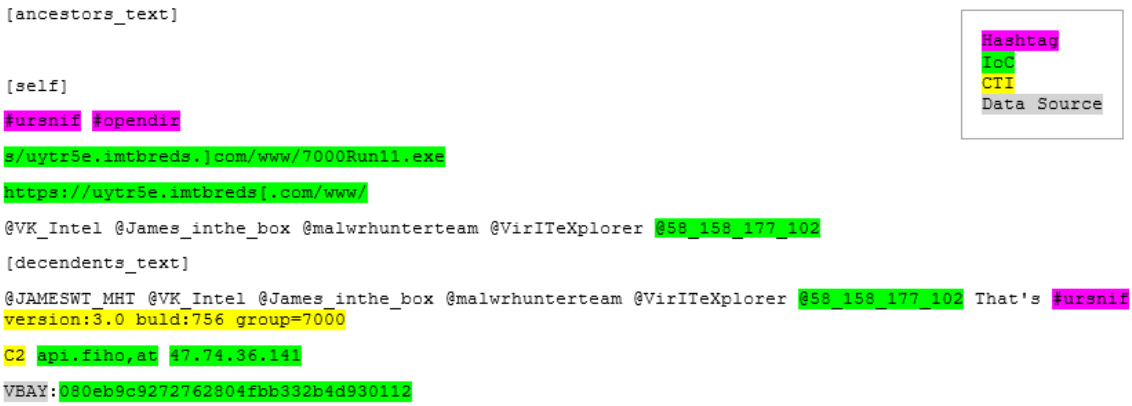


Figure 5.2: Example of a CTI tweet thread

are a member of. The underlying assumption is that credible users tend to follow credible users and also only like or retweet tweets that seem to be valid. Moreover, users who create lists about a specific topic are knowledgeable about the topic and tend to only add topical experts who publish useful information to their lists.

After identifying CTI experts, the next step is to monitor their tweet activities and analyze them to extract useful CTI information. CTI extraction subsystem continuously collects live tweets posted by the experts. Since not all of the collected tweets are related to cyberattacks, CTI extraction subsystem employs a custom classifier, called CTI Tweet Classifier, to separate cyber threat tweets from the non-CTI ones. Tweets are then passed to IoC Extractor to mark IoCs within the CTI-labeled tweets. IoC Extractor uses a set of regular expression rules to recognize IoCs. All tweets are then passed to Tweet Thread Reconstructor which constructs tweets threads by analyzing their `in_reply_to_status_id` fields; a sample tweet thread can be seen in Figure 5.2. Finally, CTI Aggregator links CTI tweet threads, the ones containing at least one IoC, with each other based on shared IoCs and hashtags.

5.4 Identifying Cyber-Threat Intelligence Expert

In this section, I explain the internal of the Cyber Threat Intelligence Expert Finder, CTIEFinder, subsystem. This subsystem is employed by IoCMiner to discover cyber threat intelligence experts who consistently publishes IoCs related to ongoing threats on Twitter. It exploits the relationships between users and lists on Twitter to identify potential CTI experts. It, then, further prune the list of candidates by examining their tweet histories.

On Twitter, users can create lists to categorize users into groups based on some criteria. For example, a user can create a list for tracking cybersecurity news and add cybersecurity journalists to this list. Logically, users only add twitter handles that publish useful information related to the topic of their interest. As mentioned by other researchers [78], these user-defined lists are valuable resources for identifying topical experts. IoCMiner also relies on user-defined lists to find cyber-threat intelligence experts. It exploits the relationship between users and lists to identify potential topical experts and then further analyzes their tweet histories to ensure their expertise.

Each user-defined list has a number of members, who are added by the list owner. In addition, a user can be a member of multiple lists. The many-to-many relationships between lists and users can be modeled by a bipartite graph as shown in Figure 5.3. It worth noting that not all lists related to a specific topic has the same quality or specificity. In general, the more selective lists are the better ones. CTIEFinder relies on several metrics to measure the quality of user-defined lists in a specific topic; in this work cyber-threat intelligence. These metrics measure the relevancy, popularity, comprehensiveness of a list in addition to the credibility of its owner.

- Relevancy score - is a composite indicator that measures the degree of which a given list is relevant to a topic of interest, in this work cyber threat intelligence.

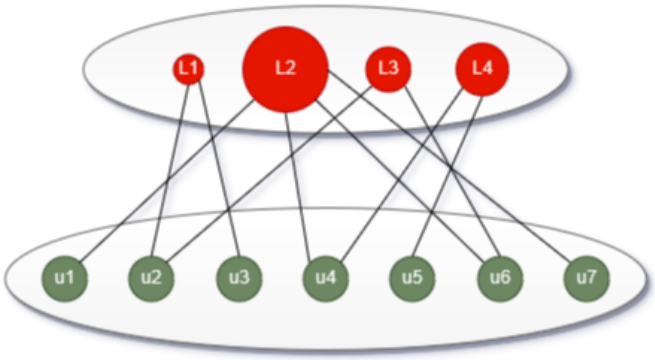


Figure 5.3: Relationship between users and lists is modeled as a weighted bipartite graph

As mentioned by other researchers [78], the name and description of a list are valuable semantic cues that can be exploited to uncover its topic. Formula 5.1 is defined to measure the relevancy of a list to CTI topic based on its name and description.

Let *Lists* be a set containing all the lists to be examined, and *list* be one of these lists. In addition, Let t_{list} represents the concatenation of *list* name and description. Moreover, let G be a list containing sets of keyphrases. In current IoCMiner prototype, I manually defined two sets of keyphrases: specific and generic keyphrase sets (as defined in Table 5.1). Specific keyphrases are the ones that are closely related to the topic of interest (*i.e.*, CTI). Generic keyphrases are the words that are related to the domain in which the topic resides (*i.e.*, cybersecurity).

Let *relevancy_nd* be a function to compute the relevancy score of *list* based on its name and description. It is defined as:

$$relevancy\_nd(list) = \sum_{i=0}^{|G|} w_{G_i} * |match(t_{list}, G_i)| \tag{5.1}$$

Where *match* is a function that takes t_{list} and G_i, and returns the number of times keyphrases in G_i appeared in t_{list}. Moreover, w_{G_i} is the weight assigned

Table 5.1: Keyphrase sets in IoCMiner. Note .? means zero or one character

Specific keywords	ioc
	malware
	indicator.?of.?compromise
	threat.?hunt
	phishing.?hunt
	phish.?hunt
	threat.?int
	threat.?research
	ransomware
	mal.?doc
Generic keywords	info.?sec
	cyber.?sec
	security

to G_i; adjustable by the IoCMiner operator.

It is worth noting that the name and description of a list are very short and they may not entirely reflect the content of the list. To further check the relevancy of a list to the topic of interest, I define *relevency_hist* function to measure the relevancy of a list based on its published statuses (*i.e.*, tweet history). In this function, the textual contents of the last N tweets posted in the input list are examined to see whether they contain any IoC. It is also crucial to consider the number of times a specific IoC appeared in various lists as the ones reported in many lists are less attractive than the ones reported in a few lists.

Let *relevency_hist* be a function to calculate the relevancy of *list* to the topic of interest. *relevency_hist* is defined as:

$$relevency\_hist(list) = \sum_{i \in IoC_{list}} \frac{1}{|\{l \in Lists, i \in IoC_l\}|} \tag{5.2}$$

Where IoC_{list} is a set of all IoCs appeared in $list$ and $\{l \in Lists, i \in IoC_l\}$ is a set of all lists containing IoC i.

- Popularity score - measures how much a list received attention from the community. Users can subscribe to lists that they like in order to see their timeliness. My intuition is that better quality lists attract more subscribers. To calculate this metric, I consider the number of subscribers of a list:

$$popularity\_score(list) = subscriber\_count(list) \tag{5.3}$$

- Completeness - measures the coverage of lists in the terms of the number of experts they are enlisted.

$$member\_score(list) = \frac{member\_count(list)}{log_2(member\_count(list))} \tag{5.4}$$

- Owner credibility - measures how credible a user is. To calculate this metric, I consider the number of follower and friends of the list owner. It is defined as:

$$cred\_score(list) = log_2(\frac{|owner\_followers(list)| + |owner\_friends(list)|}{|owner\_friends(list)|}) \tag{5.5}$$

A multiplicative scoring approach is used to combine the described metrics to calculate the overall score for a list. To be more specific, the overall score of a list is computed with the following formula:

$$overall\_score(list) = \prod_{k=0}^{|Scores(list)|} (\frac{score_k(list)}{avg(\{score_k(l) : l \in Lists\})})^{w_{Score_k}} \tag{5.6}$$

Where $Scores(list) = \{relevancy\_nd(list),\ relevency\_hist(list),\ popularity\_$

$score(list)$, $member\_score(list)$, $cred\_score(list)\}$, $score_k(list)$ is kth item in $Scores$ $(list)$, and w_{score_k} is the weight for the kth score in Scores list. By changing w_{score_k}, an analyst can increase or decrease the importance of score k in $Scores(list)$.

In formula 5.6, all the metric scores are first normalized by dividing them with the average value of these metrics. By doing so, the measurement unit for the metrics are not important and one can compare the metrics. Then, they are multiplied with each other to get the final score for each list. It is worth noting that in this formula, metric scores below the average will be in the range [0, 1) and scores above the average will be between 1 and positive infinity. As a result, scores below average have an adverse on the magnitude of the final list score, and scores above average have a positive contribution. Resulted list scores are, then, used as the weight of lists in the bipartite graph.

After ranking the lists, CTIEFinder takes the top N lists. The expectation is that users in these lists are topical experts. However, the overall score does not directly show the expertise level of each individual user in the list; it only shows the collective effort of its users. After finding such lists, the goal is to select the most valuable experts among the members of the lists. Intuitively, users who are listed in many lists with high weights are considered better in the eyes of the community. CTIEFinder uses formula 5.7 to compute the credibility of users based on the lists that they were added to, which indicate the amount of belief that the list owners have in these users.

$$user\_list\_score(u_i) = \sum_{l \in neighbor\_lists(u_i)} overall\_score(l) \tag{5.7}$$

Where $neighbor\_lists(u_i)$ are the nodes (*i.e.*, list) that are directly connected with u_i in the bipartite graph. In other words, it represents the lists that u_i is a member of.

CTIEFinder, then, select top $5 * k$ users based on the calculated $user\_list\_score$

scores. However, without examining the tweet activities of users, one cannot be confident that the users will publish IoCs in the future. Both the quantity and quality of information disseminated by users are important. In terms of quantity, I want to identify experts that are highly active in posting information regarding a topic domain. In terms of quality, I also want to get fresh and accurate data. The focus is to identify those experts that generate new credible data.

The expectation is that the publishing history of a user is a good indicative of their behavior in the near future. If they published lots of IoCs in the past, they will do in the future. As a result, CTIFinder counts the number of relevant posts that a user published in the past. In the current prototype of CTIFinder, the most recent 400 tweets of a user are considered. In general, recent history is a better predictive of the future than far back history. Thus, the importance of counts must decay as one goes far back in history. To this end, CTIFinder uses a polynomial function to assign weights for each day in the history as shown in formula 5.8.

$$user\_ioc\_score(u_i) = \sum_{d=0}^{365} \frac{ioc\_count(u_i, d)}{(d+1)^{\frac{1}{3}}} \tag{5.8}$$

ioc_count functions returns the numbers of IoCs published by u_i in d days before today. The final score is then calculated by multiplying $user\_list\_score(u_i)$ with $user\_ioc\_score(u_i)$. The top k users will then be selected by CTIEFinder as CTI experts who publish cyber threat information, in special IoCs.

5.5 Classifying Tweet Streams

In this section, I describe the CTI Tweet Classifier module, which is responsible for tagging tweets with CTI or Non-CTI labels, in IoCMiner. The input of this module is a sequence of tweets collected by Tweet Stream Listener. By using this classifier, IoCMiner can narrow down the IoC extraction to only CTI tweets, which increases

the degree of accuracy. Classifying tweets also helps to improve the output of CTI aggregator module as it only considers tweet threads containing CTI tweets and attempt to join them.

CTI Tweet Classifier, first, tokenize the input tweets, turning them into bags of words. Next, each bag is filtered by removing stop words and then stemming the words. It, then, counts the number of each word in a tweet and make a vector from these counts. The resulted vector is ultimately used as features to help classification. Finally, it performs machine classification on the vectorized tweets to identify the one that contains IoCs. CTI Tweet Classifier utilizes RandomForest algorithm to recognize tweets containing IoCs. To be more specific, it constructs a classifier by training the Random Forest classifier on a set of already-labeled tweets. The features in this classifier are the words that constitute the tweet contents, number of hashtags, and the number of mentions in the training dataset.

5.6 Evaluation

In this section, I evaluate the effectiveness of IoCMiner in extracting fresh IoCs from tweet streams. I narrow down the focus to atomic IOCs and primarily on the quality of data that can be obtained from tweets published on Twitter. To evaluate IoCMiner, I seek to answer the following research questions: first, whether security professionals commonly publish IOCs on Twitter. Second, whether the atomic IOCs extracted from tweets are fresh; this is important due to the ever-evolving threat landscape. Third, whether data published on Twitter is first-hand; in other words, whether twitterers publish first-hand data, unpublished by traditional channels, or they are just rehashing the existing knowledge known to the cybersecurity community. The input to IoCMiner is a list of manually-selected security professionals that publish IOCs on Twitter. The output is a growing list of IOCs published on Twitter. In the rest of this section, I describe how the current implementation of IoCMiner works.

5.6.1 Observation list

To bootstrap IoCMiner, one must input a seed list of exemplar cybersecurity experts who publish IoCs on Twitter. I created a seed list by manually searching different keywords related to cyber threats, including malware names such as Emotet and GandCrab, to identify security professionals who publish cyber threat-related data. I, further, considered the number of followers, the company that they work for, and the number of security-related tweets that they had published. In this way, 62 well-known threat intelligence experts were identified and added to the seed list; which then was fed to IoCMiner as input for experimentation.

For each seed user, IoCMiner fetches the metadata information, such as name, description, member count, subscriber count, and owner info, of all lists that the user has been added to; this process resulted in retrieving metadata information for 5,851 lists in my experiment. Based on collected information, IoCMiner ranks the retrieved lists and picked the top 1,000 ones based on their metadata. For each of these selected lists, IoCMiner further retrieves 1,000 recent tweets and user information of all of its members. In this way, IoCMiner selected 118,847 users. Next, it adjusted the scores of each list based on the new information and the formula presented in section 5.4. IoCMiner, then, computes the score for each of the users in these lists based on the list scores that they were a member of and, then, selected the 5,000 users with the highest scores. Next, for each of these users, it retrieved 400 recent tweets and used that information to readjust the scores for the top 5,000 users. The top 1,000 users then constituted the observation list. In addition to the IoCMiner selected top 1000 cyber threat intelligence experts, I selected 1,000 users randomly from the remaining users (117,847) and added them to the observation list to represent the baseline.

