Internet Communications Data Profiling for Detection of Evolving Cyber Attacks

進化するサイバー攻撃の検知のための インターネット通信データプロファイリング

July, 2017

Daiki CHIBA

千葉　大紀

Internet Communications Data Profiling for Detection of Evolving Cyber Attacks

進化するサイバー攻撃の検知のための インターネット通信データプロファイリング

July, 2017

Waseda University

Graduate School of Fundamental Science and Engineering
Department of Computer Science and Communications Engineering,
Research on Information Systems

Daiki CHIBA

千葉　大紀

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The Internet has become an indispensable part of the infrastructure of our social lives, but important information on the Internet is constantly under threat from ever-evolving cyberattacks. Most cyberattacks arise from the infection of users' devices by malicious software (malware). Once the user devices are infected by malware, they can be coerced by attackers into conducting new cyberattacks. Thus, to break the vicious cycle of cyberattacks, we must implement countermeasures against malware infections.

Countermeasures against malware infections can be divided into two main types: host-based countermeasures and network-based countermeasures. Host-based countermeasures include antivirus software that mainly focuses on the information contained in malware files. Antivirus software uses predefined signatures generated from known malware files to scan the files on end hosts to detect malware infections. However, these host-based countermeasures have reached the limits of their detection capabilities, as they have lately been unable to catch up with the increasing number of malware files that bypass their predefined signatures [1, 2, 3]. For example, attackers can easily change their malware files using different packing techniques, encryption, and polymorphism to bypass the detection capabilities of antivirus software.

Network-based countermeasures include blacklists based on information regarding malicious network communication [4, 5, 6]. Generally, such blacklists are composed of malicious domain names, uniform resource locators (URLs), and communication patterns. These blacklists can be used not only to prevent end hosts from being infected by new malware but also to detect malicious end hosts already infected with malware. In most malware infections, network communication is needed to for attackers to carry out malicious activities such as downloading malware from external servers or sending control commands. Owing to this underlying feature of malware infections, network-based countermeasures are more promising and effective than host-based countermeasures.

The lists of malicious domain names, URLs, and communication patterns used in network-based countermeasures are generated by a three-step process [7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. First, decoy systems called honeypots are used to monitor cyberattacks and to store malicious network communications and samples of the malware associated with the attacks. Second, malware analysis systems run the malware samples collected by the honeypots and monitor malicious behavior while simulating malware infections. Finally, the data collected in the previous steps are analyzed to detect malicious domain names, URLs, and communication patterns for the countermeasures. When analyzing the data, it is extremely important to detect previously unknown malicious communications that cannot be directly observed by honeypots or malware analysis systems so that we can deploy effective countermeasures against future malware threats. However, conventional data analysis methods have been unable to catch up with current malicious communications due to the evasion techniques employed by attackers. This thesis determines that such malicious communications cannot be discovered by conventional methods owing to the following four fundamental problems.

Malicious web content is dispersed by attackers Attackers use several types of malicious web content to distribute malware and maximize the success rate of their malware infection. For example, in a typical drive-by

download attack, they deploy multiple versatile redirection websites ahead of the actual malware distribution website to evade detection. Conventional analysis methods fail to detect such dispersed malicious websites since they deal with the domain name or URL of each malicious website individually.

Malicious domain names change over time Attackers abuse domain names and the Domain Name System (DNS) to obfuscate their attack ecosystems. Specifically, they systematically generate a huge number of distinct domain names to make it infeasible to keep up with all the newly-generated malicious domain names. Conventional analysis methods cannot catch up with such ever-changing malicious domain names since they only use information from a particular point in time.

Attackers generate differently-structured malicious domain names Attackers create their cyberattacks using malicious domain names with different structures to prevent their attacks from being detected by fixed defense solutions. For example, some malicious domain names are created by abusing legitimate services, such as online advertising and web hosting. If we filter out the domain names used by such legitimate services, we may prevent users from accessing legitimate services and disrupt legitimate businesses. Other malicious domain names are created by an algorithm known as a domain generation algorithm (DGA) with the intention of deceiving users. If we hesitate to filter these domain names, we cannot decrease the threat of cyberattacks. Conventional analysis methods that only focus on a single countermeasure will not always use the right countermeasure for each malicious domain name.

Malicious HTTP communications blend in to evade detection Attackers design their malicious communications to blend in with legitimate ones so that their attacks can bypass detection methods. Specifically, attackers make their malicious communications (associated with malware infections) look as much like legitimate communications generated by real users in a network as possible to evade detection. Because of this, conventional analysis methods may falsely regard some legitimate communications as malicious, resulting in false positives.

1.2 Thesis Contributions

This thesis focuses on data analysis methods for network-based countermeasures against cyberattacks and malware. In particular, this thesis sheds light on the aforementioned four fundamental problems that hamper conventional countermeasures against cyberattacks and explores possible solutions to these problems. The goal of this thesis is to improve countermeasures against cyberattacks significantly by implementing new and practical data analysis methods. To this end, this thesis proposes the following four new analysis methods and evaluates the effectiveness of them from several perspectives using large, real datasets.

Detecting unseen malicious web content To tackle the problem of dispersed malicious web content, this thesis proposes a new scheme for detecting websites with malicious web content by profiling the characteristics of their IP addresses. The scheme leverages the empirical observation that IP addresses are more stable than other metrics, such as URLs and DNS records. While the strings that form URLs or DNS records are highly variable, IP addresses are much less variable, for example the IPv4 address space is mapped onto four-byte strings. This thesis develops a lightweight and scalable detection scheme based on the characteristics of IP addresses, utilizing machine learning techniques. The effectiveness of the scheme is validated by using real IP address data from existing blacklists and real traffic data on a network. The results demonstrate that the proposed scheme can expand the coverage and accuracy of existing blacklists and also detect unseen malicious websites that cannot be found by conventional methods.

Finding domain names that may be abused in future To provide a solution to the problem of malicious domain names changing over time, this thesis proposes a new system to predict which domain names could potentially be put to malicious use in future. The key idea behind the system is to profile the temporal variation patterns (TVPs) of malicious domain names. The TVP of a domain name includes information about how and when the domain name has been listed in legitimate/popular and/or malicious domain

name lists. The system actively collects DNS logs, identifies their TVPs, and predicts whether a given domain name will be used for a malicious purpose. Tests using large-scale data reveal that the system can predict which domain names will be used maliciously 220 days beforehand with an extremely high true positive rate (TPR) of 0.985.

Determining optimal countermeasures against malicious domain names To address the problem of malicious domain names with different generated structures, this thesis proposes a new analysis pipeline to determine the most effective countermeasure for each malicious domain name. The pipeline is designed to identify abused domain names and offer defense information by profiling the characteristics of malicious domain names as well as the possible defense solutions and points of defense. Specifically, the pipeline reveals *what, where, and how* countermeasures need to be taken against such malicious domain names. Testing using a large, real dataset revealed that the proposed analysis pipeline is both effective and valid. In particular, the defense information output by the pipeline was confirmed to cause no collateral damage to legitimate accesses.

Identifying malicious HTTP communications To tackle the problem of malicious HTTP communications evading detection, this thesis proposes a new system to achieve more accurate detection of malicious communications caused by malware-infected hosts. The system focuses on the key idea that malicious infrastructures, such as malware samples or command and control servers, tend to be reused instead of created from scratch. Specifically, the system profiles the variability of substrings in HTTP requests, making it possible to identify fixed keywords that can be tied to particular malicious infrastructures and to generate patterns for network-based countermeasures automatically. The results of implementing the system and validating it using real traffic data indicate that it reduces false positives by up to two-thirds compared to the conventional system and even increases the detection rate for infected hosts.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 introduces a new scheme for detecting unseen malicious websites by profiling the characteristics of the IP addresses of websites. The scheme focuses particularly on malicious websites used in recent drive-by download attacks, designs new methods for extracting features from IP addresses, and detects new malicious websites using machine learning classifiers. Chapter 3 presents a new system for detecting domain names with the potential to be malicious in the future. The system profiles the variation of domain names related to cyberattacks over time and is effective at finding potential malicious domain names using large-scale DNS logs and machine learning techniques. Chapter 4 presents a new analysis pipeline that provides the optimal countermeasure for each malicious domain name. The pipeline profiles both the operational characteristics of malicious domain names and possible defense solutions to determine how best to utilize malicious domain name information for taking countermeasures. Chapter 5 proposes a new system for detecting malicious communications precisely. The system profiles the characteristics of reused malicious infrastructure to identify fixed keywords in malicious HTTP requests and generate detection patterns automatically. Finally, Chapter 6 presents the conclusions of this thesis.

Chapter 2

Profiling Spatial Structures of IP Addresses

2.1 Introduction

Web-based malware attacks have become one of the most serious threats that need to be addressed urgently. Some malicious websites steal users' confidential information, which may include login IDs, passwords, and personal information. Other malicious websites enforce users to download malicious software (malware).

Web-based malware attacks target vulnerabilities that exist in web browsers and several plugins such as Flash players, Java VMs, and PDF plugins [17]. These vulnerabilities are exploited by compromising the browser so that malware is downloaded and run on the targeted system. Such attacks are often called *drive-by download* attacks [18].

Computers are subjected to conventional attacks when they are connected to the Internet or external devices such as a USB memory that is infected with malware. In contrast, drive-by download attacks are triggered by users' access to certain websites. Figure 2.1 illustrates the procedure of a typical drive-by download attack. When a browser accesses a compromised *landing* site, the HTTP connection is redirected to a *hopping* site. A hopping site is a website that contains a redirect instruction code that redirects an HTTP connection to the next hopping site or an *exploit* site. After a connection has

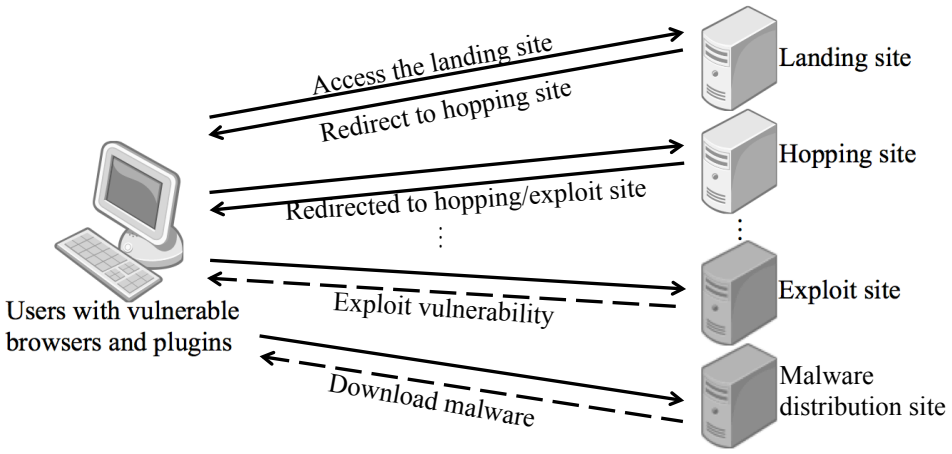


Figure 2.1: Procedure of a drive-by download attack

been redirected to an exploit site, the browser is forced to download malware from a *malware distribution site*. An exploit site is a website that actually exploits vulnerabilities of users' web browsers.