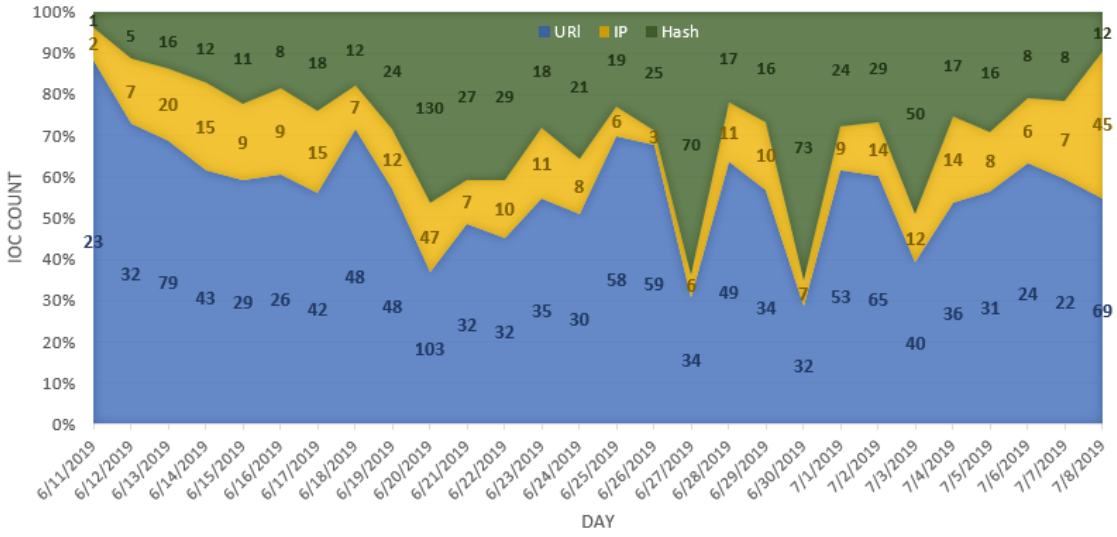


Figure 5.4: Extracted IoCs, namely URLs, IP addresses, and hashes (total: 2261)

5.6.2 IOC extractor

I have simplified the problem of extracting IoCs to extracting malicious URLs, IP addresses, and hashes reported by security professional on Twitter. The current implementation of IOC extractor module in IoCMiner relies on a set of regular expression rules to identify such IoCs. I observed that security professionals do not post malicious URLs and IP addresses in a well-formatted form to prevent unwary users from accidentally clicking these links and infecting themselves. For example, instead of starting URLs with http, they may start malicious URLs with hxxp or / (slash). During my experimentation, I identified several such patterns and created regular expression rules to match them. Figure 5.4 shows the number of IoCs harvested by IoCMiner between June 11th to July 8th. During this period, 1208 of the IoCs were URLs. To test the freshness of these URLs, I checked them with Google Safe Browsing List (SBL) on the time of extraction. On average, less than 10 percents of extracted URLs marked as malicious by Google SBL as shown in Figure 5.5.

I rescanned the undetected URLs every day for one week after their extractions to see when these URLs will be added to SBL. Furthermore, I checked these URLs with

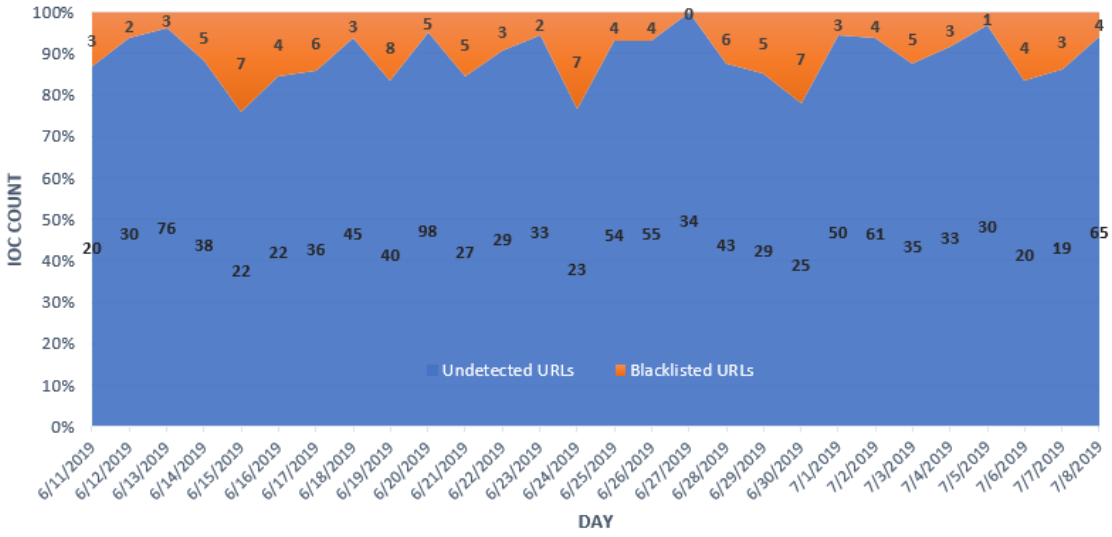


Figure 5.5: Malicious URL collected over three weeks by IOMiner. 116 out of 1208 URLs were blacklisted by Google SB

VirusTotal after seven days to see whether they were identified by any of 60 blacklists on this platform and if yes, when was the scan time. Figure 5.6 shows the results of rescanning of IoCs detected between June 11th and July 8th. About 10 percents of the URLs were detected by either Google SBL or VirusTotal on the same day of their discovery. After one week from discovery time, the percent of detected URLs has increased to 26 percent. This result suggests that the data extracted from Twitter is fresh and can complement existing blacklists such as Google SBL.

Several researchers, such as [79] and [80], have mentioned that despite the usefulness of considering hashtags for determining the topic of tweets, one should not rely on them as a large percentage of tweets do not contain any tags. Although several researchers, such as [79] and [80], have reported low usage of hashtags by Twitter users, in my preliminary experimentation, I observed that more than 52 percent of tweets containing IoCs have at least one tag. This suggests that tags can be used to determine the topics of tweets in the CTI domain.

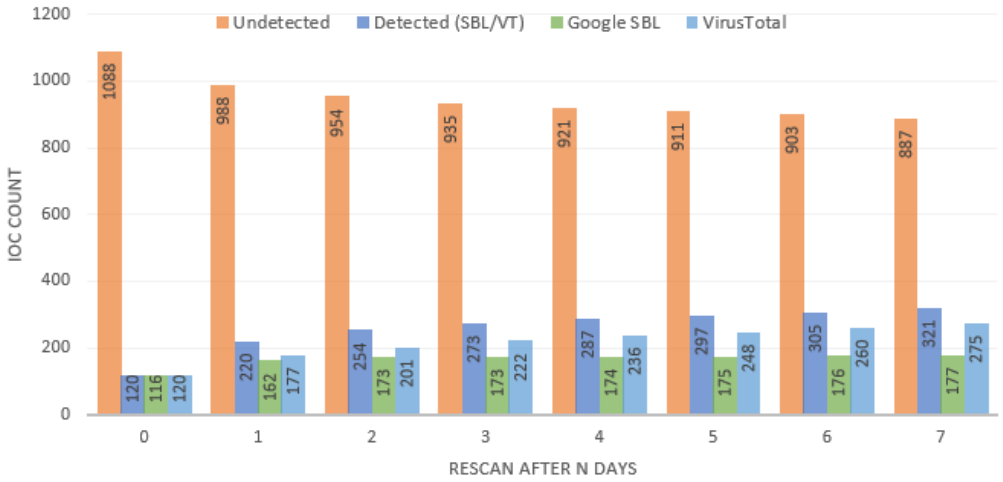


Figure 5.6: Daily rescanning of URLs harvested between June 11th and July 8th with Google SBL and VirusTotal for one week after collection

5.6.3 CTI Tweet Classifier

To evaluate CTI Tweet Classifier, 75 accounts that are known to share cybersecurity were monitored for a week in April 2018 and their tweets were collected. 2,300 tweets were collected in this way. It is worth noting that not all of the collected tweets were security-related as the users also tweeted about other matters such as their personal life. The collected tweets were, then, manually labeled as having IoCs or not having IoCs. During this process, the tweets with content not entirely in English were also discarded. To evaluate the classifier, first, it is trained with a set of 200 random tweets and tested on a group of 143 random tweets, all from the same week. The machine classification of this group yielded an accuracy of 97.2%. In the second experiment, the classifier is trained on 200 tweets, but this time, I ensured that 20% of the tweets were labeled as containing an IoC. When testing the 143 tweets once more, the accuracy was, again, around 97%.

5.7 Related Work

As mentioned in the previous section, my goal in this chapter is to identify credible users on Twitter that post about cyber threats and to extract IoCs from their tweets. In recent years, many research works have been published on identifying credible sources of information and also influential users in terms of propagation of information on social media such as Twitter. In this part, I briefly review some of the most notable research works in this area.

Weng *et al.* [80] observed that 72.4 percent of Twitter users follow more than 80 percent of their followers which they attributed this reciprocity to the phenomenon of homophily seen in many other social networks [81]. This phenomenon suggests that people follow their followers because of the similarity in their topics of interest. Based on this observation, they proposed TwitterRank, which is an extension of PageRank algorithm to measure the influence of users on Twitter. It considers both the topical similarity of users and link structure between them to rank influential users on a specific topic.

Montangero *et al.* [82] proposed a method to identify the most influential twitterers on a specific topic, where the topic is denoted by a hashtag. They proposed the following three indicators to measure the influence of a user on a given topic: followers influence, retweet influence, and favorite influence. To identify influential users on a topic, first, for each candidate user (*i.e.*, the one that tweeted about the topic), they compute these indicators. Then, for each indicator, they create a list of k user with the highest scores. The users appeared in these lists are divided into three group: 1) Highly influential users, those who appeared in all the three lists 2) Influential users, those who appeared in 2 of them, and 3)Potential influential, those who only appeared in one of the top k lists.

Castillo *et al.* [83] proposed a method to determine whether a set of tweets related to a topic are credible. The method relies on a number of features extracted from

users' posting, and reposting behaviors, from the content of the tweets, and from external references mentioned in the tweets. The results show that the method can discern between credible and not credible tweets with the precision and recall in the range of 70% to 80%. They concluded that credible news is propagated through users that have previously posted a large number of tweets, originate at a single or a few users in the network, and have many retweets.

Alrubaian *et al.* , in [84], proposed a new approach to determine the credibility of information sources (*i.e.*, users) on a specific topic in Twitter, which can be utilized to detect malicious users conducting various malicious activities such as propagation of false or derogatory information on this network. In addition to considering the popularity of twitterers, they consider how sentimental the users are regarding a particular topic to calculate their credibility. To do so, they calculate the ratio of positive tweets to all tweets published by a user. In their experiments, the could achieve 93.4% accuracy at locating users who can be considered credible on a predefined topic.

Ghosh *et al.* , in [78], proposed Cognos, a crowdsourcing search engine for identifying experts on Twitter. To find domain experts, it relies on features extracted from the name and description of Twitter lists. To be more specific, in Cognos, the keywords in the title and description of lists are assigned to their members. These assigned keywords to users are then used to find a topical expert. In Cognos, all lists are treated as equal regardless of their quality and specificity. In our work, we consider other features to assign weights to lists based on their quality and trustworthiness. Moreover, not only we consider user-list relationships, but also we incorporate several user-specific features to identify topical experts better.

In [85], authors proposed a binary classifier based on Support Vector Machine (SVM) to classify spammers and non-spammers on twitter. The proposed classifier relies on 1) 39 features extracted from the textual content of the user's tweets such as the average number of words of each tweet and number of hashtags on each tweet,

and 2) 23 features that capture the user behavior in terms of the posting frequency, influence, and social interactions on the Twitter network. The proposed classifier achieved 70% true positive and 96% true negative on a large dataset containing 54 million users.

Sabottke *et al.* [76] examined data published on Twitter for the possibilities of early detection of exploits that are being used in the wild (i.e. before detailed information about a vulnerability officially announced by its vendor). Based on their experimentation, they introduce a set of techniques utilizing supervised machine learning for detecting such exploits.

CHAPTER 6: ShadowMove: A Stealthy Lateral Movement Strategy

6.1 Introduction

Advanced Persistent Threats (APTs) are sophisticated, well-planned, and multistep cyber attacks against high profile targets such as government agencies or large enterprises. Such attacks are conducted by groups of well-resourced knowledgeable attackers and cost companies and government agencies billions of dollars in financial losses per year. An exemplar of APT groups is Lazarus, also known as APT38. Since 2014, this group has attempted to steal over 1.1 billion dollars from financial institutions worldwide, including the recent 81-million-dollar heist of Bangladesh's central bank [7].

APT attackers commonly use spearphishing or watering hole attacks to find a foothold within target networks. Once they entered the target networks, they cautiously use the compromised systems as stepping stones to reach other systems until they get access to the critical systems, such as file server containing confidential documents, buried deep inside the networks; this incremental movement toward the critical systems is called *lateral movement*.

Lateral movement can be achieved in a number of ways. Attackers can exploit vulnerabilities in network services, such as SMB or RDP, to laterally move across networks. However, due to advances in defense mechanisms, finding such vulnerabilities and successfully exploiting them without being detected has become increasingly hard. Alternatively, attackers can harvest user credentials from compromised systems and reuse such credentials to do the lateral movement (e.g., credential dumping [86], pass-the-hash, or pass-the-ticket [87, 88, 89, 90, 91]). However, this approach requires

new network connections to be created and thus can be detected by network-level defenses if the new connection deviates from the normal communication pattern among legitimate systems [92, 93, 94]. Using another approach, adversaries can employ hijacking attacks that modify a legitimate client in order to reuse its connection for lateral movement (e.g., by patching an SSH client to communicate with the SSH server without knowing the password [95]). However, existing attacks are application- and protocol-specific and require process injection. Such attacks are hard to implement and prone to detection as existing host-based defensive solution recognize various process injection techniques.

In this chapter, I present a novel lateral movement strategy, called ShadowMove, which enables APT attackers to move stealthily among the systems in enterprise networks without being discovered by existing host-level and network-level defensive mechanisms as demonstrated in Section 6.6. Attackers are assumed to deliberately avoid exploiting vulnerabilities in remote services during their operations to reduce the chance of being exposed by intrusion detection systems (IDSes). In the presented attack scenario, an attacker passively observes communication dynamics of the compromised systems to gradually construct their model of normal behaviors in the target network and utilizes this model to choose the next victim system. Moreover, to make the attack even stealthier, the attacker restricts themselves to only *reuse the established connections*. Many application protocols such as WinRM (Windows Remote Management) and FTP allow users to perform some operations on the remote server. The attacker injects their own commands in the command streams of such application protocols to achieve their goal. For example, the attacker can execute a program remotely by injecting commands in an established WinRM session (Section 6.5.2), or they can inspect the file system on the remote system by injecting FTP commands on an established FTP connection (Section 6.5.1).

ShadowMove does not inject any code into benign client processes or alter their

execution paths in order to inject fabricated commands. Instead, it employs a novel technique to secretly duplicate sockets owned by legitimate clients and injects commands through such stolen sockets (Section 6.4.5). This technique exploits by-design vulnerabilities in modern commodity operating systems such as Windows and Linux to duplicate sockets; thus hijacking connections. These vulnerabilities arise from the differences between the goals of two conflicting but yet fundamental operating system requirements, namely resource sharing and process isolation. By leveraging this technique, no new connection is needed to be created and also no new authentication will be performed as the injected commands are interpreted in the context of already established sessions; this means that the attacker does not need to pass any authentication.

In this work, I also show how an attacker can implement such an attack on a typical enterprise network. To this end, a prototype system is developed that can hijack existing TCP connections established by an FTP client (Section 6.5.1), a WinRM client (Section 6.5.2), and a Microsoft SQL client (Section 6.5.3) running under the same user account as the prototype system on a Windows system. I also present a first-order logic that an attacker can utilize to systematically plan for lateral movement by hijacking available connections. In this way, the attacker can reach the critical systems significantly stealthier than existing attack scenarios. I discuss the technical challenges on how attackers can inject their packets that conform to the protocol running over an established TCP connection and be acceptable to the server on the other end of the connection.

My contributions in this chapter can be summarized as follows:

- A new class of lateral movements is presented which is completely undetectable by the existing network and host-based defensive solutions including IDS, Antivirus, and EDR (Endpoint Detection and Response) systems.

- A novel socket duplication technique is introduced that enables attackers to reuse

connections established by other processes on a compromised system. I, then, develop a lateral movement framework on top of this technique.

- The feasibility of the presented idea is demonstrated by building a prototype system that successfully hijacks FTP, WinRM, and TDS (used by Microsoft SQL Server) connections for lateral movements.

- I experimentally confirm that the prototype systems can evade the detection of five top-notch anti-virus products (McAfee, Norton, Webroot, Bitdefender, and Windows Defender), the Snort IDS, and two emerging Endpoint Detection and Response systems: CrowdStrike Falcon Prevent and Cisco AMP. It is important to point out that CrowdStrike Falcon Prevent is known to detect lateral movements [94].

6.2 Technical Background

A socket is a transport endpoint, used to send or receive data. Winsock 2 (Windows socket API) uses a layered service provider architecture: from top to bottom, we have the Winsock DLL/multiplexer layer (e.g., WS2_32.dll) and the service providers layer (e.g., mswsock.dll) [96]. Windows supports sharing of sockets between processes. However, the Windows Socket interface does not implement any type of access control. Instead, it relies on the processes involved to coordinate their activities on the shared socket, in order to avoid anything undesirable [2]. Obviously, this is problematic because the support for socket sharing opens the door for socket hijacking by untrusted processes. As demonstrated later in this chapter, the socket opened by a benign process such as Microsoft SQL Server Management Studio can be secretly used by a malicious process to upload and execute its malware on a remote SQL server.

6.3 Underlying Problem

Process isolation and resource sharing are conflicting requirements. Attackers can misuse resource sharing capabilities in modern operating systems (OSes) to bypass the process isolation mechanisms enforced in such operating systems. For example, Microsoft Windows supports several dangerous Inter-Process Communication (IPC) mechanisms, such as enumerating processes, creating remote threads, and modifying other processes' address space [97]. Conventional OSes do not completely isolate access to resources, such as network connections, owned by a process due to offering built-in mechanisms that enable shared access to such resources. One such mechanism is duplicating handles.

There are legitimate reasons for handle duplication capability. As a result, general-purposes OSes, such as Windows and Linux, have some mechanisms for that. On Windows, NtDuplicateObject allows one process to get a copy of a handle in another process' context. On Linux, dup, dup2, dup3, and fcntl(..., F_DUPFD, ...) create a copy of a given file descriptor, and the old and new file descriptors refer to the same open file description and thus share file offset and file status flags. Linux file descriptor duplication is more restricted because the old and new file descriptors are in the context of the same process or closely-related processes (i.e., a child process gets duplicated descriptors of the parent process implicitly).

Another underlying problem that enables ShadowMove is the lack of message origin integrity checks in most standard application protocols such as FTP and TDS (for MS SQL). As a result, endpoints cannot verify the origins of the messages to ensure that the messages are not interleaved by malicious actors. An attacker who duplicated a shared socket can interject a request in between requests of a client and mislead the server to think the original client sent it; thus processing the request and replying an appropriate response.

6.4 ShadowMove Architecture and Design

6.4.1 Threat Model

An attacker is assumed that has established a foothold on a victim system under a normal user's privilege, and she wants to make a lateral movement towards the critical asset(s). The attacker has to run malware to achieve this. It is also assumed that the victim process whose TCP connection is going to be hijacked is not aware of the malware process.

6.4.2 Demonstration Scenario

I use an Employee Self-service Application of a company as an example. This is a typical multi-tier enterprise application that can be accessed from a browser. Below is the description of the components of such a system:

- Employee desktop computers, which run the web client. Some employees are IT personnel at the same time, and they need to occasionally push content to the application server, so their computers have file copying tools (such as FTP) installed.
- Application server, which runs many applications such as payroll, stock, health insurance, retirement plan, and travel.
- Database server, which stores personnel information such as DOB, SSN, contact info, and salary, and is accessed by the application server.

In this example, the attacker landed on an employee desktop (via spearphishing), and this employee happens to be an IT personnel. The critical assets that the attacker goes after is the information of all employees stored in the database server. Therefore, the attacker needs to move from the desktop to the application server then to the database server. Moreover, the attacker needs to have some tool persist on the database server in order to get daily reports about any updates to employee

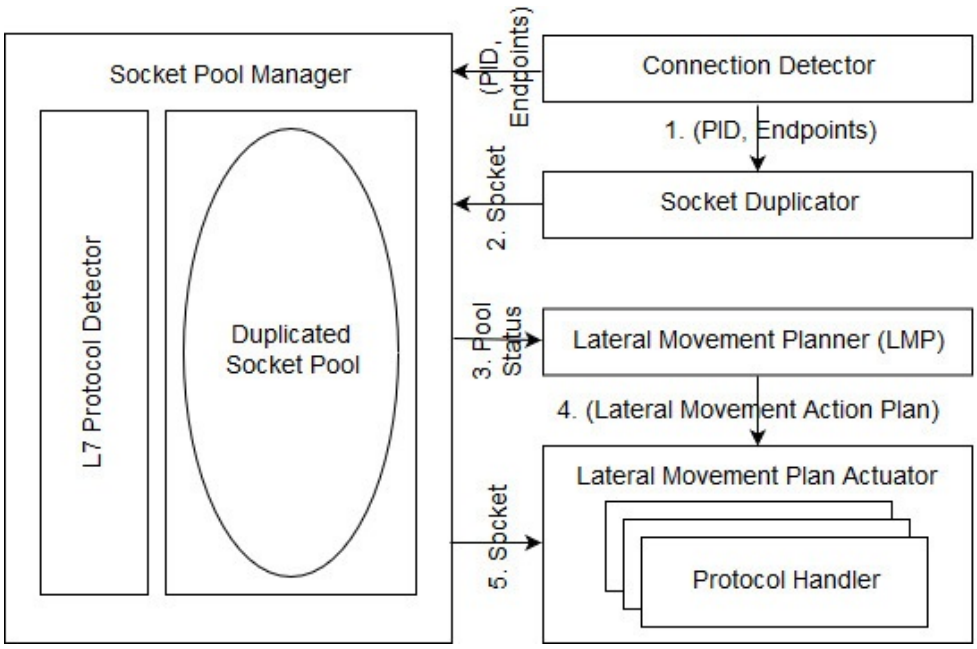


Figure 6.1: ShadowMove Architecture

records. They can leverage the FTP connection (see Section 6.5.1) to copy a piece of malware to the application server under the Startup Folder [98]. Based on how Windows is designed, the malware will be executed on the application server when a user logs in [98, 99]. When that happens the malware can leverage the database connection (such as Microsoft SQL discussed in Section 6.5.3) to copy and launch further malware on the database server.

6.4.3 Overall Architecture

Figure 6.1 depicts the overall architecture of ShadowMove, which has a modular architecture consisting of five major modules: Connection Detector, Socket Duplicator, Lateral Movement Planner (LMP), Plan Actuator, and Socket Pool Manager.

Connection Detector (CD) module (Section 6.4.4) is responsible for detecting newly-established TCP connections that can be exploited for lateral movement. This kind of passive observation allows an attacker to gradually construct a model of normal network communication pattern in the victim environment. To this end, CD constantly

monitors the TCP connection tables by calling `GetTcpTable2` and `GetTcp6Table2` API functions. Upon detection of a new TCP connection, CD checks whether the ShadowMove process has enough permission to open the owner process of the corresponding socket and if so, it requests the Socket Duplicator to duplicate the socket.

Socket Duplicator (Section 6.4.5) duplicate sockets owned by other processes in a way that can be reused by ShadowMove. Duplicated sockets with additional contextual information, such as the PID of the owner process and the IP:port of the remote endpoint, is passed to the Socket Pool Manager.

The Socket Pool Manager (Section 6.4.8) combines a few methods to determine the service type (or application protocol) supported by each duplicated socket in the pool. It also maintains the liveness of the duplicated sockets by removing socket objects that are no longer usable (e.g., the TCP connection has been closed for some reason).

Periodically, ShadowMove Lateral Movement Planner (Section 6.4.6) create a lateral movement plan based on the available sockets in the Duplicated Socket pool. LMP has prior knowledge about the types of capabilities that each supported application protocol can provide. For example, FTP can be used to bidirectionally transfer files between the local system to a remote server, and TDS protocol (Section 6.5.3) additionally allow an attacker to execute commands on a remote server. Based on these capabilities and the current socket pool state, the planner creates a lateral movement action plan. This action plan specifies the socket that must be used, and the type of action that must be carried out, and the payload.

Finally, the Plan Actuators (Section 6.4.7) execute individual steps in a lateral movement plan, such as transferring a file to the remote server, by sending packets to and/or receiving packets from the given sockets.

6.4.4 ShadowMove Connection Detector

Two approaches exist for detecting and tracking TCP connections. First, the TCP connection table, which contains information about all TCP connections, can be polled periodically by calling Win32 APIs such as `GetTcpTable2` and `GetTcp6Table2`, and compare the returned table with the result of the previous call. This approach is used by command-line tools such as TCPView. A second approach is an event-driven approach in which an event handler is registered for the creation or tear-down of connections. In Windows OS, one can get information about connection state changes by creating a WMI (Windows Management Instrumentation) filter and registering a WMI event consumer [100]. However, registering a WMI event consumer requires administrative privilege.

As a result, in ShadowMove, the first approach is chosen. By calling `GetTcpTable2` and `GetTcp6Table2`, the Connection Detector can get basic information about a TCP connection, such as connection state, local IP address, local port, remote IP address, remote port, and the process ID of the owner of the TCP connection [101]. From the process ID, it further gets the process name. When the Connection Detector observes a connection state change from non-ESTABLISHED to ESTABLISHED, it notifies the Socket Duplicator (Section 6.4.5) about the new TCP connection. On the other hand, when it observes a connection state change from ESTABLISHED to non-ESTABLISHED, it notifies the Socket Pool Manager to remove a duplicated socket from the pool because the associated TCP connection becomes unusable.

The Connection Detector does some simple filtering of TCP connections before it notifies the Socket Duplicator or the Socket Pool Manager. Specifically, it checks whether the ShadowMove process has enough permission to open the owner process of a TCP connection with PROCESS_DUP_HANDLE access flag, and it skips those connections for which the ShadowMove process does not have enough permission. Other than that the Connection Detector passes along the owner PID, owner process

name, local IP address, local port, remote IP address, remote port, and the connection state.

6.4.5 ShadowMove Socket Duplicator

The Socket Duplicator duplicates sockets associated with new TCP connections when it receives a notification from the Connection Detector (Section 6.4.4). The underlying idea of the presented approach is to duplicate the socket inside the target process and to use the resulted socket to secretly access the established TCP connection. In Windows, one can call `DuplicateHandle` API to duplicate different types of handles from a remote process. However, as mentioned in `DuplicateHandle` documentation [102], this function cannot be used to duplicate sockets.

Although Windows offers an API named `WSADuplicateSocket` to duplicate a socket, one cannot directly use this function as it requires cooperation between the processes. Table 6.1 illustrates the typical scenario of using this function. In this table, the source process is the one that created the original socket, and the destination process is the one that wants to reuse that socket. In a nutshell, first, the source process obtains the process id of the destination process (through step 1-3). Then, it calls `WSADuplicateSocket` to get a special WSAPROTOCOL_INFO structure. This info structure is given to the destination process via inter-process communication (IPC) mechanism. The destination process passes the info structure to `WSASocket` to reconstruct the socket on its side. The main challenge in this approach (i.e., using `WSADuplicateSocket`) is that both processes must cooperate with each other to duplicate a socket, which is not the case in the presented scenario where the attacker wants to duplicate a socket from an unwary victim process. One way to address this issue is to inject code into the victim process to implement the missing steps due to a lack of cooperation. However, existing defense mechanisms such as Windows Defender ATP flag usages of common process injection techniques [103], which makes

the solution less attractive.

I devised a novel technique, by using Windows APIs in an unconventional way, that enables an attacker process to duplicate a socket from a target process without requiring its cooperation. Table 6.2 depicts the steps that the attacker process performs to duplicate a socket from a target process, assuming it knows the process ID of the target, thanks to real-time connection detection (Section 6.4.4). First, it opens the target process by using `OpenProcess` to enumerate all of the open handles in the target. The attacker process only seeks for file handles with the name of `\device\afd` (step 3-6). During this operation, the attacker process duplicates all file handles as it is required for reading the name of a handle. I discovered that the attacker process could treat these duplicated `afd` handles as sockets.