Owing to the complexity of the infection procedure shown above, the detection of infection by web-based malware is often complex. Authors of malware use several techniques that redirect users to actual malware distribution sites by masquerading them as exciting and fun themes on social networking websites or e-mails [19]. The redirection sites can be easily updated. Several intermediate redirection URLs are effective for one-time access only. Moreover, they employ various obfuscation techniques such as encryption, polymorphism, and tunneling to evade detection. Therefore, conventional approaches often fail to detect new attacks owing to the versatility of malicious website deployment, which is discussed in Section 2.2. To resolve this problem, this chapter presents a new scheme for detecting malicious websites using the characteristics of IP addresses.

This chapter proposes a scheme for detecting communication associated with web-based malware using the features extracted from structures of IP addresses in order to prevent users from accessing even *unknown* malicious websites. Our approach leverages the empirical observation that IP addresses are more stable than other metrics such as URLs and DNS records. While

the strings that form URLs or DNS records are highly variable, IP addresses are less variable, i.e., IPv4 address space is mapped onto 4-byte strings. In this chapter, a lightweight and scalable detection scheme that is based on machine learning techniques is developed and evaluated. Note that the goal of this chapter is not to provide a single solution that effectively detects web-based malware but to develop a technique that compensates for the limitations of existing approaches. The effectiveness of our approach is validated by using real IP address data from existing blacklists and real traffic data on a campus network. The results demonstrate that our scheme accurately differentiates between the IP addresses used for benign websites and malicious websites. In addition, this chapter illustrates that our scheme expands the coverage/accuracy of existing blacklists and detects even *unknown* malicious websites that are not covered by conventional approaches.

The rest of this chapter is organized as follows. First, Section 2.2 reviews related work and discusses their limitations. Next, our detection scheme is presented in Section 2.3. Then, Section 2.4 illustrates the experimental results using actual web traffic data collected on a large-scale campus network. Finally, Section 2.5 concludes this chapter.

2.2 Related Work

This section reviews related work and discusses their limitations. The systems proposed in related work are divided into three categories: blacklists and reputation systems, intrusion detection systems (IDS), and client honeypots.

2.2.1 Blacklists and Reputation Systems

So far, blacklists and reputation systems have been the most popular solutions that prevent users from accessing malicious websites. Blacklists can be applied to both network-side and client-side filtering. Network-side filtering can be used with DNS blacklist or blocklist (DNSBL) [20, 21, 22] and com-

mercial security appliances. Client-side filtering can be included in current web browsers [23, 24].

In reputation systems, reputation is based on various types of information present in each IP address or domain name and is applied to prevent users from accessing malicious websites. Criteria for reputation include features retrieved from domain names, WHOIS information, and link structures. Antonakakis et al. [4] developed a dynamic reputation system called *Notos*. The system collects information from multiple sources such as DNS zones, border gateway protocol prefixes, and Autonomous System (AS) information to model network and zone behaviors of benign and malicious domain names. Then, it applies these models to calculate a reputation score for each domain name. Felegyhazi et al. [25] proposed domain-based proactive blacklisting. It utilizes a small set of known malicious domain names to predict other malicious domain names using registration and name server information. Ma et al. [26] proposed a supervised learning approach for classifying URLs as benign or malicious using both lexical structure of URLs and host-based features such as WHOIS records and geographical information. Yadav et al. [27] focused on algorithmically generated malicious domain names and found that such domain names were quite different from legitimate ones. They utilized this characteristic to develop their detection method based on statistical learning.

Although these approaches are widely employed, they often fail to keep up with the transient nature of malicious activities. For instance, it was demonstrated that maintaining up-to-date blacklists is not easily accomplished because new malicious domain names can be easily and continuously generated [28, 29, 8]. Furthermore, it has been reported that attackers frequently change domain names to evade detection by reputation systems [17]. In addition, Shin et al. [30] showed that only 17% of worm/bot victims are covered by several blacklists and they pointed out that better ways to detect future emerging malware are needed.

In summary, existing blacklist-based approaches are prone to failure with

respect to detecting versatile web-based malware.

2.2.2 Intrusion Detection Systems

IDS can be used to block users from accessing malicious websites. It can be classified into two types: signature-based IDS and anomaly-based IDS. Signature-based IDS such as Bro [31] and Snort [32] monitor network traffic and search packets that correspond to predefined attack signatures. Because attack signatures are generated by known attacks, signature-based IDS cannot detect unknown attacks. Anomaly-based IDS learns normal network behavior and uses this information to detect attacks. Therefore, it has the potential for detecting unknown attacks [33]. However, there is a high probability of anomaly-based IDS falsely regarding normal traffic as attacks, namely false positives. Moreover, malicious websites often contain *obfuscated* scripts to evade such IDS. As shown later in Section 2.4.5, detecting malicious websites with IDS is not readily performed in real environments. Therefore, it is recognized that IDS has limitations with respect to blocking all kinds of malicious websites.

In summary, existing IDS-based approaches may fail to detect obfuscated malware.

2.2.3 Client Honeypots

Recent studies have proposed *client honeypot systems* that aim to detect and analyze drive-by download attacks [34, 35, 36, 18, 37]. A client honeypot is a system that crawls websites and detects malicious websites. Client honeypots can be classified into two types: low-interaction honeypots and high-interaction honeypots. Low-interaction honeypots such as HoneyC [34] include an emulator of a browser to crawl websites. Therefore, there is no risk that honeypots themselves will be infected with malware. High-interaction honeypots such as Capture-HPC [35] and BLADE [36] include a real web browser and system; therefore, they can collect more information such as malware behavior after exploitation. One clear drawback of high-interaction

honeypots is the risk of malware infection because they actually run exploit codes. Akiyama et al. proposed a state-of-the-art high-interaction honeypot called *Marionette* [18, 37] that overcomes the risk of malware infection. Marionette has a real vulnerable web browser and plugins that can detect attacks without being infected with malware.

However, client honeypots still have three major problems that affect their detection of all malicious websites. One problem is the lack of scalability. At present, attackers generally deploy a large number of malicious websites [8], whose URLs change within a short time period [29]. Thus, it is not feasible to study the entire set of malicious URLs with client honeypot systems. Another problem is that even if websites are benign while being investigated, they may be attacked afterwards and vice versa. Therefore, it is necessary to shorten the intervals between investigations. However, because of time and cost limitations, it is not feasible to investigate all URLs several times. In addition, malicious websites utilize *cloaking* techniques to hide their malicious contents from particular IP addresses used by honeypots [38]. This cloaking makes it more difficult for client honeypots to crawl and detect malicious websites [17].

In summary, existing client honeypot-based approaches have some problems: lack of scalability and versatility, and failure to detect cloaking techniques.

2.3 Detection Scheme

In this section, a high-level overview of our detection scheme is presented first in Section 2.3.1. Next, the effectiveness of our approach is shown in Section 2.3.2. Then, Section 2.3.3 illustrates several feature extraction techniques. Finally, Section 2.3.4 shows how a machine learning approach can be applied to our detection scheme.

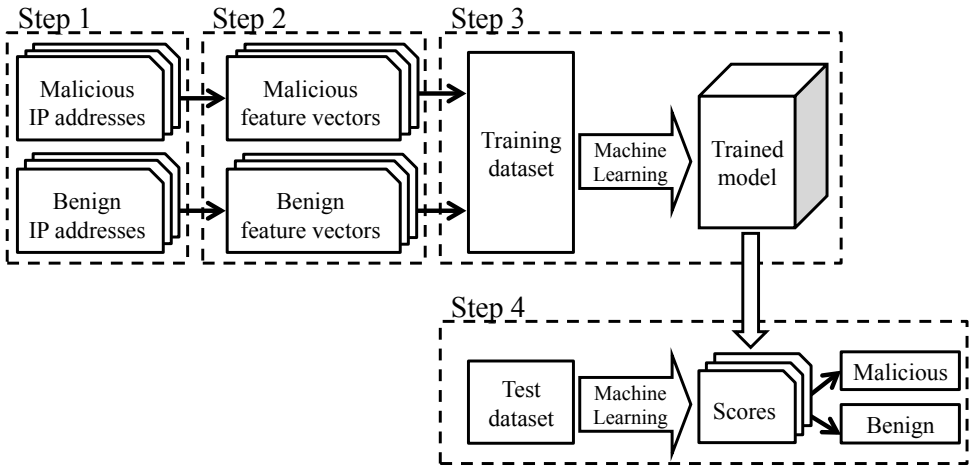


Figure 2.2: Overview of our detection scheme

2.3.1 High-level Overview

Our detection scheme is based on two intrinsic characteristics of IP addresses: (1) stability with time [39] and (2) address space skewness [40, 41]. First, although attackers can change URLs and DNS records at a low cost, it is much more difficult to change IP addresses, which are essentially associated with malicious activities. Thus, the characteristics of an IP address should be more stable compared with other metrics. Second, IP addresses associated with malicious activities are likely to be concentrated in certain network address spaces, as reported by previous measurement studies [40, 41]. In the following Section 2.3.2, our preliminary experiments show that IP addresses of web-based malware also have these characteristics. To the best of our knowledge, the approach presented here is the first IP address-based approach against malicious websites that employ drive-by download attacks. On the other hand, approaches against botnets, phishing, and spam mails have been widely studied, as in [39, 40, 41].

Basically, our approach blocks users' access to malicious websites by extending existing blacklists. The unique feature of our scheme is that it can evaluate IP addresses that are not listed in existing blacklists. To complicate analysis and detection attempts, drive-by download attacks lead users

Table 2.1: Training dataset

Data	Period	# URLs	# IP addresses
TRN_B	Apr 30, 2011	10,000	10,372
TRN_M	Jan 1, 2009–Apr 30, 2011	63,694	14,171

to an actual attacking website via multiple stepladder sites by redirecting users' browsers many times. Therefore, blocking access to the IP address of a destination malicious website can protect users from malware infection.

Our detection scheme involves the following four steps: 1) collecting IP addresses, 2) extracting feature vectors, 3) building a trained model, and 4) detecting malicious IP addresses. Our key technical contribution is building a novel feature extraction methods (step 2), which is described in Section 2.3.3. Section 2.3.4 shows our supervised machine learning technique that is employed for step 3 and step 4. Figure 2.2 outlines our detection scheme.

2.3.2 Effectiveness of IP Address-based Approach

This section illustrates the effectiveness of our IP address-based approach to detect malicious websites. Two preliminary experiments using real IP address data are conducted to show stability with time in malicious networks and address space skewness.

Training Dataset

To evaluate the effectiveness of IP address-based approaches, IP addresses of both benign and malicious websites are collected. Table 2.1 shows the collected training dataset. Note that this training dataset is also used for building a trained model in step 3, which is described later in Section 2.3.4.

Our benign training dataset TRN_B comprises URLs of the top 10,000 websites on the Alexa traffic ranking list [42] on April 30, 2011. From these URLs, 10,372 IP addresses are resolved. Domain names in the ranking list include those to which multiple IP addresses are assigned for load balancing, using DNS round robin and content delivery networks (CDNs). Therefore,

Table 2.2: Top 5 malicious AS in TRN_M

AS	# URLs	# IP addresses
AS #1	1,389	482
AS #2	2,422	400
AS #3	1,061	355
AS #4	761	280
AS #5	1,047	275

the number of IP addresses in the ranking list exceeds the number of URLs, which correlate with domain names. The Alexa ranking is calculated from a combination of the average number of daily visitors and page views during the month [42]. Therefore, it contains both benign sites and less benign sites such as pornographic and file-sharing sites.

Our malicious training dataset TRN_M consists of IP addresses selected from the malware domain list (MDL) [43]. Note that MDL contains some malicious websites on web hosting servers, which utilize the same IP address for multiple domain names. Therefore, the number of URLs is greater than that of IP addresses. TRN_M contains 14,171 unique malicious IP addresses collected over a period of more than two years from January 1, 2009 to April 30, 2011.

Stability with Time in Malicious Networks

To evaluate stability with time in malicious networks, malicious IP addresses of TRN_M as shown in Table 2.1 are analyzed. In this preliminary experiment, IP addresses are treated per AS. Table 2.2 shows analyzed AS, whose AS numbers are masked for security. From the day when IP addresses are observed in TRN_M, the elapsed days since Jan 1, 2009 are calculated. Figure 2.3 represents the cumulative distribution function (CDF) of IP addresses observed in each AS. This result illustrates that malicious IP addresses tend to be contained in certain AS. Moreover, such AS is continuously used for malicious activities for over two years.

Unlike IP addresses, malicious URLs change within a short time pe-

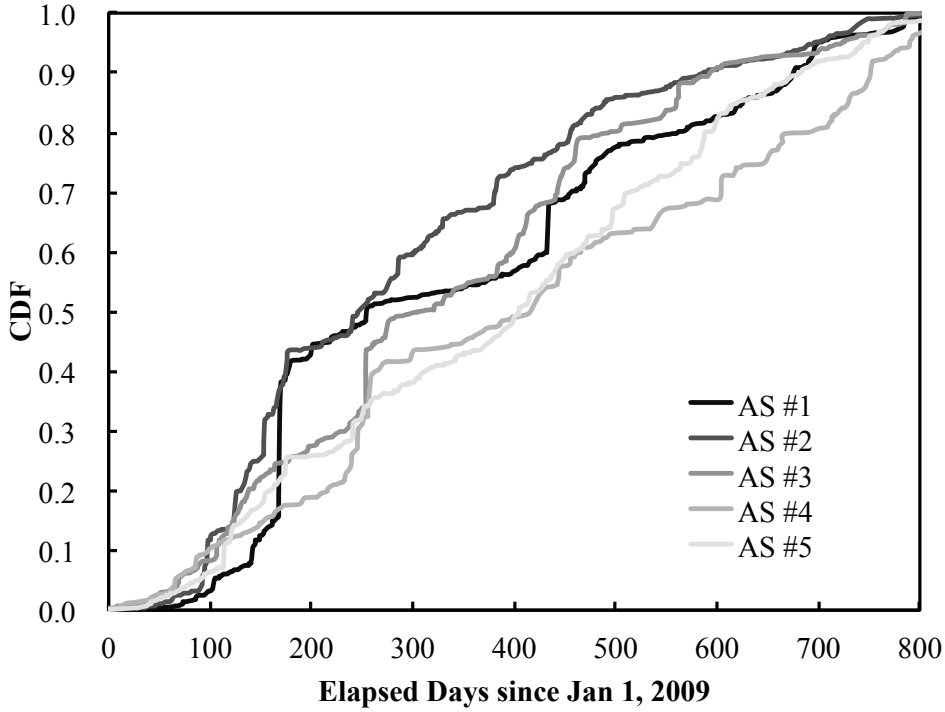


Figure 2.3: Lifetime of malicious AS in TRN_M

riod [29]. For example, more than 60% of URLs contained in TRN_M or the malware domain list (MDL) were active for less than one month [8]. Furthermore, attackers create a lot of new malicious URLs one right after the other to avoid being blacklisted [17, 8]. Our analysis indicates that IP addresses are more stable than URLs in terms of time, i.e. IP addresses have less variability over time.

Address Space Skewness

In this preliminary experiment, IP addresses are projected on a Hilbert curve [44] to visually confirm the locality of malicious IP addresses. Hilbert curve is a recursively defined space-filling curve. A space-filling curve maps points on to a d-dimensional space so that the closeness among the points is preserved. In this study, any consecutive IP addresses will be projected onto a single contiguous part on the curve [40, 45]. Figure 2.4 shows an

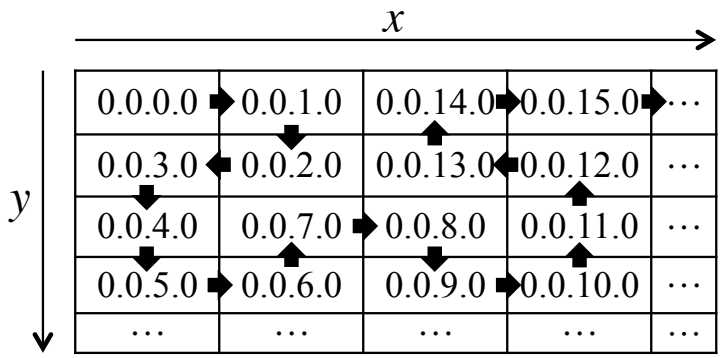


Figure 2.4: Example of IP address mapping on a Hilbert curve

example of IP address mapping on a Hilbert curve. In Fig. 2.4, the squares are ordered according to the Hilbert curve where a single square represents a /24 network block. This technique has also been applied in some existing research to visualize IP address positions of the entire IPv4 address space. For example, Hao et al. [40] utilized the Hilbert curve to display the IP addresses of spam mail senders and Cai et al. [45] showed block-level IP address usage patterns using the Hilbert curve. Figure 2.5 visualizes the IP addresses of `A.B.0.0/16`, the first and the second octets of which are masked with A and B for security and projected into the Hilbert space using the training dataset in Table 2.1. The gray square represents a benign /24 network block, and the black square represents a malicious /24 network block. Figure 2.5 demonstrates that the IP addresses of malicious websites are concentrated in certain network blocks.

2.3.3 Feature Extraction Methods

As described in Fig. 2.2, step 2 projects an IP address space onto a feature space that reflects the relationship between IP addresses and the network structure for interpretation by machine learning algorithms. This chapter proposes three methods to extract a feature vector from an IP address: *octet-based extraction (Octet)*, *extended octet-based extraction (ExOctet)*, and *bit string-based extraction (Bit)*.

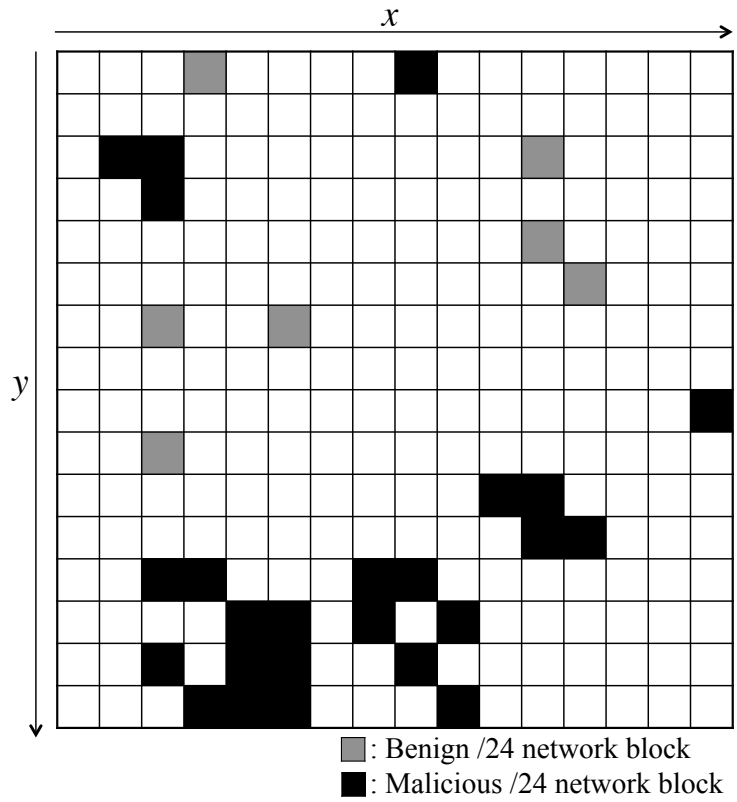


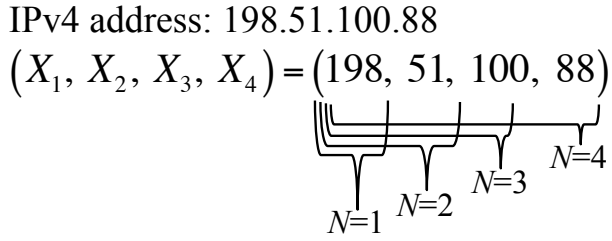
Figure 2.5: Visualization of IP addresses on `A.B.0.0/16`

Octet-based Extraction (Octet): Octet constructs an $M = 2^8 \times N$-dimensional feature vector represented as a sparse bit sequence $\{b_0, \cdots, b_{M-1}\}$ from the most significant N octets of an IPv4 address, where N is a natural number, ranging between one and four that is used as a parameter. The initial value for each bit $\{b_0, \cdots, b_{M-1}\}$ is zero. A feature vector is represented by the following equation:

$$\begin{cases} b_k = 1 & (k \ in \ \bigcup_{n=1}^{N} \{2^8 \cdot (n-1) + X_n\}) \\ b_k = 0 & (otherwise), \end{cases}$$

where k is an index set of feature vectors.