To locate the exact socket corresponding to a TCP connection, the attacker process obtains the remote IP address and remote port to which the `afd` handle of socket is connected (by invoking `getpeername`) and compare them with the information passed in by the Connection Detector. If there is a match, the attacker process passes the `afd` handle to WSADuplicateSocketW to obtain the information necessary for duplication of the original sockets. After obtaining the protocol info structure, the attacker process calls the `WSASocketW` function to duplicate the sockets. These sockets are then saved in the Duplicated Socket Pool together with context information such as the owner PID, the owner process name, local IP address, local port, remote IP address, and remote port; and these sockets can be used to receive and send data through existing TCP connections already established by the target process.

Table 6.1: A typical usage of WSADuplicateSocket [2]

Source Process	IPC	Destination Process
1) WSASocket, WSAConnect 2) Request target process identifier	==>	
		3) Receive process identifier request and respond
4) Receive process identifier 5) Call WSADuplicateSocket to get a special WSAPROTOCOL_INFO structure 6) Send WSAPROTOCOL_INFO structure to target	<==	
	==>	7) Receive WSAPROTOCOL_INFO structure 8) Call WSASocket to create shared socket descriptor 9) Use shared socket for data exchange
10) closesocket	<==	

It is also noteworthy that in Windows, the TCP connection tables for IPv4/6 only contain information about the original socket descriptors not the duplicated ones and the owner PID of a socket descriptor will never change even after the termination of the owner process. This means that conventional tools, such as `netstat`, that rely on Windows APIs to retrieve TCP connection tables cannot be used to detect whether a connection is duplicated and if so which processes are the duplicators.

One should note that in the proposed attack, the socket is shared between the original client and the attacker, which can cause a race condition in receiving and sending data from the remote endpoint. The one who calls the `recv` function first will get the data from the input buffer and the one who call `send` function first will send the data to the server. This may result in reading a partial response from the server or sending a garbled request to the server. To prevent such a possibility, the attacker can simply pause the client process temporarily while she is sending/receiving data from the server.

6.4.5.1 The race between the benign application and the attack

To suspend a process, one can enumerate all the threads belonging to the target process using `CreateToolhelp32Snapshot`, and then call `SuspendThread` for each of them. However, this may cause a problem when threads are resumed. Therefore, I use another approach, in which undocumented `NtSuspendProcess/NtResumeProcess` functions in ntdll.dll are called to suspend and resume processes. If the period of the suspension is long, the user may notice that and hence become suspicious. More complex synchronization strategy can be adopted to only suspend a process for a short period of time, thus preventing user suspicion.

Table 6.2: ShadowMovePOC - Socket Duplication Given Owner Process ID, Remote IP, and Remote Port Number

Step	Description	kernel/ntdll functions
1	Open the owner process with PROCESS_DUP_HANDLE	OpenProcess(PROCESS_DUP_HANDLE, , pid)
2	Foreach handle with type 0x24 (file)	NtQuerySystemInformation(SystemHandleInformation, ...)
3	Duplicate the handle	NtDuplicateObject
4	Retrieve its names	NtQueryObject(ObjectNameInformation)
5	Skip if the name is not \device\afd	
6	Obtain remote IP and remote port number	getpeername(handle, ...)
7	Skip if remote IP and port do not match the input parameters	
8	Call WSADuplicateSocketW to get a special WSAPROTOCOL_INFO structure	WSADuplicateSocketW(handle, ...)
9	Create a duplicate socket	WSASocketW(WSAPROTOCOL_INFO, ...)
10	Use the socket	recv(), send()

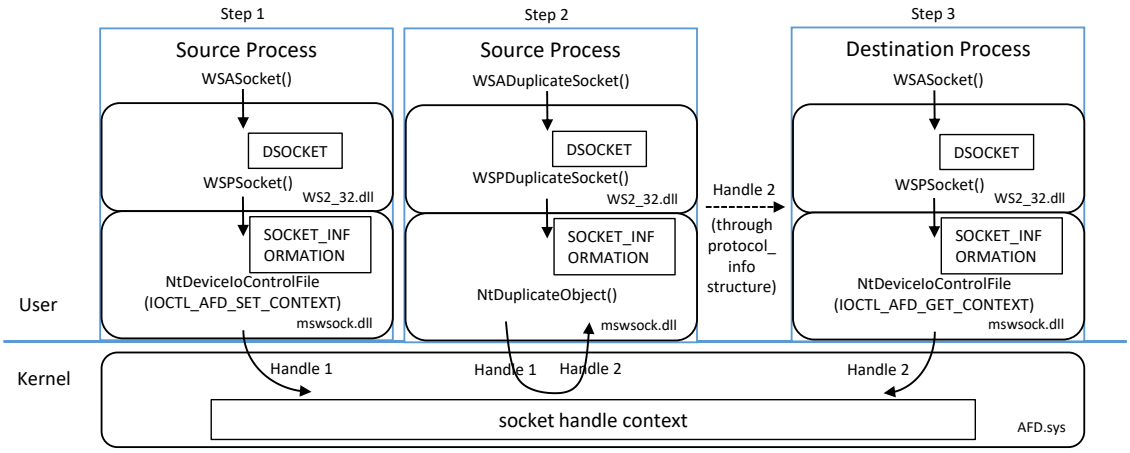


Figure 6.2: Winsock Duplication

6.4.5.2 Deep Dive into Socket Duplication

To understand why ShadowMove's socket duplication works, it is necessary to first understand *socket context*. The winsock2 libraries maintain socket context for each socket handle in a number of data structures at different layers ([104] and Figure 6.2). Inside WS2_32.dll, there is a hash table called `sm_context_table`, which maps a socket handle to a `DSOCKET` object that stores information about the socket such as the process, service provider, and catalog item. At the next layer, mswsock.dll (a service provider), there is another hash table called `SockContextTable`, which maps a socket handle to a `SOCKET_INFORMATION` object, which stores information such as socket state, reference count, address family, socket handle, local address, and remote address. Every user-level operation on the socket, such as `connect`, `send`, and `recv`, has to refer to and may change the socket context (e.g., the remote address and the reference count). Moreover, such context information including the hash tables is maintained for each process. The kernel side of socket functionality, which is the Ancillary Function Driver or AFD.sys, also maintains socket context information (e.g., local address and remote address), which is necessary for the kernel driver to eventually construct network packets. When a new socket is created, the owning process sets its context in the kernel by invoking a system call; when a socket is to

be shared, its context information is retrieved from the kernel by the target process.

What happens during normal socket sharing via WSADuplicateSocket (Table 6.1). The normal socket sharing in Windows involves three steps, as illustrated in Figure 6.2. When the source process invokes `WSASocket` to create a new socket, it does three things [104]: (1) calling `NtCreateFile` to get a socket handle (e.g., Handle 1), (2) creating a new `SOCKET_INFORMATION` object for Handle 1, and (3) calling `NtDeviceIoControlFile` to set the kernel side context information of Handle 1. Next, when the source process invokes `WSADuplicateSocket` to share Handle 1 with the destination process, it first creates a duplicate of Handle 1 (e.g., Handle 2), and then puts Handle 2 in the `dwProviderReserved` field of a `WSAPROTOCOL_INFO` structure to be shared with the destination process [105]. When the destination process invokes `WSASocket` with the `WSAPROTOCOL_INFO` structure as one parameter, `WSASocket` extracts Handle 2 from the `dwProviderReserved` field and uses it to call `NtDeviceIoControlFile` to get the kernel side context information; once this is done, it uses the obtained information to construct an `SOCKET_INFORMATION` object for Handle 2, which makes Handle 2 a functional socket handle.

What happens during ShadowMove's socket hijacking (Table 6.2). Using the same scenario above, ShadowMove can secretly share the socket with handle `Handle 1` without the cooperation of the source process. ShadowMove also uses a combination of `WSADuplicateSocket` and `WSASocket`, but it does one more step as preparation: it first creates a duplicate of Handle 1 by calling `NtDuplicateObject`; this is necessary because Handle 1 is in the address space of the source process so ShadowMove cannot directly operate on it, but ShadowMove can directly use the duplicate handle (e.g., Handle 1') because it is created in the context of ShadowMove. Next, ShadowMove invokes `WSADuplicateSocket` to share Handle 1' with itself. As a result, Handle 2 is created and put in the `dwProviderReserved` field of the `WSAPROTOCOL_INFO` structure. Finally, ShadowMove invokes `WSASocket` with the

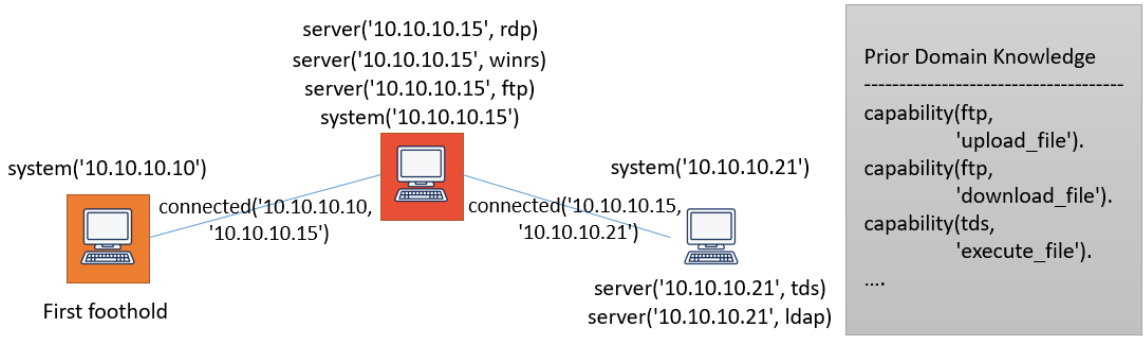


Figure 6.3: ShadowMove Knowledge Base is constructed gradually as it moves across the target network

`WSAPROTOCOL_INFO` structure as one parameter, in order to make Handle 2 a functional socket handle. Here since `WSADuplicateSocket` and `WSASocket` are invoked in the same process (i.e., ShadowMove), there is no need to pass `WSAPROTOCOL_INFO` structure across processes.

6.4.6 ShadowMove Lateral Movement Planner

In this section, I describe how attackers can use predicate logic to plan their lateral movements through a target network to access a specific system while keeping themselves under the radar. Let's assume that attackers aim to reach a specific system and perform some actions on this system. The lateral movements are achieved by using the hijacked sockets in the Duplicated Socket Pool.

To be as stealthy as possible, the attackers restrict themselves to reuse established connections to neighboring systems of the systems that they have already compromised. In this way, they can ensure that those intrusion detection systems that raise alarms for unexpected connections cannot detect their operation. Moreover, by doing so, the attackers can bypass the authentication phase required for establishing a new connection. For example, if the attacker hijacks an established FTP connection, she can send `STOR` command to the server to upload a file to the server, without authentication.

Table 6.3: ShadowMove Predicates to model target networks

Predicate	Definition
system	system(ip_addr)
connected	connected(src_ip, dst_ip)
server	server(ip_addr, service).
capability	capability(service, action).

I formulate the attack planning problem in Prolog. The attacker uses the predicates defined in Table 6.3 to specify the systems that she has knowledge about including their capabilities, and the interconnections between them: A *system* predicate defines the IP address of a host that can be leveraged for lateral movement. A *connected* predicate defines connections between two systems. A *server* predicate defines the type of services available on a system.

Figure 6.3 illustrates the ShadowMove knowledge base consisting of a set of *facts* that representing a network with two systems in addition to the initial compromised system. The *facts* in the knowledge base define a system that runs FTP, RDP , WINRS and another system that runs MS SQL Server. Moreover, 10.10.10.10 is connected to 10.10.10.15, and 10.10.10.15 is connected to 10.10.10.21. For each protocol, I also use the *capability* predicate to specify the actions that the attacker can do if she hijacks the corresponding TCP connection. ShadowMove uses the following rules to determine whether a specific operation can be carried out on a remote system Y from a given system X.

```
remoteOperation( X, Y, Action, Route):-
  connected(X,Y), server(Y, Z),
  capability(Z, Action), Route=[X|[Y]].

remoteOperation( X, Y, Action, Route):-
  connected(X,Z),
  server(Z, Service), capability(Service, Action),
  remoteOperation( Z, Y, Action, R), Route=[X| R].
```

By using remoteOperation, the attacker can check whether there exists a path between two servers that would allow her to perform a specific operation such as execute or upload a file. For example, the attacker can execute the following query:

```
remoteOperation('10.10.10.10', '10.10.10.21', 'upload_file', R).
```

which returns [′10.10.10.10′,′10.10.10.15′,′10.10.10.21′]. This result means that an attacker on 10.10.10.10 can upload an arbitrary file to 10.10.10.21 by first uploading it to 10.10.10.15 via FTP hijacking (e.g., FTPShadowMove), then moving from 10.10.10.15 to 10.10.10.21 because they are connected, 10.10.10.21 is running the LDAP service, which supports file uploading. Here, it is assumed that an LDAP connection between 10.10.10.15 and 10.10.10.21 can be hijacked.

The attacker also can mix the actions to perform more complex tasks such as executing malicious file (owned by attacker) on a specific machine. To achieve that, the attacker need to upload the file on that specific machine and execute the file. In order to do so, the attacker can query:

```
remoteOperation('10.10.10.10', '10.10.10.21', 'upload_file', R),
remoteOperation('10.10.10.10', '10.10.10.21', 'execute', R).
```

Several predicates used by the Lateral Movement Planner are derived from the *Duplicated Socket Pool* data structure. Specifically, local IP addresses and remote IP addresses can be mapped to *system* predicates, the local IP address and remote IP address pair associated with each duplicated socket can be used to create a *connected* predicate, and the remote IP address and the service type pair associated with each duplicated socket can be used to create a *server* predicate. On the other hand, *capability* predicates are derived from domain knowledge and implemented by lateral movement actuators.

6.4.7 Lateral Movement Actuator

Lateral Movement Actuator (LMA) is a module manager containing several actuation modules. Each of these modules is responsible to handle one protocol such as TDS (Section 6.5.3). LMA module can act both passively and actively. In the passive mode, the module only read from a socket by passing MSG_PEEK flag to recv API call. In this way, the input buffer is not emptied the original process can read the content. In the active mode, the module read from the socket without passing the MSG_PEEK flag; hence the recv call consumes the data in the input buffer. In this state, the module also writes to the socket out buffer to send crafted messages.