Let $b_{2^8(n-1)+X_n} = 1$ when the n-th $(1 \leq n \leq N)$ octet of an IPv4 address is represented as X_n $(0 \leq X_n \leq 2^8 - 1)$ in decimal notation. Figure 2.6 shows an example of feature vectors corresponding to the IPv4 address `198.51.100.88`

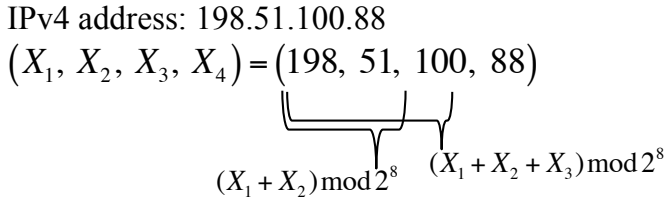


N	n-th octet	Feature vector	M
1	1	$b_k = 1\ (k = 198)$ $b_k = 0\ (otherwise)$ $(0 \le k \le 255)$	256
2	1, 2	$b_k = 1\ (k = 198, 307)$ $b_k = 0\ (otherwise)$ $(0 \le k \le 511)$	512
3	1, 2, 3	$b_k = 1\ (k = 198, 307, 612)$ $b_k = 0\ (otherwise)$ $(0 \le k \le 767)$	768
4	1, 2, 3, 4	$b_k = 1\ (k = 198, 307, 612, 856)$ $b_k = 0\ (otherwise)$ $(0 \le k \le 1023)$	1024

Figure 2.6: Example of Octet-based Extraction (Octet)

with Octet. The upper half of Fig. 2.6 illustrates the correspondence between the parameter N and feature extraction coverage. A feature vector uses the first octet of an IP address when $N = 1$; the first and second octets when $N = 2$; the first, second, and third octets when $N = 3$; and the first, second, third, and fourth octets when $N = 4$. The lower half of Fig. 2.6 lists the feature vectors extracted by parameter N. For example, when $N = 3$, k consists of 198, 307 $(= 256 + X_2)$, and 612 $(= 512 + X_3)$.

Extended Octet-based Extraction (ExOctet): ExOctet extends Octet's feature vector to construct an $M = 2^8 \times (N + 2)$-dimensional feature vector represented as a sparse bit sequence $\{b_0, \cdots, b_{M-1}\}$ from the most significant N octets of an IPv4 address, where N is a natural number greater than or equal to three. The initial value for each bit $\{b_0, \cdots, b_{M-1}\}$ is zero. A feature



N	n-th octet	Feature vector	M
3	1, 2, 3	$b_k = 1 \ (k = 198,\ 307,\ 612)$ $b_k = 1 \ (k = 2^8 \cdot 3 + (198 + 51) \bmod 2^8)$ $b_k = 1 \ (k = 2^8 \cdot 4 + (198 + 51 + 100) \bmod 2^8)$ $b_k = 0 \ (otherwise)$ $(0 \le k \le 1279)$	1280

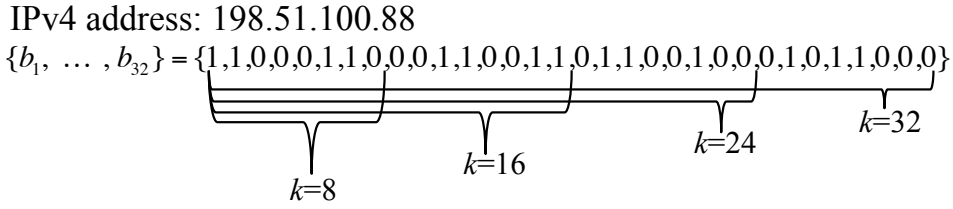
Figure 2.7: Example of Extended Octet-based Extraction (ExOctet)

vector is represented according to the following equation:

$$\begin{cases} b_k = 1 & (k \ in \ \bigcup_{n=1}^{N} \{2^8 \cdot (n-1) + X_n\}) \\ b_k = 1 & (k \ in \ \bigcup_{m=N\geq 3}^{N+1} \{2^8 \cdot m + (\sum_{i=1}^{m-1} X_i) \bmod 2^8\}) \\ b_k = 0 & (otherwise). \end{cases}$$

Let $b_{2^8(n-1)+X_n} = 1$ when the n-th ($1 \le n \le N$) octet of an IPv4 address is represented as X_n ($0 \le X_n \le 2^8 - 1$) in decimal notation. ExOctet adds $b_{2^8 \cdot 3 + (X_1 + X_2) \bmod 2^8} = 1$ and $b_{2^8 \cdot 4 + (X_1 + X_2 + X_3) \bmod 2^8} = 1$ when $N = 3$. Figure 2.7 shows an example of feature vectors corresponding to the IPv4 address `198.51.100.88` with ExOctet. The upper half of Fig. 2.7 illustrates the correspondence of the extended coverage that is different from that of Octet. The lower half of Fig. 2.7 lists feature vectors extracted when $N = 3$. A feature vector uses the first, second, and third octets of an IP address when $N = 3$, which is the same as Octet. Moreover, a feature vector is extended with a combination of the first and second octets, and the first, second, and third octets.

Bit String-based Extraction (Bit): Bit constructs a k-dimensional feature vector represented as a bit sequence $\{b_1, \cdots, b_k\}$ from a 32-bit binary



k	Feature vector
8	$b_k = 1\ (k = 1,2,6,7)$; $b_k = 0\ (otherwise)$ $(1 \le k \le 8)$
16	$b_k = 1\ (k = 1,2,6,7,11,12,15,16)$; $b_k = 0\ (otherwise)$ $(1 \le k \le 16)$
24	$b_k = 1\ (k = 1,2,6,7,11,12,15,16,18,19,22)$; $b_k = 0\ (otherwise)$ $(1 \le k \le 24)$
32	$b_k = 1\ (k = 1,2,6,7,11,12,15,16,18,19,22,26,28,29)$; $b_k = 0\ (otherwise)$ $(1 \le k \le 32)$

Figure 2.8: Example of Bit String-based Extraction (Bit)

form of the IPv4 address, where k is a natural number used as a parameter. The value for each bit $\{b_1, \cdots, b_k\}$ is equivalent to the first k bits of a binary-formatted IPv4 address. A feature vector is represented according to the following equation:

$$\begin{cases} b_k = 1 & (\text{when the } k\text{-th bit value of IPv4 address is 1}) \\ b_k = 0 & (otherwise). \end{cases}$$

Let $b_k = 1$ when the k-th bit value of binary-formatted IPv4 address is 1. Figure 2.8 shows an example of feature vectors corresponding to the IPv4 address `198.51.100.88` with Bit. The upper half of Fig. 2.8 illustrates the correspondence between parameter k and feature extraction coverage, and the lower half lists the feature vectors extracted by parameter k.

Table 2.3: Example of a training dataset

Label t_n	IP address	Feature vector $\boldsymbol{x}_n$
$+1$	`198.51.100.88`	$\{1,1,0,0,0,1,1,0,0,0,1,1,0,0,1,1\}$
-1	`192.0.2.1`	$\{1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0\}$
-1	`203.0.113.24`	$\{1,1,0,0,1,0,1,1,0,0,0,0,0,0,0,0\}$
$\cdots$	$\cdots$	$\cdots$

2.3.4 Application of Machine Learning

As shown in Fig. 2.2, step 3 applies machine learning to the feature vectors extracted in step 2. Several options exist for machine learning classifiers such as k-nearest neighbors (k-NN), neural networks, and support vector machines (SVMs) [46]. K-NN are methods for classifying test data based on proximity between test data and training data in the feature space. Due to their high calculation complexity, k-NN are not suitable for high-speed classification [46]. Also, typical k-NN cannot accurately handle high-dimensional data such as our IP address-based feature spaces. Neural networks, especially feedforward multi-layer neural networks, are computational models that can be used for data classification. Neural networks need a lot of parameters to perform and they cannot find global optimum solutions [46]. SVM is a popular machine learning method for classification. Unlike k-NN, SVM can appropriately handle high-dimensional feature spaces. Moreover, the number of parameters in SVM is small and SVM can find a global optimum solution [46, 47]. In addition, it has been reported that SVM works very well for various problems in a lot of areas [47]. Our scheme is intended to be an accurate and lightweight detection method. Therefore, SVM is selected to demonstrate the effectiveness of our approach. As shown in Section 2.4, our detection scheme with SVM works successfully for real datasets. Note that our scheme is generic and can leverage any type of classifier that fits our problem formulation. A key contribution of this chapter is the building of effective feature vectors that can be input into supervised classifiers.