In some protocols, ShadowMove needs to learn a few secrets before being able to craft valid messages (e.g., shellID for WinRM in Section 6.5.2). In these scenarios, the actuator modules start in the passive mode, sniffing the receiving messages to learn such secret values. After learning all of such required data element, the actuator module can switch itself to active mode and start communicating with the remote endpoint. Moreover, in some case, the goal might be to do surveillance instead of lateral movement; in such a case, one can just launch these modules in passive mode to collect application-level messages. It is noteworthy that LMA module can only read incoming messages; it cannot read the outgoing messages as to the best of my knowledge there is no such API that allows one to read from the socket output buffer. In the current prototype, LMA has three actuation modules for FTP, TDS, and WinRS protocols. However, one can add a new protocol to LMA by implementing an interface called IPModule.

6.4.8 Socket Pool Manager

Socket Pool Manager module watches socket objects in the *Duplicated Socket Pool* and performs the following two operations: first, it determines the application protocol that is used by the legitimate client/server applications. Second, it retires socket

objects that are no longer usable for some reason (e.g., the connection is terminated by the remote server). It maintains a data structure for each socket in the pool, which is a tuple: <connection state, local IP address, local port, remote IP address, remote port, service type, owner PID, owner process name>. Most of these fields are passed in by the Socket Duplicator, except for service type (or protocol).

Socket Pool Manager has a sub-module called *Layer 7 Protocol Detector* that combines a few methods to determine the service type of each duplicated socket. First, the *Layer 7 Protocol Detector* guesses from the destination port because many services run behind well-known default ports [106], e.g., the default port number for FTP is 21. Second, it guesses from the owner processes if they are well-known client-side tools for some services, e.g., ssms.exe or the Microsoft SQL Server Management Studio is a client of SQL server. Finally, if the port number and the owner process information are not sufficient for a reliable guess, it passively sniffs the network traffic by calling the *recv* API on each socket and setting the MSG_PEEK flag. Then it analyzes the received payload to recognize the application-level protocol, leveraging existing protocol analysis techniques such as automatic protocol detection feature in Suricata [107].

The Socket Pool Manager also handles notifications from the Connection Detector to remove sockets whose connections are closed. It uses information passed in by the Connection Detector (such as owner PID, owner process name, local IP address, local port, remote IP address, and remote port) to locate the socket in the Duplicated Socket Pool. It also checks whether the connections are still alive by calling recv function on a regular basis on the sockets in the pool (with MSG_Peek flag) and removes sockets that are no longer usable from the pool.

It is noteworthy that, in Windows, closing a socket does not always entail in TCP connection termination handshake. The termination handshake occurs only when the last socket descriptor is closed. As a result, the connections will remain open even

if the owner processes close their sockets. However, a TCP connection may be not usable because of several reasons such as network failure, remote process crash, or connection inactivity timeout. To prevent connection inactivity timeout to occur, the Socket Pool Manager set *SO_KEEPALIVE* flag for all duplicated sockets using *setsockopt* API function; by doing so, keep-alive packets will be sent through these connections automatically.

6.5 Prototypes for ShadowMove Actuators

6.5.1 FTPShadowMove: Hijacking FTP Sessions

In this section, I describe the prototype system that I developed to hijack established FTP connections, which allows an attacker to download and upload files to a remote FTP server without authentication. As I discussed earlier (Section 6.4.5), in Microsoft Windows, an attacker can duplicate a socket handle created by another peer process, owned by the same user, without requiring special privilege (i.e., requiring UAC). By doing so, the attacker shares the established connection between the client and the server; without following the required authentication steps.

In FTP protocol, a client uses one TCP connection to send their commands to a server and receive the corresponding responses from the server; this connection is called *command channel*. The client also uses another TCP connection to send or receive data such as file contents; this connection is called *data channel*. A client can open multiple data channels for a given command channel. Authentication is required only for establishing the command channel, which means one does not need to re-authenticate herself for creating a new data channel. An attacker who hijacked the command channel can send a request to the server to open a new data channel for herself, thus avoiding any collision with the client contents that are being transferred on existing data channels. However, the attacker still should adopt a strategy to prevent a race condition in the shared command channel.

A FTP client can request for creating a new data channel in two ways: active FTP and passive FTP. In the active FTP, the client sends `Port` command to the server specifying the port that server needs to connect back to establish the connection. In the passive FTP, the client sends PASV command to the server, asking the server to listen to a port that client can connect in order to create a new data channel. In a nutshell, the difference between these two modes is with respect to who initiates the new TCP connection: server in active mode and client in passive mode are supposed to connect to the port specified by client and server, respectively. In the prototype, I implemented the passive FTP for demonstration. However, active FTP can also be implemented with negligible effort.

In passive FTP, the client sends PASV command to the server, and the server responds back by giving the information about the endpoint, including IP address and port, that the client must connect to in order to create a new data channel. The PASV is documented in RFC-959.

To evaluate ShadowMove, a vsftpd server is deployed on a Linux-based VPS (Virtual Private Server) hosted on the Internet. As a client, the `ftp` command and `Windows Explorer` were used to connect to the configured server. The anonymous login is blocked on the server and client needs to send a valid username and password to connect to it. As it can be seen in a demo video uploaded to [108] and the top half of Figure 6.4, the client exchanges several messages with the server in order to login to the server.

After the authentication phase, the client can send a variety of commands to do different things, such as changing the working directory, getting the list of files, downloading a file, and uploading a file. At this point, an attacker can hijack the FTP connection to send commands that she likes. To demonstrate this, I developed a tool called FTPShadowMove, which sends several commands to upload a binary file to a specific directory on the server. The specific commands (such as `CWD /files/`) and

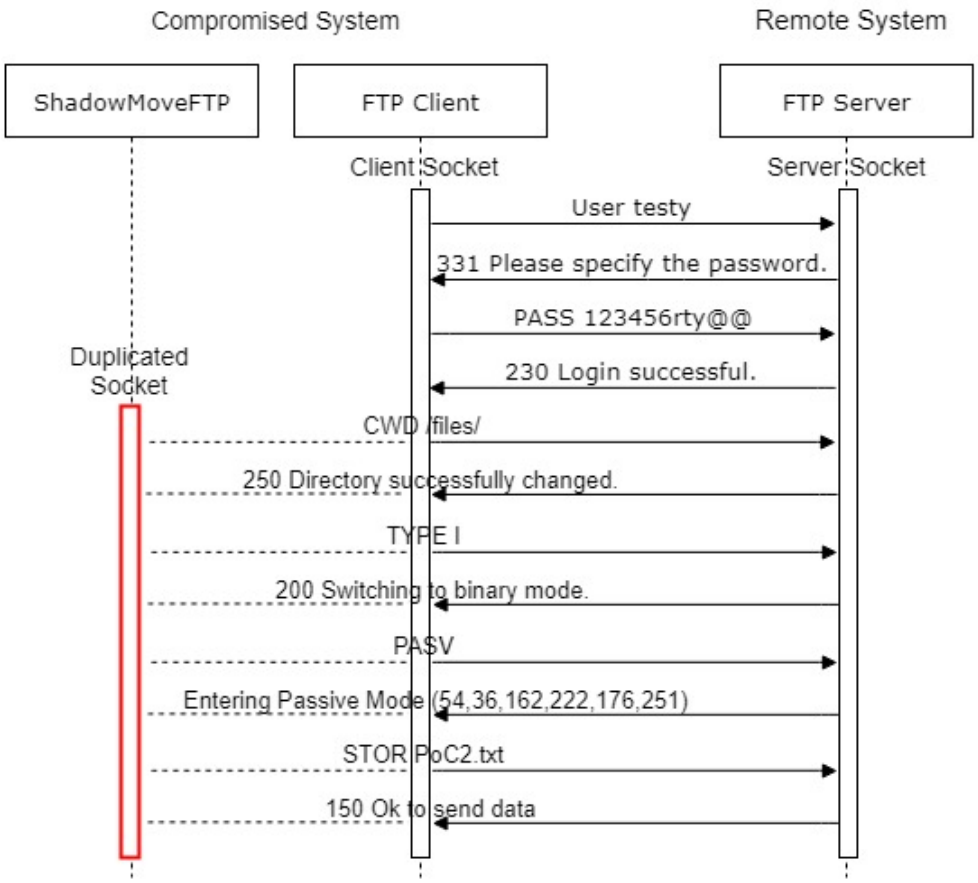


Figure 6.4: ShadowMove injects commands to duplicated FTP socket in order to open a new data channel connection

the server responses are shown in the bottom half of Figure 6.4.

Specifically, in Figure 6.4, it can be seen that the server responded to the PASV request and asked FTPShadowMove to connect back to 54.36.162.222 on port 45307 (i.e., $176 * 256 + 251$). FTPShadowMove, run by an attacker, requests to upload a file named $PoC2.txt$ on the server. After receiving response code 150 from the server, FTPShadowMove opened a TCP connection to the specified remote endpoint and sent the content of the file to the opened connection. The server interpreted the file as binary content and stored it in $/files/PoC2.txt$ on the server.

In the prototype system, only a few FTP commands are implemented. However, there are many other FTP commands that can be utilized by attackers. A complete list of all possible FTP commands can be found at [109]. Specifically, The FTP

SITE command allows a user to execute a limited number of commands via the FTP server on the host machine [110]. No further authentication is required to execute the command. The commands that may be executed vary from system to system, and some useful ones include EXEC and CHMOD. The EXEC command executes provided executable on the server, which can be used to start malware. Another interesting example is the SPSV command, which is an FTP extension that allows IP forwarding. By using this command, the attacker can use the FTP server as a proxy to reach other systems.

Unfortunately, on many systems, the SITE command is not implemented, and it is also recommended that the SITE command be disabled on FTP servers if possible.

6.5.2 WinRMShadowMove: Remote Execution based on WinRM

Windows Remote Management (WinRM) is a feature of Windows that allows administrators to remotely run management scripts [111]. I have confirmed that it is possible to hijack WinRM sessions to run malware on a remote machine. Let's assume that the remote machine is running the WinRM service and the malware has been uploaded to the remote machine and it just needs to be launched.

WinRM handles remote connections by means of the WS-Management Protocol, which is based on SOAP. It is designed to provide interoperability and consistency for enterprise networks that have a variety of operating systems, to locate and exchange management information. WinRM provides a command-line interface that can be used to perform common management tasks, and it also provides a scripting API so users can write their own Windows Scripting Host-based scripts.

6.5.2.1 Brief introduction to the WinRM protocol

WinRM protocol [111, 112, 113] uses HTTP to communicate with the remote server. To authenticate with remote machine WinRM has six authentication mechanisms:

Basic, Digest, Kerberos, Negotiate, Certificate, and CredSSP. By default it uses Negotiate. Using these authentication mechanisms a WinRM client first authenticates with the WinRM server. After authentication, the WinRM client receives a shellID from the server, which is used in later communications. Beside shellID there are a few other IDs in every request message. For every request and response message, there is a messageID, which is used to uniquely identify a particular request or response message. Moreover, this ID is used for matching the response with the corresponding request. In the response message, the request messageId is present as the "RelatesTo" field. Figure 6.5 illustrate the message exchanges during a WinRM session.

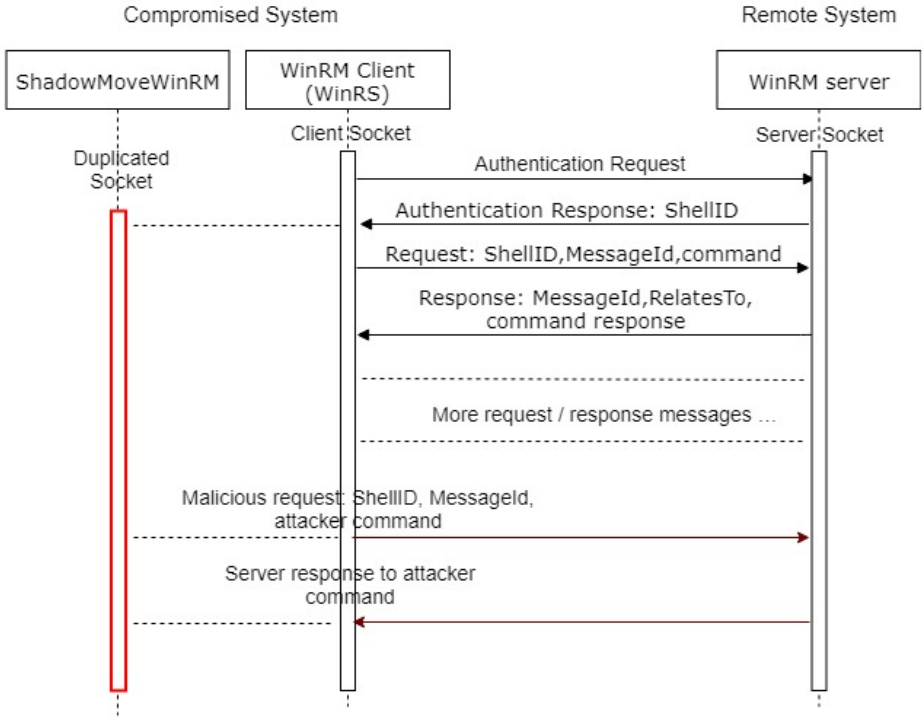


Figure 6.5: ShadowMove Injects attack payload to execute a binary in remote system.

6.5.2.2 Prepare the environment for WinRM hijacking

WinRM server on the remote machine is configured with the default WinRM configuration, to enable basic authentication, and to allow the transfer of unencrypted data by the WinRM server. WinRM client is also configured to use basic authentication

and allow the transfer of unencrypted data. Moreover, the WinRM server is addded as a trusted host on the WinRM client.

To run commands in a remote Windows machine, on client-side, `winrs` is used as the client. The command below is used in Windows PowerShell command line, which will create a new `winrs` process and open a command shell to the remote machine. The `-un` flag specifies that the request and response messages will not be encrypted.

```
winrs -un -r:http://host_ip:5985 -u:user -p:pass cmd
```

Note that allowing unencrypted traffic makes the WinRM session vulnerable. Unfortunately, most of the time this configuration is used to get WinRM work quickly. Furthermore, some third party WinRM client and libraries [114] require unencrypted payload to communicate with the WinRM server.

6.5.2.3 Hijacking WinRM

As the `winrs` process starts execution, it establishes a TCP connection to the WinRM server, which is captured by the Connection Detector. As a result, the Connection Detector notifies the Socket Duplicator, which finds and duplicates the socket inside the `winrs` process. If this socket is chosen for lateral movement to the WinRM server, WinRMShadowMove accesses the socket to peek into the incoming network packets, in order to learn the shellID from the packets. To learn how to construct the payload, I leveraged an open-source WinRM client called winrm4j [115] to communicate with a remote WinRM server, and I use the request packets generated by winrm4j as the template for the payload. Figure 6.6 shows the payload of an example WinRM request packet.

Because the WinRM server supports unencrypted payload, I can construct a plaintext HTTP payload and send it to the server through the TCP socket. For this scheme to work, the constructed payload must appear legitimate to the server. After

analyzing the HTTP request and response packets using Wireshark, I found that in request payload there is a messageID and a shellID. MessageID is unique for every payload, and the shellID is unique for every session. I use a UUID generator to generate messageID and I get the shellID from a response message. Using these two IDs, one can construct a payload to execute a binary file on the remote WinRM server.

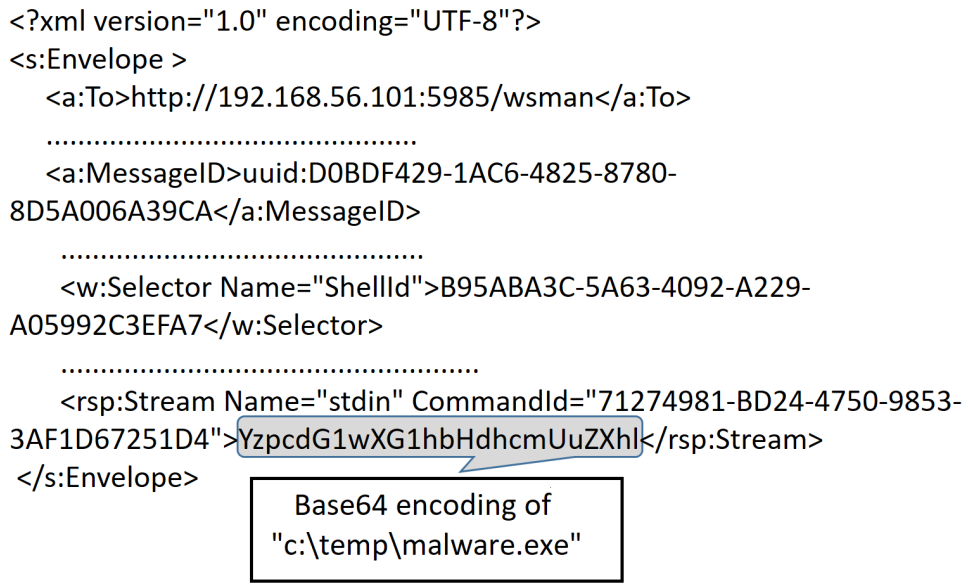


Figure 6.6: A WinRM request message for running malware.exe on a WinRM server whose IP address is 192.168.56.101

Before sending the payload to the remote machine using the hijacked TCP socket, WinRMShadowMove suspends the legitimate process (by invoking `NtSuspendProcess`) to prevent it from getting the response message from the WinRM server. After getting the response from the WinRM server it resumes the legitimate client, by invoking `NtResumeProcess`. The time interval between the suspension and resumption is very short, so the legitimate client may not notice it. Figure 6.5 shows the interleaving of the attack messages with the legitimate WinRM messages.

6.5.3 SQLShadowMove: Hijacking Microsoft SQL Sessions

I have confirmed that it is possible to (1) hijack Microsoft SQL connections to upload malware executables from a SQL client machine to a SQL server, and (2) execute the malware on the SQL server. The legitimate SQL client is Microsoft SQL Server Management Studio 17, and the server is Microsoft SQL Server version 14.0.1000.169. I develop a proof-of-concept called SQLShadowMove.

My SQL hijacking scheme requires several preconditions to work successfully: (1) the traffic is not encrypted, (2) the SQL client has successfully connected (i.e., authenticated) to the SQL server, (3) the SQL client assumes a role that is allowed to create a table on the SQL server, (4) there is a folder on the SQL server writable by the SQL server process.

The above preconditions can often be satisfied. By default, the Microsoft SQL traffic is not encrypted, and the SQL server is almost stateless. The client and the server uses the TDS (Tabular Data Stream) Protocol [116] to communicate. Although several fields in the TDS header are designed for maintaining some states, they are optional or are not used by the current implementation. For example, the SPID field in the TDS packet header is the process ID on the server corresponding to the current connection. If this ID is strictly checked, the attacker has to somehow learn it before fabricating a rogue packet. Unfortunately, this field is not required, and 0x0000 is acceptable by the server. Similarly, two more fields are defined but ignored: PacketID and Window.

There are several types of TDS packets. The most relevant type to ShadowMove attack is the Batch Client Request type [117], whose payload can be a Unicode encoding of any SQL statement string, and there is no checksum in the packet header. This makes it straightforward to capture a real Batch Client Request packet and then use it as a template to create new rogue requests by replacing the payload with new Unicode strings; in my case, such strings correspond to a series of SQL statements.

SQLShadowMove first looks for the Microsoft SQL Server Management Studio process named "ssms.exe". If this process exists, SQLShadowMove duplicates its socket. Then SQLShadowMove uses the duplicated socket to send a series of Batch Client Request packets to the SQL server, and receives any response packets from the SQL server. The payload of these Batch Client Request packets consists of SQL scripts that upload an executable file to the SQL server and execute it.

Specifically, the SQL scripts first create a table on the SQL server, then they insert chunks of bytes from the executable file into the table. Finally, they invoke the `bcp` command to export the content of the table to a regular file on the server, thus restoring the original executable file. The pseudo-code of the SQL scripts is shown in Figure 6.7. With the executable on the SQL server, the current prototype can further run it through a SQL statement.

```
• Create a table "BlobFileTable" that has a column with
  name "FileContent" and type varbinary(max).

• Add chunks of bytes from the source file into the table, e.g.,
     insert into BlobFileTable(FileContent) values (0x4D5A000000)
     insert into BlobFileTable(FileContent) values (0x89EB003401)
     More insert statements ...

• Invoke the "bcp" command to dump the content of BlobFileTable
  to a file on the SQL server

• Use xp_cmdshell to execute the dropped file.
```

Figure 6.7: SQL scripts used by SQLShadowMove

To experimentally confirm the feasibility of SQLShadowMove, I develop a simple Windows application (named notepad.exe) to represent a piece of "malware". This application creates a file (named notepad.txt) in the same folder as the application executable and writes the current date and time into that file. Then I generate SQL scripts to upload the simple "malware" to $C:\backslash tmp\backslash notepad.exe$ on the SQL server and run it. After I run the proof-of-concept of SQLShadowMove, I can visually confirm that first $C:\backslash tmp\backslash notepad.exe$ appears on the SQL server, and then $C:$

\tmp\notepad.txt appears and its content matches the time and date on the SQL server. A video clip of how SQLShadowMove works is available at [118].

Note that in order to run the `bcp` command or the executable file, `xp_cmdshell` has to be enabled on the SQL server. However, this is not a hurdle for the current prototype because the SQL scripts enable `xp_cmdshell` before using it.

6.6 Evaluation of ShadowMove Proof-of-concepts

6.6.1 Theoretical Evaluation

As I demonstrated in Section 6.6.2, ShadowMove cannot be detected by the current state-of-the-art lateral movement detectors. In this section, I discuss the underlying reasons that make such existing solutions ineffective in the detection of ShadowMove lateral movements.

At the host level, first, to perform lateral movements, ShadowMove relies on a few Windows API functions that are also commonly used by other benign processes. As an example, as mentioned in [119], many processes on Windows call OpenProcess() with PROCESS_ALL_ACCESS access flag, which is essentially asking for all possible permissions on the target process, including permission for duplicating its handles. Moreover, ShadowMove calls WSADuplicateSocket() which also has legitimate use cases such as offloading sockets to child processes. Second, it is hard to trace back from a socket descriptor to all processes that have access to it, because only the process id of the owner is recorded in a socket descriptor.

At the network level, ShadowMove tunnels its messages through existing connections established by benign processes on both ends. In other words, it injects its messages within the streams of benign messages send by a benign client to a remote service. Hence, anomaly-based solutions that detect unusual new connections are oblivious to ShadowMove. Moreover, ShadowMove begins the lateral movements after the required authentication steps are performed by the client and the remote

server. This means that ShadowMove operations do not entail any additional authentication attempts. As a result, those anomaly detection solutions that correlate user login activities with network connection activities such as [92] are ineffective.

6.6.2 Experimental Evaluation

In this section, I extensively evaluate ShadowMove in presence of the host and network-based defensive mechanisms that are typically found in Enterprise environments. To be more specific, I test ShadowMove against emerging Endpoint Detection and Response (EDR) systems, top-notch anti-virus products, and network-based IDSes.

I evaluate ShadowMove in the presence of emerging Endpoint Detection and Response (EDR) systems, namely CrowdStrike Falcon Prevent and Cisco AMP. EDR systems are relevant to this evaluation because some EDRs (such as CrowdStrike Falcon [94]) are designed to detect lateral movements. I also evaluate ShadowMove in presence of host-based antivirus products; I choose the top four antivirus products ranked by [120] for this evaluation: McAfee, Norton, Webroot, and Bitdefender. I also choose Windows Defender because it is the default AV on Windows systems. Moreover, I choose the Snort IDS to evaluate ShadowMove against network-based solutions. Snort rules V2.9 is used in this evaluation.

Stealthiness against EDR and IDS solutions I experimentally confirmed that ShadowMovePOCs can evade the detection of Strike Falcon Prevent, Cisco AMP, and Snort. The detailed result is shown is Table 6.4.

Stealthiness against host-based Antivirus products: I also experimentally confirmed that ShadowMovePOCs can evade the detection of the latest version of the above five AVs on Windows 10. The overall result is shown in Table 6.4.

Vendor Feedback: I made the initial contact with Microsoft Security Response Center (MSRC) on June 14, 2018, which responded with a request for technical

details and a POC. On June 15, 2018, a case (number 46036) was opened for the reported issue, and I was requested to “respect coordinated vulnerability disclosure and not report this publicly” before MSRC have notified me that the issue is fixed. On June 21, 2018, MSRC dismissed the reported issue as a vulnerability, stating that "this behavior is by-design ... because from a system security standpoint, one cannot duplicate a handle from a process without already having full control over it and at that point there are many other attacks possible." This feedback from Microsoft engineering team confirmed that this attack is non-trivial to deal with because fully addressing it will require a re-design of the access control mechanism of handles in Windows. This also implies that techniques like ShadowMove will continue to help attackers on Windows in the foreseeable future.

Table 6.4: Effectiveness of Anti-Virus and IDS against ShadowMove POCs

Type	Name	Version	Update time	FTPShadowMove	SQLShadowMove	WinRMShadowMove
AV	McAfee	16.0	3-Feb-2019	No	No	No
AV	Norton	22.16.2.22	3-Feb-2019	No	No	No
AV	Webroot	9.0.24.37	3-Feb-2019	No	No	No
AV	Bitdefender	6.6.7.106	3-Feb-2019	No	No	No
AV	Windows Defender	4.18.1901.7	3-Feb-2019	No	No	No
IDS	Snort	2.9.12	7-Feb-2019	No	No	No
EDR	Cisco AMP	6.1.5.10729	14-Jun-2018	No	No	No
EDR	CrowdStrike Falcon Prevent	4.20.8305.0	11-Feb-2019	No	No	No

6.7 Limitations

The current ShadowMove approach has several limitations. First, it cannot hijack system processes' sockets due to a lack of privileges. Combining ShadowMove with privilege escalation would empower ShadowMove to hijack protocols such as SMB and WMI. Second, it has to find an unencrypted TCP channel because it is a user-level attack that cannot obtain secrets inside the victim process. Due to this limitation, ShadowMove cannot hijack connections for which user-level encryption is applied to the payload. However, there are proposals to implement encryption service (such as TLS) in the kernel space [121], which will make the TLS session vulnerable to ShadowMove because the unencrypted payload is sent to or received from the socket interface in systems that deploy such kernel-level services. Third, ShadowmMove has to peek through the receiving buffer to get information such as the shellID. If the legitimate client reads the buffer before ShadowMove can peek through the buffer, it cannot get the information. However, in this kind of attack, attackers do not need to be successful every time: the attacker needs to succeed only once to achieve lateral movement.

6.8 Related Work

In this section, I, first, discuss existing lateral movement strategies used by attackers, in particular, advanced persistent threats (APTs). Then, I review in details several important defensive mechanisms to prevent or detect lateral movements in enterprise networks.

Traditionally, attackers exploit vulnerabilities in network services, such as SMB or RDP, to laterally move across networks. However, due to the advancements in defense mechanisms, finding such vulnerabilities and exploiting them successfully without being detected has become increasingly hard. As a result, attackers have shifted

their attention to more fruitful approaches such as harvesting user credentials from compromised systems and reusing such credentials to do the lateral movement. In credential dumping approach [86], attackers retrieve plaintext account information including passwords from memory of processes such as LSASS. Several open-source frameworks such as Mimikatz exist that can carve passwords from various locations in a system. Instead of retrieving the credentials, it is also possible to harvest and reuse security tokens, such as Kerberos TGT, Kerberos service ticket, and NTLM hash, to get access to other systems in a network. Many APT groups, including APT 19, 32, use such techniques to expand their access across the target networks.

The hijacking approach presented in this paper is different from traditional hijackings such as session hijacking in web applications and network-level TCP hijacking. Instead, what I propose is a host-level TCP hijacking by performing socket duplication. SSH-Jack [95] is a technique that injects code into the memory of a legitimate SSH client in order to establish a rogue SSH connection via the SSH client, which is trusted by the SSH server. Process injection is a well-understood attack that can be detected by state-of-the-art anomaly detection systems such as Windows Defender ATP [103]. ShadowMove does not inject any code into benign processes, so it is stealthier than SSH-Jack. Moreover, unlike SSH-Jack, ShadowMove is application-agnostic in the sense that it does not need to know the internal implementation of clients in order to inject commands. ShadowMove is also protocol agnostic and can be extended to support other application protocols. In the current prototype, ShadowMove can handle FTP, WINRS, and TDS protocols.

ShadowMove is capable of spoofing traffic, but it is different from other traditional spoofing techniques: instead of eavesdropping on the network, ShadowMove sniffs traffic on the host; instead of capturing packets at the kernel level (like what WireShark does), ShadownMove sniffs traffic at the user level. The sniffing technique of ShadowMove is enabled by a novel way of socket duplication on Windows OS.