Table 2.3 shows an example of a training dataset used in this study. The feature vectors in this example are extracted using Bit when $k = 16$. The

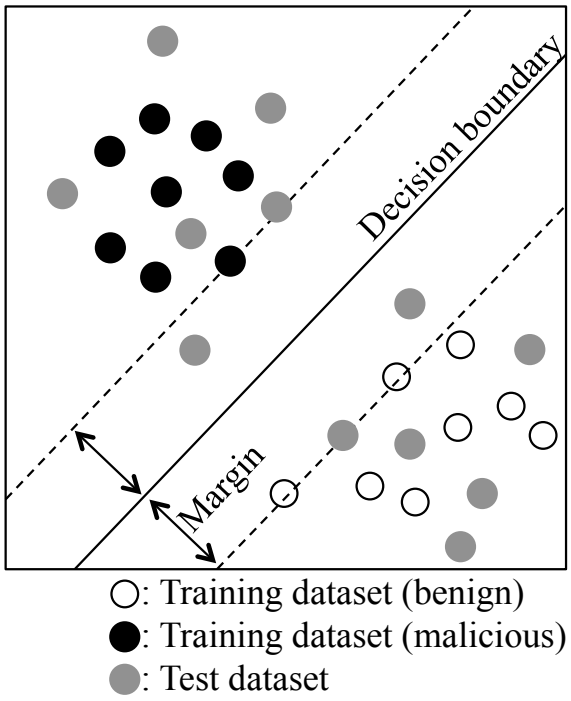


Figure 2.9: Conceptual image of SVM's feature space

training dataset comprises N input vectors $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ with corresponding target label values $t_1, \cdots, t_N$, where $t_n \in \{-1(\text{benign}), +1(\text{malicious})\}$.

Now the way to train SVM classifiers using the training dataset is described below. SVM uses the concept of a *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples. The decision boundary is chosen to be the one whose margin is maximized. Figure 2.9 illustrates a conceptual image of SVM's feature space. The discrimination function of SVM is defined as follows:

$$y(\mathbf{x}) = \mathbf{w}^{\mathbf{T}}\phi(\mathbf{x}) + \beta,$$

where $\mathbf{w}$ denotes the parameter to move the decision boundary, $\mathbf{w}^{\mathbf{T}}$ is the transposed matrix of $\mathbf{w}$, $\phi(\mathbf{x})$ denotes the feature space transformation function, and β is a bias parameter. Using the discrimination function above,

the label C of a sample with feature vector $\mathbf{x}$ can be inferred as

$$\widehat{C} = \begin{cases} 1 & (y > 0) \\ -1 & (y < 0). \end{cases}$$

In SVM, parameters $\mathbf{w}$ and β are trained so that margins are maximized. The optimization problem is formulated as follows:

$$\text{argmax}_{\mathbf{w},\beta} \left\{ \frac{1}{|\mathbf{w}|} \min_n \left[t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + \beta) \right] \right\}.$$

To numerically derive the solutions, sequential minimal optimization [48] is applied. For the kernel function, our scheme selects the Gaussian kernel: $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) = e^{-\gamma||\mathbf{x}-\mathbf{y}||}$. To obtain the best parameters, γ and β, our scheme performs grid search with cross-validation tests. To estimate the probability of SVM's binary output, the technique proposed by Platt in [49] is applied. The key idea is to apply the logistic sigmoid function to the discrimination function of SVM, $y(\mathbf{x})$. The *score* is calculated as follows:

$$score = P(t = 1|\mathbf{x}) = \sigma(Ay(\mathbf{x}) + B), \tag{2.1}$$

where $\sigma(a)$ is the logistic sigmoid function defined by $\sigma(a) = 1/(1+\exp(-a))$. The values for parameters A and B are found by minimizing the cross-entropy error function defined by a training dataset consisting of pairs of values $y(\mathbf{x}_n)$ and t_n. The scores assigned to each IP address are used for controlling the risk of various errors, which are discussed in Section 2.4.4.

2.4 Experiments

This section illustrates the experimental results using real IP addresses data obtained from existing blacklists and actual web traffic data collected on a large-scale campus network.

2.4.1 Test Dataset

Test dataset is created from both benign and malicious websites, which include web traffic data captured on a campus network. Table 2.4 shows the test dataset used to evaluate our scheme.

Table 2.4: Test dataset

Data	Period	# URLs	# IP addresses
TST_B	May 1, 2011–May 14, 2011	96,597	57,190
TST_M_ACTIVE	May 1, 2011–May 14, 2011	11,223	2,450
TST_M_NEW	May 1, 2011–May 14, 2011	455	161

Our benign test dataset TST_B consists of HTTP traffic data captured on a campus network for two weeks in May 2011. The campus network is a production network with /16 prefix lengths and is used by more than 50,000 students and faculty members. The average throughput of the campus traffic is approximately 300-400 Mbps, and that of HTTP traffic is up to about 25 Mbps. To minimize the probability that malicious websites are contained in TST_B, all URLs in the captured data are checked with the Google Safe Browsing API [24]. Google Safe Browsing is constantly updated with blacklists of suspected phishing and malware related websites. As a result, 2,515 URLs that are suspected of being malicious sites are excluded.

Our malicious test dataset consists of IP addresses selected from the malware domain list (MDL) [43], which is the same data collection source as our malicious training dataset TRN_M. Note that the collection period of our malicious test dataset is different from that of TRN_M. The malicious test dataset are divided into two subsets: TST_M_ACTIVE and TST_M_NEW. TST_M_ACTIVE contains 2,450 unique active malicious IP addresses observed for two weeks from May 1 to May 14, 2011. TST_M_NEW consists of 161 malicious IP addresses that are unknown to the trained model. To evaluate the generalization ability of the trained model, any IP addresses that are also contained in TRN_M are eliminated from TST_M_NEW. Moreover, from TST_M_NEW, any IP addresses for which the top three octets are equal to those of an IP address contained in TRN_M are eliminated. Therefore, TST_M_NEW contains IP addresses that have *never* been observed. They can be seen as the IP addresses that are *not* covered by blacklists, yet.

Table 2.5: Relationships among terms

		Actual value	
		Malicious	Benign
Prediction outcome	Malicious	True Positive (TP)	False Positive (FP)
	Benign	False Negative (FN)	True Negative (TN)

2.4.2 Evaluation Method

In the following experiments, the training dataset shown in Table 2.1 is used for the SVM's trained model as explained in Section 2.3.4. Then, our scheme is evaluated with the test dataset of Table 2.4.

This chapter defines the correct classification of an actually malicious IP address into a malicious category as a true positive (TP), the incorrect classification of an actually benign IP address into a malicious category as a false positive (FP), the incorrect classification of an actually malicious IP address into a benign category as a false negative (FN), and the correct classification of an actually benign IP address into a benign category as a true negative (TN). Table 2.5 shows the relationships among these terms.

Our scheme is evaluated using the following criteria: accuracy, false positive rate (FPR), false negative rate (FNR), receiver operator characteristic (ROC) curve, and area under the curve (AUC). Accuracy, FPR, and FNR are calculated as follows:

$$\begin{aligned} Accuracy &= (TP+TN)/(TP+FP+FN+TN) \\ FPR &= FP/(FP+TN) \\ FNR &= FN/(FN+TP). \end{aligned}$$

The ROC curve is a graphical plot of the true positive rate ($TPR = TP/(TP+FN)$) vs. FPR for every possible cut-off point. The cut-off point relates to *score*, as described in Section 2.3.4. Each point on the ROC curve represents a (FPR, TPR) pair corresponding to a particular decision cut-off point. Therefore, if the curve rises rapidly towards the upper left corner, it means that the overall test result is good. AUC is the area under the ROC curve

Table 2.6: Detection performance with the test dataset combination TST_B and TST_M_ACTIVE

Method	Accuracy	AUC	FPR	FNR
Octet ($N = 1$)	0.751	0.788	0.253	0.151
Octet ($N = 2$)	0.862	0.878	0.138	0.140
Octet ($N = 3$)	0.885	0.989	0.119	0.024
Octet ($N = 4$)	0.860	0.988	0.145	0.016
ExOctet ($N = 3$)	0.905	0.994	0.098	0.020
Bit ($k = 8$)	0.751	0.790	0.253	0.151
Bit ($k = 16$)	0.847	0.874	0.154	0.136
Bit ($k = 24$)	0.857	0.979	0.148	0.026
Bit ($k = 32$)	0.835	0.991	0.172	0.017

that is used to score a binary classifier. AUC is calculated as follows:

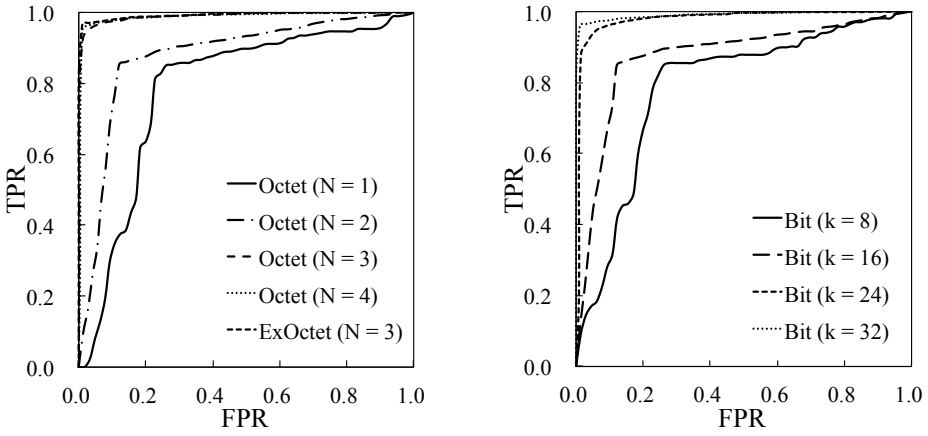
$$AUC = \sum_{i \in \Omega} \delta_i FNR(i),$$

where δ_i is $FPR(i+1) - FPR(i)$, and $FPR(i)$ and $FNR(i)$ are the false positive rate and false negative rate for the i th parameter setting, respectively. Ω is the parameter space to be explored. Note that $FPR(i)$ is sorted in ascending order. AUC ranges from 0.0 (worst) to 1.0 (best).

2.4.3 Experiment 1: Comparing the Detection Performance of Feature Extraction Methods

This experiment evaluates the detection performance of our feature extraction methods. As an implementation of SVM, our scheme uses LIBSVM [47], which is currently one of the most widely used SVM software tool for many disciplines. In this experiment, $0.0 \leq score < 0.5$ is considered as benign and $0.5 \leq score \leq 1.0$ as malicious, which is equivalent to the cut-off point 0.5. This is the default configuration of binary classification using SVM. Our scheme is evaluated with two kinds of test dataset combinations.

The first set of tests was conducted with the test dataset combination TST_B and TST_M_ACTIVE. Table 2.6 shows the evaluation results and



(a) Octet and ExOctet (b) Bit

Figure 2.10: ROC curves of detection performance with the test dataset combination TST_B and TST_M_ACTIVE.

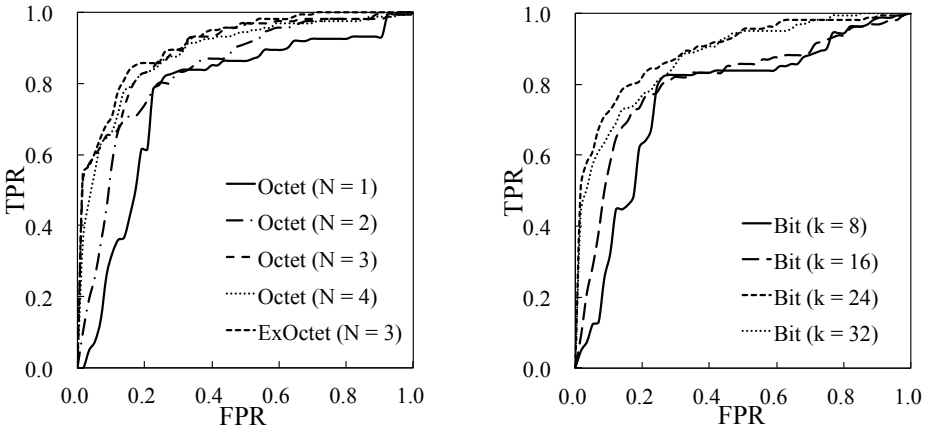
Fig. 2.10 shows the ROC curves. In the Octet method, the accuracy increases to 0.885, AUC increases to 0.989, and FPR decreases to 0.119, as N increases to 3, where N is the parameter for the Octet method. However, when $N = 4$, the accuracy and AUC decrease and FPR increases. FNR decreases linearly with increasing N. ROC curves improve as N increases to 3. In particular, the ROC curves for both $N = 3, 4$ are closer to the upper left corner and have the potential to be ideal for binary classification. In the ExOctet method, which prioritizes the upper octets of IP addresses, the best results obtained are as follows: accuracy of 0.905, AUC of 0.994, and FPR of 0.098. The obtained ROC curve is also better than those of all other methods. In the Bit method, the accuracy increases to 0.857 and FPR decreases to 0.148 as k increases to 24, where k is the parameter for the Bit method. However, when $k = 32$, the accuracy decreases and FPR increases. AUC increases linearly with increasing k, whereas FNR decreases linearly with increasing k. The ROC curves improve as k increases and the curves obtained when $k = 24, 32$ are excellent. These test results indicate that the top three octets of IP addresses prove to be effective in discriminating between benign and malicious websites. We believe that the reasons are (1) IP address allocation

Table 2.7: Detection performance with the test dataset combination TST_B and TST_M_NEW

Method	Accuracy	AUC	FPR	FNR
Octet ($N = 1$)	0.747	0.768	0.253	0.180
Octet ($N = 2$)	0.861	0.834	0.138	0.304
Octet ($N = 3$)	0.880	0.897	0.119	0.335
Octet ($N = 4$)	0.855	0.885	0.145	0.217
ExOctet ($N = 3$)	0.902	0.914	0.098	0.292
Bit ($k = 8$)	0.747	0.762	0.253	0.180
Bit ($k = 16$)	0.846	0.804	0.154	0.304
Bit ($k = 24$)	0.851	0.893	0.148	0.205
Bit ($k = 32$)	0.828	0.878	0.172	0.261

policies [50] provide locality to IP addresses and (2) most of the networks are based on the /24 subnet.

The second set of tests was conducted with the test dataset combination of TST_B and TST_M_NEW. Table 2.7 shows the evaluation results and Fig. 2.11 shows the ROC curves. Note that the TST_M_NEW dataset does *not* contain any IP addresses that are used in the training stage as *known* IP addresses. This chapter first notices that our scheme successfully detects *unknown* malicious IP addresses that could be missed by existing blacklists. Now this test evaluates the performance in detecting unknown malicious data. In the Octet method, the accuracy increases to 0.880, AUC increases to 0.897, FPR decreases to 0.119, and FNR increases to 0.335 as parameter N increases to 3. However, when $N = 4$, the accuracy, AUC, and FNR decrease and FPR increases. The ROC curves improve as N increases to 3. However, when $N = 4$, the ROC curve worsens slightly. In the ExOctet method, an accuracy of 0.902, AUC of 0.914, and FPR of 0.098 are the best results. The ROC curve is also the best result among all the methods. In the Bit method, the accuracy increases to 0.851, AUC increases to 0.893, and FPR decreases to 0.148 as parameter k increases to 24. However, when $k = 32$, the accuracy and AUC decrease and FPR increases. FNR is the

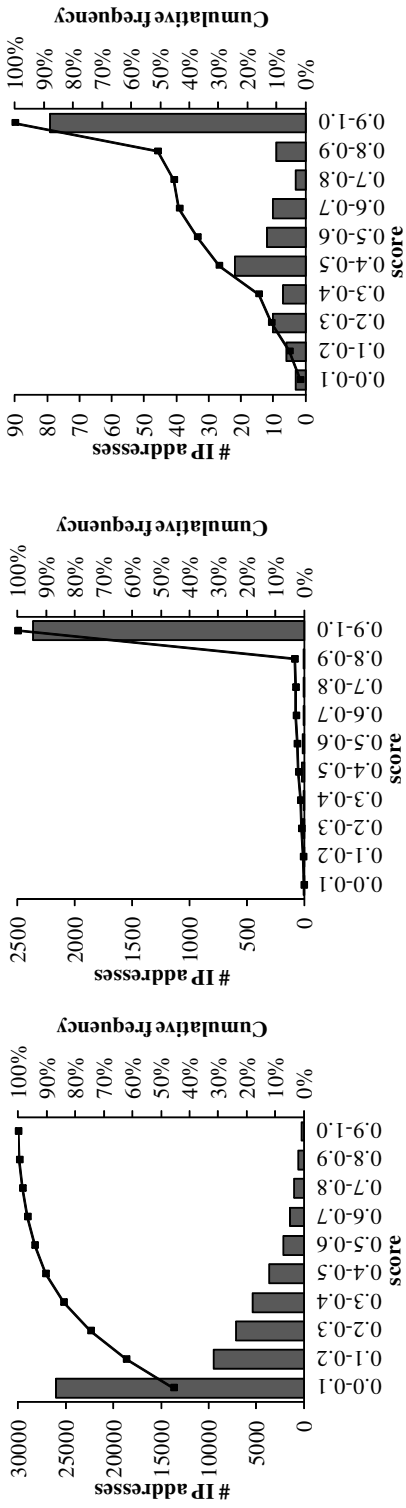


(a) Octet and ExOctet (b) Bit

Figure 2.11: ROC curves of detection performance with the test dataset combination TST_B and TST_M_NEW.

lowest when $k = 24$, whereas the ROC curve is the best when $k = 24$. The reason for the best results when $N = 3$ or $k = 24$ is considered to be the same as that of the first set of tests, i.e., most of the networks are based on the /24 subnet.

For comparing the evaluation results of the two kinds of test dataset combinations, FNR with tests using TST_M_NEW dataset is higher than that using TST_M_ACTIVE. Furthermore, the ROC curves for the test using TST_M_NEW is worse than that obtained using TST_M_ACTIVE. This is because TST_M_NEW contains only IP addresses that are sufficiently valid for the evaluation of our scheme, as described in Section 2.4.1. It should be noted that our scheme can classify unknown malicious IP addresses with high accuracy and low FPR. These results prove that IP addresses are effective indicators for determining whether a website is malicious.



(a) TST_B (b) TST_M_ACTIVE (c) TST_M_NEW

Figure 2.12: Histograms of scores with the test dataset TST_B, TST_M_ACTIVE, and TST_M_NEW.

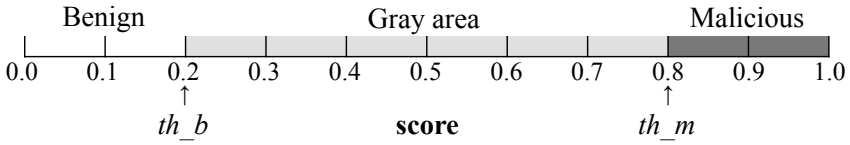


Figure 2.13: Example of the relationship between score and two kinds of thresholds.

2.4.4 Experiment 2: Evaluating the Distribution of the Score

This experiment evaluates the distribution of the scores assigned to IP addresses by our scheme. Figure 2.12 shows histograms with cumulative frequency curves illustrating the distribution of scores for each test dataset: TST_B, TST_M_ACTIVE, and TST_M_NEW, with the ExOctet ($N = 3$) model showing the best results in Section 2.4.3.

Figure 2.12 shows that for TST_B, our scheme assigns low scores to benign IP addresses. Moreover, for TST_M_ACTIVE in Fig. 2.12, our scheme gives significantly high scores to malicious IP addresses that were active during the observation term. This means that our scheme can accurately determine whether an IP address is malicious. TST_M_NEW in Fig. 2.12 shows that our scheme can assign high scores to even unknown malicious IP addresses. As described in Section 2.4.1, the first, second, and third octets of IP addresses in TST_M_NEW do not match any IP addresses in the training dataset TRN_M. This indicates that the IP addresses in TST_M_NEW first appeared during the observation period.