Lateral movement usually involves privilege escalation or harvesting of additional credentials [122]. ShadowMove does not rely on either privilege escalation or credential harvesting, so it is a new type of lateral movement. More importantly, ShadowMove reuses authenticated connections; which means that it will not cause a new authentication procedure to be performed on the remote end.

Structural Anomaly Detection Attackers commonly attempt to steal user credentials within an enterprise to do lateral movements. To identify such attacks, researchers in [92] proposed a new machine learning framework that extracts normal users' login patterns and identifies login attempts that deviate from such patterns. The underlying assumption in their work is that attackers attempt to reuse learned credentials in a greedy way (i.e., testing all credentials on all reachable systems), which deviates from the way that normal users behave in a network, thus enabling defenders to effectively identify attacker's lateral movements within their networks. However, as I show in this work, attackers can adopt a more intelligent strategy to silently evade such systems. Basically, they can use existing connections between the system without creating new login connections.

Differentiating User Authentication Graphs Kent et al. [93] report that non-privileged users and privileged users have very different authentication behavior patterns in a large organization that has a unified central authentication system (e.g., Kerberos). By comparing authentication graphs generated out of the access records (with approximately 10,000 users and over a period of 4 months) of the Microsoft Windows Active Directory (KDC) authentication system at Los Alamos National Laboratory, they found that three categories of users (non-administrators, administrators, and institutional administrators) have distinct differences in terms of host (node) count, graph diameter, and maximum in degree. Specifically, administrators have more complex and extensive graphs (e.g., logging into more computers) than non-administrator users. Therefore, they suggest that user authentication graphs can

be used to detect authentication credential misuse or malicious insiders in large-scale, enterprise networks.

Proactive Insider Threat Detection through Graph Learning and Psychological Context In [123], the authors proposed a scheme to proactively detect insider attackers. In this scheme, a Structural Anomaly Detection (SA) technique is used in conjunction with a Psychological Profiling (PP) model to find a potential insider attacker. SA enables the defenders to detect communication anomalies within the network while PP helps the defenders to spot users who commit cyberattacks with a higher probability. Incorporating behavioral models can significantly reduce the false-positive ratio of the anomaly detection system compared to when it only relies on structural anomaly detection.

CHAPTER 7: Conclusions and Future Work

In this dissertation, I presented two different approaches for identifying unknown attacks. In the first approach, the underlying idea is to consume cyber threat intelligence in order to predict techniques, tactics, or procedures in addition to network resources that will be used by attackers in the near future. The second approach is to detect zero-day vulnerabilities in existing systems that can be abused by attackers to isolate such vulnerability preemptively. Chapter 2 to 5 presents systems to locate and consume cyber threat intelligence information and utilize them to predict zero-day attacks. Chapter 6 introduces a new class of vulnerabilities that can be abused by attackers to misuse resources shared on compromised systems. It showcases this vulnerability class by presenting a new lateral movement strategy.

7.1 Overview of Contributions

In chapter 2, I described a new approach to predict zero-day IP addresses that potentially will be used by the attackers in the near future based on recent cyber threat intelligence data reports. It is based on two critical observations: I) malicious IP addresses are not spread uniformly on the Internet. II) attackers commonly employ shared hosting services, where the services are shared by multiple independent entities, rather than standalone services; which is reverse for benign users. Through experimentation on two different cyber threat intelligence sources, I showed that with the presented approach, one can detect about 88% of unrecognized malwares by top five antivirus vendors and detect about 68% of phishing URLs.

In chapter 3, I introduced PhishMon, a feature-rich machine learning system to detect phishing websites. PhishMon relies on twenty salient features, seventeen of which

are new, to decide whether the webpage pointed by a given URL is a phish. These features can be efficiently calculated for a given URL, without requiring interaction with any third-party system, which can prohibitively delay the decision-making process. They capture various characteristics of the web application and its underlying infrastructure. By using these features, I indirectly measure the amount of effort invested in development and deployment of the web application, which is remarkably different between legitimate and phishing websites. Through experimentation, I showed that PhishMon achieves a high degree of accuracy in distinguishing legitimate webpages from zero-hour phishing webpages without raising many false alarms: on the test dataset, PhishMon reaches a 95.4% detection rate with 1.3% false positive, which makes it a suitable approach to be used in practice.

In chapter 4, I presented a NLP-empowered information retrieval system, SECCMiner, that enables security professionals to analyze unstructured APT reports and extract key security concepts, such as adversarial techniques and tactics, from them. Next, with the assistance of the proposed solution, I conducted a comprehensive study of real-world APT attacks with the goal of identifying the trend of APT techniques as well as the inter-relationship between APT techniques. To do so, I analyzed 445 technical reports on real-world APT published between 2008 and 2017.

In chapter 5, I presented IoCMiner, which is a scalable framework to extract IoCs from data sharing platforms such as Twitter. Instead of examining all published information in data sharing platforms to locate IoCs, IoCMiner attempts to first identify credible sources of information that regularly publish cyber threat information, and then analyze information posted by such users to extract the indicators of compromise about ongoing cyber attacks. Through evaluation, I showed that a large percentage of the indicators of compromises discovered by IoCMiner is not reported by traditional threat intelligence data sources at the time discovery.

In chapter 6, I introduced a novel lateral movement strategy, called ShadowMove,

that allows APT attackers to make stealthy lateral movements within an enterprise network. Built upon a novel socket duplication technique, ShadowMove leverages existing benign network connections and does not require any elevated privilege, new connections, extra authentication, or process injections. Therefore, it is capable of evading the detection of host- and network- level defensive mechanisms. To confirm the feasibility of my approach, I developed a prototype of ShadowMove for modern versions of Windows operating system, which successfully abuses three common enterprise protocols (i.e., FTP, WinRM, and Microsoft SQL) for lateral movement, such as uploading malware to the next target machine and starting the malware execution on the next target. I also experimentally confirm that the prototype implementation is undetectable by state-of-the-art antivirus products, IDSes (such as Snort), and endpoint detection and response (EDR) systems.

7.2 Future Research

The results in chapter 2 strongly support the idea that shared hosting services are being targeted by attackers and used for launching different types of attacks and hence is a good metric that can be used to detect zero-day attacks. My preliminary investigation of such services suggests that these services are attractive for attackers mainly due to their low cost of renting as well as poor security. In the future, I want to investigate the reasons more deeply to understand the business model of the attackers and determine whether attackers can evade by changing their behaviors easily. I also plan to improve the proposed algorithm of detecting shared hosting service providers such that it can distinguish the IP addresses of shared hosting providers from the ones that are used by content delivery networks (CDNs) and DNS parking servers and study each of these groups of IP addresses separately.

Phishers may attempt to evade PhishMon, the presented system in chapter 3, by tweaking their web applications. For example, to fool PhishMon classifier, they

may add some dead code to their applications to increase the LOC and cyclomatic complexity. However, in essence, the resulted application is still simple regarding functionalities they offer; hence more expressive features can be proposed to measure code complexity more precisely. I argue that the only true way to evade the presented system is to follow coding best practices, to utilize conventional security mechanisms, and to provide more functionalities to the users; which is cost prohibitive. In the future, I plan to consider more resilient features to capture the complexity of client-side codes such as measuring the number of functions reachable from user inputs.

Current prototype of IoCMiner aggregates cyber threat posts based on existence of shared IoCs. In the future, I plan to enhance the aggregation process to more accurately locate and combine posts related to a specific attack. Moreover, I plan to mix IoCMiner with SECCMiner and extend them to generate possible attack plans used by cyber attakcers based on the observed threat information. Based on these attack plans, IoCMiner can then suggest what defensive mechanisms are appropriate for dismantling the observed attacks. This information can help defenders to evaluate their protections against current cyber attacks.

In addition to facilitating lateral movement, ShadowMove socket duplication technique can be used for other malicious purposes, such as data stealing and malicious data injection. As discovered by [124], TCP communication among applications inside a machine (such as a browser and a backend password manager) is not totally secured. Therefore, the presented socket duplication technique can be used to intercept and steal sensitive data from such applications. Moreover, in this study, I tried to exploit only client applications, but using the same technique one can also exploit server applications. For example, by duplicating sockets used by a server application, one can inject malicious data to mount a phishing attack against a client machine, hence providing an alternative implementation for the attack described in [125].

REFERENCES

[1] Z. Dong, A. Kapadia, J. Blythe, and L. J. Camp, "Beyond the lock icon: real-time detection of phishing websites using public key certificates," in *Electronic Crime Research (eCrime), 2015 APWG Symposium on*, pp. 1–12, IEEE, 2015.

[2] Microsoft, "Wsaduplicatesocket function." https://msdn.microsoft.com/en-us/library/windows/desktop/ms741565(v=vs.85).aspx, 2017. [Online; accessed 10-May-2018].

[3] I. Rob McMillan Gartner, "Definition: Threat intelligence." https://www.gartner.com/doc/2487216/definition-threat-intelligence, 2013.

[4] S. D. Forensics and I. R. Blog, "Security intelligence: Attacking the cyber kill chain." https://digital-forensics.sans.org/blog/2009/10/14/security-intelligence-attacking-the-kill-chain, 2009.

[5] F. Li, A. Lai, and D. Ddl, "Evidence of advanced persistent threat: A case study of malware for political espionage," in *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, pp. 102–109, IEEE, 2011.

[6] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *IFIP International Conference on Communications and Multimedia Security*, pp. 63–72, Springer, 2014.

[7] N. Fraser, J. O'Leary, V. Cannon, and F. Plan, "Apt38: Details on new north korean regime-backed threat group." https://www.fireeye.com/blog/threat-research/2018/10/apt38-details-on-new-north-korean-regime-backed-threat-group.html, 2017.

[8] I.-C. Mihai, S. Pruna, and I.-D. Barbu, "Cyber kill chain analysis," *Int'l J. Info. Sec. & Cybercrime*, vol. 3, p. 37, 2014.

[9] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[10] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: a passive dns analysis service to detect and report malicious domains," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 14, 2014.

[11] M. Felegyhazi, C. Kreibich, and V. Paxson, "On the potential of proactive domain blacklisting.," *LEET*, vol. 10, pp. 6–6, 2010.

[12] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM workshop on Recurring malcode*, pp. 1–8, ACM, 2007.

[13] G. C. Moura, R. Sadre, and A. Pras, "Internet bad neighborhoods: the spam case," in *Network and Service Management (CNSM), 2011 7th International Conference on*, pp. 1–8, IEEE, 2011.

[14] M. P. Collins, T. J. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane, "Using uncleanliness to predict future botnet addresses," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 93–104, ACM, 2007.

[15] W3techs, "Usage of top level domains for websites." https://w3techs.com/technologies/overview/top_level_domain/all, 2017.

[16] J. Zhang, P. A. Porras, and J. Ullrich, "Highly predictive blacklisting.," in *USENIX Security Symposium*, pp. 107–122, 2008.

[17] F. Soldo, A. Le, and A. Markopoulou, "Predictive blacklisting as an implicit recommendation system," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[18] B. Stone-Gross, C. Kruegel, K. Almeroth, A. Moser, and E. Kirda, "Fire: Finding rogue networks," in *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pp. 231–240, IEEE, 2009.

[19] D. K. McGrath, A. Kalafut, and M. Gupta, "Phishing infrastructure fluxes all the way," *IEEE Security & Privacy*, vol. 7, no. 5, 2009.

[20] M. Moghimi and A. Y. Varjani, "New rule-based phishing detection method," *Expert systems with applications*, vol. 53, pp. 231–242, 2016.

[21] R. Manning and G. Aaron, "Phishing activity trends report," *Anti Phishing Work Group, Tech. Rep. 4th Quarter 2016*, 2016.

[22] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness.," in *USENIX security symposium*, vol. 13, 2013.

[23] T. Moore and B. Edelman, "Measuring the perpetrators and funders of typosquatting," in *International Conference on Financial Cryptography and Data Security*, pp. 175–191, Springer, 2010.

[24] H. Choi, B. B. Zhu, and H. Lee, "Detecting malicious web links and identifying their attack types.," *WebApps*, vol. 11, pp. 11–11, 2011.

[25] S. Marchal, J. François, R. State, and T. Engel, "Phishstorm: Detecting phishing with streaming analytics," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 458–471, 2014.

[26] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1245–1254, ACM, 2009.

[27] M.-E. Maurer and L. Höfer, "Sophisticated phishers make more spelling mistakes: Using url similarity against phishing.," in *CSS*, pp. 414–426, Springer, 2012.

[28] S. Marchal, K. Saari, N. Singh, and N. Asokan, "Know your phish: Novel techniques for detecting phishing sites and their targets," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pp. 323–333, IEEE, 2016.

[29] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web*, pp. 639–648, ACM, 2007.

[30] G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 2, p. 21, 2011.

[31] K.-T. Chen, J.-Y. Chen, C.-R. Huang, and C.-S. Chen, "Fighting phishing with discriminative keypoint features," *IEEE Internet Computing*, vol. 13, no. 3, 2009.

[32] G. Xiang, B. A. Pendleton, J. Hong, and C. P. Rose, "A hierarchical adaptive probabilistic approach for zero hour phish detection," in *European Symposium on Research in Computer Security*, pp. 268–285, Springer, 2010.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] T.-C. Chen, T. Stepan, S. Dick, and J. Miller, "An anti-phishing system employing diffused information," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 16, 2014.

[35] R. Gowtham and I. Krishnamurthi, "A comprehensive and efficacious architecture for detecting phishing webpages," *Computers & Security*, vol. 40, pp. 23–37, 2014.

[36] S. Gastellier-Prevost, G. G. Granadillo, and M. Laurent, "Decisive heuristics to differentiate legitimate from phishing sites," in *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pp. 1–9, IEEE, 2011.

[37] Webapplalyzer, "Webapplalyzer - identify technologies on websites." https://www.wappalyzer.com/, 2017.

[38] D. L. Lee, H. Chuang, and K. Seamons, "Document ranking and the vector-space model," *IEEE software*, vol. 14, no. 2, pp. 67–75, 1997.

[39] Internet Assigned Numbers Authority (IANA), "Message headers." https://www.iana.org/assignments/message-headers/message-headers.xhtml, 2017.

[40] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, "Predicting maintenance performance using object-oriented design complexity metrics," *IEEE transactions on Software Engineering*, vol. 29, no. 1, pp. 77–87, 2003.

[41] H. Zhang, X. Zhang, and M. Gu, "Predicting defective software components from code complexity measures," in *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pp. 93–96, IEEE, 2007.

[42] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, pp. 78–88, IEEE Computer Society, 2009.

[43] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[44] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.

[45] S. Souders, "High-performance web sites," *Communications of the ACM*, vol. 51, no. 12, pp. 36–41, 2008.

[46] Mozilla Foundation, "Public suffix list." https://publicsuffix.org/list/, 2017.

[47] M. Aburrous, M. A. Hossain, K. Dahal, and F. Thabtah, "Predicting phishing websites using classification mining techniques with experimental case studies," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pp. 176–181, IEEE, 2010.

[48] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, pp. 1137–1145, Stanford, CA, 1995.

[49] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[50] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Advances in neural information processing systems*, pp. 431–439, 2013.

[51] E. H. Chang, K. L. Chiew, W. K. Tiong, *et al.*, "Phishing detection via identification of website identity," in *IT Convergence and Security (ICITCS), 2013 International Conference on*, pp. 1–4, IEEE, 2013.

[52] S. Singh, P. K. Sharma, S. Y. Moon, D. Moon, and J. H. Park, "A comprehensive study on apt attacks and countermeasures for future networks and communications: challenges and solutions," *The Journal of Supercomputing*, pp. 1–32, 2016.

[53] K. Blanda, "Aptnotes." https://github.com/kbandla/APTnotes, 2017.

[54] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, pp. 133–142, 2003.

[55] S. N. Kim, T. Baldwin, and M.-Y. Kan, "Evaluating n-gram based evaluation metrics for automatic keyphrase extraction," in *Proceedings of the 23rd international conference on computational linguistics*, pp. 572–580, Association for Computational Linguistics, 2010.

[56] Y. Shinyama, "Pdfminer: Python pdf parser and analyzer (2010)."

[57] S. Bird and E. Loper, "Nltk: the natural language toolkit," in *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, p. 31, Association for Computational Linguistics, 2004.

[58] Securelist, "The cozyduke apt." https://app.box.com/s/8vksggruwwqzg7a4y7xrsrysrje56pqn, 2015.

[59] P. A. Networks, "Dimnie: Hiding in plain sight." https://app.box.com/s/scdmr7ekxhx4ktprct29ojxyllr41bjq, 2017.

[60] Citizen Lab, "The million dollar dissident: Nso group's iphone zero-days used against a uae human rights defender." https://app.box.com/s/adaa4lfxeohb7ehxv3ao6104gmvq226i, 2016.

[61] Vectra Networks, "Moonlight - targeted attacks in the middle east." https://app.box.com/s/f7p6hmdojxrh6mzs91yvjmpgz528b7h9, 2016.

[62] ESET, "Telebots are back: supply-chain attacks against ukraine." https://app.box.com/s/740pmk3f6nrhfbj9nmcvovc64oah2ibi, 2017.

[63] ESET, "Win32/industroyer a new threat for industrial control systems." https://app.box.com/s/ec8zyav7snvm6vsfhy8ocvvngphe8lqp, 2017.

[64] Kaspersky, "On the strongpity waterhole attacks targeting italian and belgian encryption users." https://app.box.com/s/c9w0xp0mgndij268ku7ti5ee4lxu54bv, 2016.

[65] Cysinfo, "Cyber attack targeting indian navy's submarine and warship manufacturer." https://app.box.com/s/zdwfwsi2pw1081j2reu3qotz577g7pt6, 2017.

[66] Bluecoat, "From seoul to sony: The history of the dark-seoul group and the sony intrusion malware destover." https://app.box.com/s/xyyord0b806e6or2nh92coxw2areyyx4, 2016.

[67] Clearsky, "Operation dusty sky." https://app.box.com/s/cydpeasz6l8cv9o o99o4tpazd5tq4xkm, 2016.

[68] Symantec, "Have i got newsforyou: Analysis of flamer c&c server." https://app.box.com/s/6ujt4gi1c962id9o4iviesurww2grbxi, 2012.

[69] Cylance, "Operation cleaver report." https://www.cylance.com/content/dam/cylance/pdfs/reports/Cylance_ Operation_Cleaver_Report.pdf, 2014.

[70] M. Marschalek, "Evil bunny: Suspect #4." https://app.box.com/s/xvilsesi5qd2gh6so2g3tnric51ndv57, 2014.

[71] Kaspersky, "Energetic bear-crouching yeti." https://app.box.com/s/z0apbug9w1ztt8ex0pe99sq0d2u9r3nu, 2014.

[72] FireEye, "Hammertoss: Stealthy tactics define a russian cyber threat group." https://app.box.com/s/xqp6s3fb8w65f6mkm1zc89ftrl8lyfw7, 2015.

[73] MITRE, "Adversarial tactics, techniques & common knowledge." https://attack.mitre.org/wiki/Main_Page, Accessed October 2018.

[74] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *Software reliability engineering (ISSRE), 2010 IEEE 21st international symposium on*, pp. 111–120, IEEE, 2010.

[75] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 755–766, ACM, 2016.

[76] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting Real-World exploits," in *USENIX Security Symposium*, pp. 1041–1056, usenix.org, 2015.

[77] G. Husari, X. Niu, B. Chu, and E. Al-Shaer, "Using entropy and mutual information to extract threat actions from cyber threat intelligence," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 1–6, Nov 2018.

[78] S. Ghosh, N. Sharma, F. Benevenuto, N. Ganguly, and K. Gummadi, "Cognos: crowdsourcing search for topic experts in microblogs," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pp. 575–590, ACM, 2012.

[79] A. Hamzehei, S. Jiang, D. Koutra, R. Wong, and F. Chen, "Topic-based social influence measurement for social networks," *Australasian Journal of Information Systems*, vol. 21, Nov. 2017.

[80] J. Weng, E.-P. Lim, J. Jiang, and Q. He, "TwitterRank: Finding topic-sensitive influential twitterers," in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, (New York, NY, USA), pp. 261–270, ACM, 2010.

[81] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annu. Rev. Sociol.*, vol. 27, pp. 415–444, Aug. 2001.

[82] M. Montangero and M. Furini, "TRank: Ranking twitter users according to specific topics," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015.

[83] C. Castillo, M. Mendoza, and B. Poblete, "Information credibility on twitter," in *Proceedings of the 20th international conference on World wide web*, pp. 675–684, ACM, Mar. 2011.

[84] M. Alrubaian, M. Al-Qurishi, M. Al-Rakhami, M. M. Hassan, and A. Alamri, "Reputation-based credibility analysis of twitter social network users: Reputation-Based credibility analysis of twitter social network users," *Concurr. Comput.*, vol. 29, p. e3873, Apr. 2017.

[85] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, vol. 6, p. 12, pdfs.semanticscholar.org, 2010.

[86] D. Miller, R. Alford, A. Applebaum, H. Foster, C. Little, and B. Strom, "Automated adversary emulation: A case for planning and acting with unknowns," 2018.

[87] J. Dunagan, A. X. Zheng, and D. R. Simon, "Heat-ray: Combating identity snowball attacks using machine learning, combinatorial optimization and attack graphs," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, (New York, NY, USA), pp. 305–320, ACM, 2009.

[88] S. Duckwall and C. Campbell, "Hello, my name is microsoft and i have a credential problem," in *Blackhat USA 2013 White Papers*, 2013. https://media.blackhat.com/us-13/US-13-Duckwall-Pass-the-Hash-WP.pdf.

[89] Strategic Cyber LLC, "Cobalt strike: Advanced threat tactics for penetration testers." https://cobaltstrike.com/downloads/csmanual38.pdf, 2017. Accessed February 2019.

[90] B. Deply, "Mimikatz." https://github.com/gentilkiwi/mimikatz, 2014. Accessed February 2019.

[91] S. Metcalf, "Unofficial guide to mimikatz & command reference." https://adsecurity.org/?page_id=1821, 2018. Accessed February 2019.

[92] H. Siadati and N. Memon, "Detecting structurally anomalous logins within enterprise networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1273–1284, ACM, 2017.

[93] A. D. Kent and L. M. Liebrock, "Differentiating user authentication graphs," in *2013 IEEE Security and Privacy Workshops*, pp. 72–75, May 2013.

[94] CrowdStrike Inc, "CrowdStrike Compromise Assessment Data Sheet." https://www.crowdstrike.com/wp-content/brochures/CrowdStrike_CompromiseAssessment _DataSheet.pdf, 2019. Accessed February 2019.

[95] A. Boileau, "Trust Transience: Post Intrusion SSH Hijacking," in *BlackHat Briefings*, August 2005.

[96] J. Levin, "The dark side of Winsock." https://www.defcon.org/images/defcon-13/dc13-presentations/DC_13-Levin.pdf, 2005. Online; Accessed January 2019.

[97] Z. Shan, Y. Yu, and T.-c. Chiueh, "Confining windows inter-process communications for os-level virtual machine," in *Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems*, VDTS '09, (New York, NY, USA), pp. 30–35, ACM, 2009.

[98] Tune, "Windows 10 startup folder location. add a program to startup in windows 10." https://tunecomp.net/add-program-to-startup-windows-10/, 2018. Accessed February 2019.

[99] Microsoft, "Change which apps run automatically at startup in windows 10." https://support.microsoft.com/en-us/help/4026268/windows-10-change-startup-apps, 2018. Accessed February 2019.

[100] FireEye FLARE Team, "Windows management instrumentation (wmi) offense, defense, and forensics." https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf, 2015. Accessed February 2019.

[101] Microsoft, "Mib_tcprow2 structure." https://docs.microsoft.com/en-us/windows/desktop/api/tcpmib/ns-tcpmib-_mib_tcprow2, 2018. Accessed February 2019.

[102] Microsoft, "Duplicatehandle function." https://msdn.microsoft.com/en-us/library/windows/desktop/ms724251(v=vs.85).aspx, 2017. [Online; accessed 10-May-2018].

[103] Windows Defender Research, "Detecting stealthier cross-process injection techniques with windows defender atp." https://cloudblogs.microsoft.com/microsoftsecure/2017/ 07/12/detecting-stealthier-cross-process-injection-techniques-with-windows-defender-atp-process-hollowing-and-atom-bombing/, 2019. Accessed Feb 2019.

[104] D. Treadwell, "socket.c." http://icerote.net/doc/library/programming/source/SOURCE.CODE.MICROSOFT.WINDOWS.2000.AND.NT4-BTDE/win2k/private/net/sockets/winsock2/wsp/msafd/socket.c, 1992. Accessed January 2019.

[105] D. Treadwell, "wspmisc.c." http://icerote.net/doc/library/programming/source/SOURCE.CODE.MICROSOFT.WINDOWS.2000.AND.NT4-BTDE/win2k/private/net/sockets/winsock2/wsp/msafd/wspmisc.c, 1992. Accessed January 2019.

[106] Internet Assigned Numbers Authority (IANA), "Service name and transport protocol port number registry." https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, 2019.

[107] Suricata, "Suricata features." https://suricata-ids.org/features/, 2018. Accessed November 2018.

[108] "Video Clip for the FTPShadowMove." https://drive.google.com/%66%69le/d/1gL4xSZQUcNL eaJTGV-pkRI4D6b2o0XuX/view?usp=sharing, 2018.

[109] "List of ftp commands." https://en.wikipedia.org/wiki/List_of_FTP_commands, 2018. Accessed February 2019.

[110] SolarWinds, "SITE FTP command." https://support.solarwinds.com/Success_Center/Serv-U_Managed_File_Transfer_Serv-U_FTP_Server/Knowledge base_Articles/SITE_FTP_command, 2017. Accessed February 2019.

[111] Microsoft, "Windows Remote Management." https://docs.microsoft.com/en-us/windows/desktop/WinRM/portal. Accessed November 2018.

[112] R. Ries, "Monitoring with Windows Remote Management (WinRM) and Powershell Part I." https://www.myotherpcisacloud.com/post/Monitoring-with-Windows-Remote-Management-(WinRM)-and-Powershell-Part-I. Accessed November 2018.

[113] VMware, "Configure WinRM to Use HTTP." https://docs.vmware.com/en/vRealize-Automation/7.5/com.vmware.vrealize.orchestrator-use-plugins.doc/GUID-D4ACA4EF-D018-448A-866A-DECDDA5CC3C1.html. Accessed November 2018.

[114] "winrm for go library." https://github.com/masterzen/winrm. Accessed November 2018.

[115] "winrm4j." https://github.com/cloudsoft/winrm4j. Accessed November 2018.

[116] MSDN, "[MS-TDS]: Tabular Data Stream Protocol." https://msdn.microsoft.com/en-us/library/dd304523.aspx, 2018. Accessed November 2018.

[117] MSDN, "[MS-TDS]: SQL Batch Client Request." https://msdn.microsoft.com/en-us/library/dd304416.aspx, 2019. Accessed November 2018.

[118] "Video Clip for the SQLShadowMove Demo." https://drive.google.com/file/d/0B_GOo1eccP_xNFRW Mlg0S0NJZXRQTkVUaTdnVC1jSW9Ra3Rj/view?usp =sharing, 2019.

[119] A. Blaszczyk, "Can we stop detecting mimikatz please?." http://www.hexacorn.com/blog/2019/02/03/can-we-stop-detecting-mimikatz-please/, 2019. Accessed Feb 2019.

[120] N. J. Rubenking, "The Best Antivirus Protection for 2019." https://www.pcmag.com/article2/0,2817,2372364,00.asp, 2019. [Online; accessed 04-February-2019].

[121] M. O'Neill, S. Heidbrink, J. Whitehead, T. Perdue, L. Dickinson, T. Collett, N. Bonner, K. Seamons, and D. Zappala, "The secure socket API: TLS as an operating system service," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 799–816, USENIX Association, 2018.

[122] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *Communications and Multimedia Security* (B. De Decker and A. Zúquete, eds.), (Berlin, Heidelberg), pp. 63–72, Springer Berlin Heidelberg, 2014.

[123] O. Brdiczka, J. Liu, B. Price, J. Shen, A. Patil, R. Chow, E. Bart, and N. Ducheneaut, "Proactive insider threat detection through graph learning and psychological context," in *2012 IEEE Symposium on Security and Privacy Workshops*, pp. 142–149, May 2012.

[124] T. Bui, S. P. Rao, M. Antikainen, V. M. Bojan, and T. Aura, "Man-in-the-machine: Exploiting ill-secured communication inside the computer," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 1511–1525, USENIX Association, 2018.

[125] W. Chen and Z. Qian, "Off-path TCP exploit: How wireless routers can jeopardize your secrets," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 1581–1598, USENIX Association, 2018.