Now, our scheme introduces multiple thresholds to control the risk of incorrect classification. In a real environment, the risk should be configurable under various security policies such as minimizing the number of false positives and false negatives. This chapter discusses a way to follow these security policies. For example, it is possible to decrease the number of false positives and false negatives by setting two kinds of thresholds: th_b and th_m. th_b determines whether an IP address is benign and th_m determines whether

Table 2.8: Setting two kinds of thresholds with the test dataset combination TST_B and TST_M_ACTIVE

th_b	th_m	# IP addresses in gray area	# IP addresses of false positives	# IP addresses of false negatives
0.5	0.5	0	5,476	50
0.4	0.6	5,824	3,282	33
0.3	0.7	12,696	1,847	24
0.2	0.8	20,840	842	9
0.1	0.9	30,965	233	3

Table 2.9: Setting two kinds of thresholds with the test dataset combination TST_B and TST_M_NEW

th_b	th_m	# IP addresses in gray area	# IP addresses of false positives	# IP addresses of false negatives
0.5	0.5	0	5,476	48
0.4	0.6	5,829	3,282	26
0.3	0.7	12,701	1,847	19
0.2	0.8	20,839	842	9
0.1	0.9	30,964	233	3

an IP address is malicious. An IP address whose score is lower than or equal to th_b is considered benign, whereas an IP address whose score is greater than or equal to th_m is considered malicious. An IP address whose score is between th_b and th_m is classified as being in a *gray* area. Figure 2.13 shows an example of the relationship between the score and the two thresholds when $th\_b = 0.2$ and $th\_m = 0.8$, i.e., $0.2 < score < 0.8$ is the gray area.

Table 2.8 shows the result when setting the two kinds of thresholds for the test dataset combination TST_B and TST_M_ACTIVE. Table 2.9 shows the result with the two kinds of thresholds for the test dataset combination TST_B and TST_M_NEW. Table 2.8 and Table 2.9 show that there were 5,476 false positives when $th\_b = th\_m = 0.5$. In this case, $th\_b = th\_m$ is equivalent to setting a single threshold without any gray areas. Extending the gray area decreases the number of IP addresses that result in false positives or false negatives. As an example, compare the cases between multiple thresh-

Table 2.10: Snort alerts in benign dataset TST_B

Snort signature	# Alerts
`SHELLCODE x86 inc ecx NOOP`	272,180
`SHELLCODE x86 NOOP`	811
`SHELLCODE base64 x86 NOOP`	585
`SHELLCODE x86 inc ebx NOOP`	425
`SHELLCODE x86 setuid 0`	18
`SHELLCODE x86 setgid 0`	11
# total Snort alerts	274,030
# flows inspected by Snort	145,985,724

olds ($th\_b = 0.2$ and $th\_m = 0.8$) and a single threshold ($th\_b = th\_m = 0.5$). When using multiple thresholds, our scheme does not determine whether the IP address for which the score is $0.2 < score < 0.8$ is benign or malicious but classifies it as being in the gray area. This decreases the likelihood of false positives and false negatives compared to the case of a single threshold; however, there are a number of gray IP addresses that cannot be determined as being benign or malicious. These results indicate that our scheme can control these two kinds of thresholds to decrease the number of false positives and false negatives.

2.4.5 Experiment 3: Applying Conventional IDS to Our Data

In this experiment, conventional IDS is applied to two kinds of data. The purpose of this experiment is to compare the detection performance of our scheme with that of conventional IDS to demonstrate the superiority of the proposed scheme. As a conventional IDS, the latest Snort [32] and its rule sets are selected. Snort is one of the most famous open source signature-based IDS. The Snort signatures applied in this experiment consisted of only eight *Priority 1* signatures as shown in Table 2.10 and Table 2.11.

The first test was conducted with our benign dataset TST_B. As mentioned in Section 2.4.1, TST_B contains benign HTTP traffic data. However, as shown in Table 2.10, a number of false positive alerts were output from

Table 2.11: Snort alerts in malicious dataset D3M

Snort signature	# Alerts
`SHELLCODE x86 NOOP`	40
`SHELLCODE x86 inc ebx NOOP`	1
`SHELLCODE x86 inc ecx NOOP`	1
`POLICY Suspicious .cn DNS query`	2
# total Snort alerts	44
# URLs in D3M data	840
# flows inspected by Snort	11,393

Snort. In this case, false positive means the incorrect classification of actually benign traffic into the malicious category by Snort. Table 2.10 lists many shellcode-related alerts. The reasons for these alerts are broadly classified into two categories: upload of executable files and HTTP compression. Upload of executable files to online file storage services or web-based e-mail services is considered a shellcode-related alert by Snort. HTTP compression, which reduces the file size of HTTP data by using gzip and deflate algorithms [51], also outputs shellcode-related alerts.

The second test was conducted with the *D3M 2010* dataset from *MWS 2010* [52] and the *D3M 2011* dataset from *MWS 2011* [53]. These datasets were provided by the *anti Malware engineering WorkShop (MWS)* [54] project in Japan to facilitate data analysis in the security research area. The D3M datasets (D3M 2010 and D3M 2011) contained malicious communication data when the client honeypot Marionette [18] accessed URLs in MDL [43]. Marionette inspected websites corresponding to the URLs in MDL to determine whether they were malicious or not at the time of inspecting [8]. From inspected URLs, 840 URLs are selected for the D3M datasets by the MWS project [54]. Both the D3M datasets and our malicious training dataset TRN_M use the same MDL [43]. Therefore, the 840 URLs are a subset of the 63,694 URLs in TRN_M. Table 2.11 lists three types of shellcode-related alerts and suspicious DNS query alert output from Snort, but the number of correct detection by Snort is small.

These two kinds of test results indicate that Snort IDS is practically

Table 2.12: Processing time with the test dataset D3M

Method	Training [sec]	Test [sec]	Test speed [addresses/sec]
Octet ($N = 1$)	15.6	0.57	1486
Octet ($N = 2$)	18.1	0.59	1415
Octet ($N = 3$)	31.5	0.67	1249
Octet ($N = 4$)	44.2	1.12	752
ExOctet ($N = 3$)	24.8	0.78	1074
Bit ($k = 8$)	18.1	0.70	1194
Bit ($k = 16$)	24.7	0.73	1146
Bit ($k = 24$)	45.1	1.20	698
Bit ($k = 32$)	65.8	2.04	412
Rblcheck	–	23.60	36
Snort	–	3.98	211

incapable of detecting malicious websites because there were too many false positive alerts. Comparison of Table 2.10 with Table 2.11 reveals that the same types of alerts are output for both benign and malicious data. In real environments, there is access to both benign and malicious websites. Therefore, the detection of malicious websites with Snort IDS is not readily performed, as demonstrated in this experiment.

For reference, the 840 malicious URLs in the D3M datasets are tested by our scheme. Our scheme detects 805 URLs correctly, so the detection accuracy is 0.958 (=805/840), and FNR is 0.042 (=35/840). This result shows that our scheme can compensate for the limitations of conventional IDS.

2.4.6 Experiment 4: Comparing the Processing Time

This experiment compares the processing time between our scheme and existing approaches. The processing time is measured under the same Linux machine with an Intel Xeon 2.40 GHz CPU and 4 GB RAM. Table 2.12 shows the results of this experiment.

In our scheme, the processing time is divided into training time and test

time. Training time is the time for building a trained model using the training dataset shown in Table 2.1. Test time is the total time for checking all IP addresses in the D3M 2010 and D3M 2011 datasets shown in Section 2.4.5. Test speed is the number of processable addresses per second, which is calculated from test time. Note that our scheme does not need to build a trained model every time and the same trained model can be used repeatedly. In the Octet method, the training time and the test time increase linearly with increasing N. In the ExOctet method, the training time is shorter than that of the Octet ($N = 3$) method, but the test time is longer than that of the Octet ($N = 3$) method. In the Bit method, the training time and the test time increase linearly with increasing k.

Existing approaches include network-side blacklists and signature-based IDS. As a network-side blacklist, *rblcheck* [55] is used for measuring the test time for checking IP addresses in the D3M datasets. Rblcheck is a tool for performing lookups in multiple DNSBL services. Using more DNSBL services takes a longer time. Therefore, only one of the DNSBL services is selected for comparing the processing time fairly. As shown in Table 2.12, the test time of rblcheck is up to 41 times longer than that of our scheme. The reason is that rblcheck contacts external DNSBL servers to check IP addresses using the DNS protocol. As a signature-based IDS, Snort (see Section 2.2.2) was selected. The test time of Snort with the D3M datasets is longer than any of our schemes. It is because Snort is checking not only IP addresses but also the entire packet payload.

For comparing the processing time, the test speed of our scheme is faster than that of existing approaches. Our scheme is based on only the IP address structure, so it is more lightweight than other approaches. Therefore, our scheme can be combined with existing approaches to compensate for their drawbacks.

2.4.7 Analysis of False Positives in Our Scheme

This section analyzes why our scheme incorrectly determines some of the actually benign IP addresses as malicious, namely false positives. The targets of the analysis are the IP addresses in TST_B, whose score is in the top 100. The analysis shows that 83 IP addresses were used for websites on hosting services, 2 IP addresses were used for CDN, and the other 18 IP addresses were not used at that time and could not be investigated. This result indicates that because it uses only IP addresses, our scheme is likely to falsely regard benign IP addresses used on hosting services as malicious. When at least one of the websites deployed on a hosting service is malicious, the other benign websites are falsely considered as malicious, namely as FPs, although they are benign.

2.5 Conclusion

In this study, a new scheme to detect malicious websites by learning the IP address features has been developed and evaluated. Our experimental results have indicated that features extracted only from IP addresses are distinct indicators that enables us to compensate for the limitation of existing approaches; i.e., our scheme can detect even *unknown* malicious websites with low errors. However, our scheme incorrectly considers some benign websites as malicious mainly because they are on the same web hosting services that are utilized by malicious websites. This warrants a more thorough investigation of our scheme for hosting services as part of our future work. Moreover, this chapter considers only IPv4 addresses structure because the number of malicious websites using IPv6 addresses is much less than that of using IPv4 addresses at the present time of writing. Thus, applying our IPv4 address-based approach to IPv6 address environment will be our future work in the near future. Nonetheless, the newly developed scheme will allow us to significantly enhance the detection performance when applied to existing network security systems.

Chapter 3

Profiling Time-series Variations of Domain Names

3.1 Introduction

Domain names are used by all Internet users and service providers for their online activities and businesses. Domain names and their protocol (domain name system (DNS)) are one of the most successful examples in distributed systems that can satisfy users' needs regarding easy use of the Internet. Attackers are also Internet users and they abuse easy-to-use domain names as a reliable cyberattack infrastructure. For example, in today's cyberattacks, domain names are used in serving malicious content or malware, controlling malware-infected hosts, and stealing personal or important information.

As countermeasures against domain name abuses, detecting and blacklisting known malicious domain names are basic strategies and widely applied to protect users from cyberattacks. Attackers understand these countermeasures and abuse DNS to mystify their attack ecosystems; DNS fast-flux and domain generation algorithms (DGAs) are used to evade blacklisting. The key feature of these techniques is that they systematically generate a huge volume of distinct domain names. These techniques have made it infeasible for blacklisting approaches to keep up with newly generated malicious domain names.

Ideally, to fully address the underlying issue with domain name black-

lists, we need to observe and track every newly registered and updated domain name in real time and judge whether the domain name is involved in any attackers' infrastructure. However, in reality, it is virtually impossible to obtain a solution because of the following three reasons. One is that attackers use techniques, such as DNS fast-flux and DGAs, to systematically generate a huge volume of distinct domain names. The second is that the number of existing domain names is too large to track in real time. The number of second-level domain (2LD) names (e.g., `example.com`) is now over 296 million [56]. Multiple fully qualified domain names (FQDNs) (e.g., `www.example.com`) may exist under the same 2LD names; therefore, the number of all existing FQDNs could be in the billions. The third reason is that no one can fully understand all and real-time changes in the mappings between domain names and IP addresses. Since DNS is a distributed system and the mappings are configured in each authoritative name server, real-time observation of the mappings of all domain names is infeasible.

Given these reasons, blacklisting approaches based on DNS observations have failed to keep up with newly generated malicious domain names. Thus, we adopt an approach of *prediction* instead of observation, i.e., we aim to discover malicious domain names that are likely to be abused in future. The key idea of this approach is to exploit *temporal variation patterns (TVPs)* of malicious domain names. The TVPs of domain names include the information about how and when a domain name has been listed in legitimate/popular and/or malicious domain name lists. We use TVPs to apprehend the variations in domain names, e.g., a domain name is newly registered or updated, IP addresses corresponding to the domain name are changed, and the traffic directed to the domain name is changed.

On the basis of the aforementioned idea, we developed a system that actively collects DNS logs, analyzes their TVPs, and predicts whether a given domain name will be used for a malicious purpose. Our main contributions are summarized as follows.

- We propose a system called DomainProfiler that identifies TVPs

of domain names to precisely profile various types of malicious domain names.

- Our evaluation with real and large ground truth data reveals that we can predict malicious domain names 220 days beforehand with a true positive rate (TPR) of 0.985 in the best-case scenario.

The rest of this chapter is organized as follows. We give the motivation of our key idea in Section 3.2. In Section 3.3, we discuss our proposed system DomainProfiler. We describe the datasets we used and the results of our evaluation in Section 3.4. We discuss the limitations of our system in Section 3.5 and related work in Section 3.6. Finally, we conclude this chapter in Section 3.7.

3.2 Motivation: Temporal Variation Pattern

We define a temporal variation pattern (TVP) as the time-series behavior of each domain name in various types of domain name lists. Specifically, we identify how and when a domain name has been listed in legitimate/popular and/or malicious domain name lists. Our motivation for considering TVPs is based on the observation that both legitimate and malicious domain names vary dramatically in domain name lists over time. The following are three reasons for using different and multiple domain name lists. One is that the data are realistically observable; that is, we can easily access the data from domain name list maintainers. The second is that domain name lists are created based on objective facts confirmed by the maintainer of those lists. The third is that multiple domain name lists and the time-series changes of those lists can boost the reliability of listed domain names.

As shown in Fig. 3.1, our proposed system defines and identifies four TVPs (*null, stable, fall,* and *rise*) for each domain name in a domain name list. *Null* means the domain name has not been listed in the specified time window. *Stable* means the domain name has been continuously listed in the time window. *Fall* is a state in which the domain name was first listed then

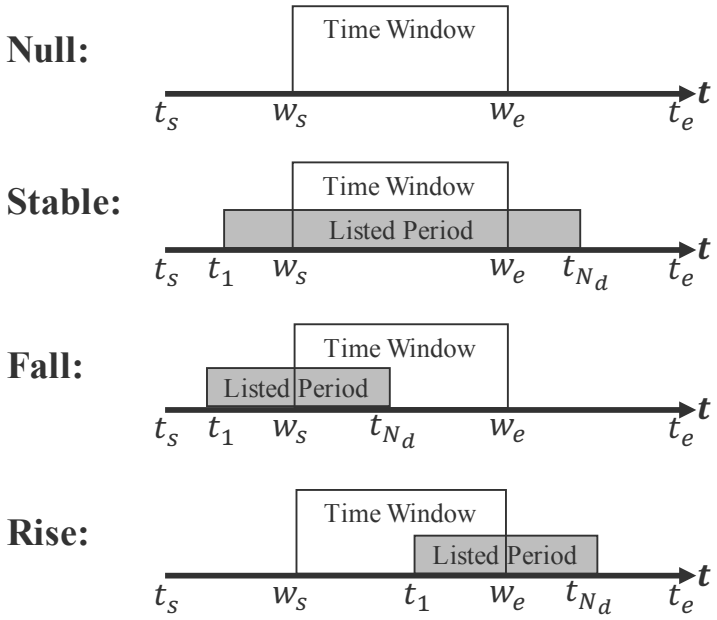


Figure 3.1: Simplified Temporal Variation Patterns (TVPs)

delisted during the time window. *Rise* means that the domain name was first unlisted then listed during the time window.

Definition: A set $T_d = \{t_1, ..., t_{N_d}\}$ is an ordered N_d of timestamps when a domain name d has been listed/contained in a domain name list. The domain name list is collected from t_s to t_e. Given a set of timestamps T_d and a time window between a starting point w_s and ending point w_e, the TVP of a domain name is defined as follows.

$$TVP = \begin{cases} Null & (\min(T_d \cup \{t_e\}) > w_e \vee \max(T_d \cup \{t_s\}) < w_s) \\ Stable & (\min(T_d \cup \{t_e\}) < w_s \wedge \max(T_d \cup \{t_s\}) > w_e) \\ Fall & (\min(T_d \cup \{t_e\}) < w_s \wedge w_s < \max(T_d \cup \{t_s\}) < w_e) \\ Rise & (w_s < \min(T_d \cup \{t_e\}) < w_e \wedge \max(T_d \cup \{t_s\}) > w_e) \end{cases}$$

These TVPs are common and generic features that can contribute to accurately discriminating malicious domain names controlled by attackers from legitimate domain names. Thus, the focus of these patterns covers a wide range of malicious domain names used in a series of cyberattacks such

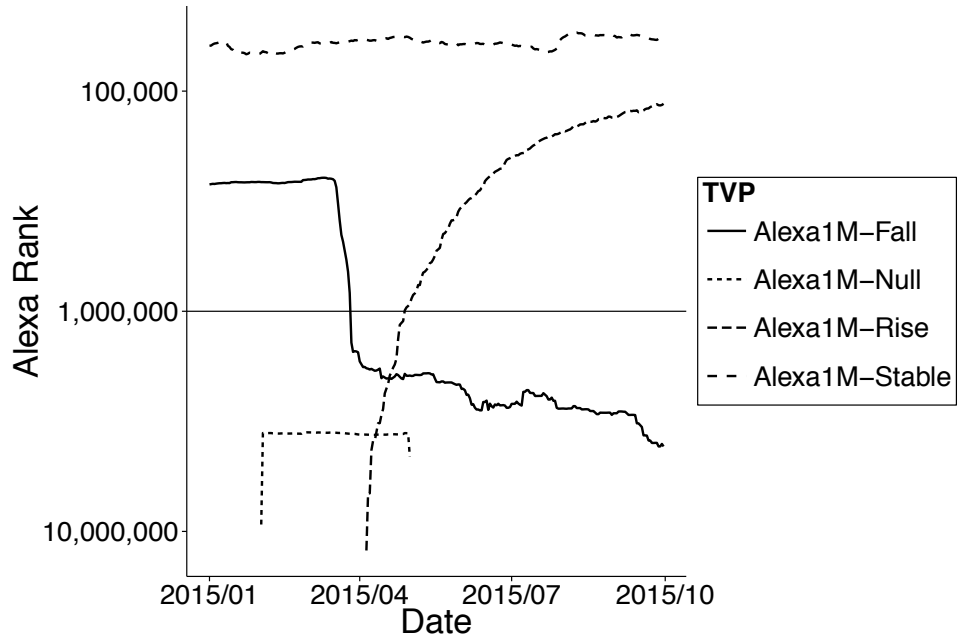


Figure 3.2: Example of TVPs in Legitimate/Popular Domain Name List (Alexa Top Sites)

as drive-by download attacks, malware downloads, command and control (C&C), and phishing attacks.

In this chapter, we use the domain names ranked in the Alexa Top Sites [42] as the legitimate/popular domain name list. Alexa provides the top one million popular sites based on the global one-month average traffic rank. We divide the Alexa list on the basis of the rank to create four domain name lists, Alexa top 1,000 (*Alexa1k*), Alexa top 10,000 (*Alexa10k*), Alexa top 100,000 (*Alexa100k*), and Alexa top 1,000,000 (*Alexa1M*). The TVPs for the Alexa Top Sites are identified based on these four lists. Figure 3.2 shows examples of typical domain names that fit the four patterns in Alexa1M. The graph indicates the relationships between domain names and their Alexa rank variations over time (note the logarithmic y-axis). In the null pattern of Alexa1M (*Alexa1M-Null*), the rank of a domain name has always been outside 1M and has never been listed in Alexa1M. The Alexa1M-Null pattern is intended to be one of the features or hints to boost

true positive rates (TPRs), which is the ratio of correctly predicted malicious domain names to actual malicious domain names. This is because the rank for legitimate domain names is more likely to be within 1M, and new domain names by attackers cannot be in Alexa1M right after they have registered. In the stable pattern of Alexa1M (*Alexa1M-Stable*), the rank of a domain name has always been within 1M and listed in Alexa1M. Alexa1M-Stable includes stable popular domain names; thus, this pattern can be used for improving true negative rates (TNRs), which is the ratio of correctly predicted legitimate domain names to actual legitimate domain names. In the fall pattern of Alexa1M (*Alexa1M-Fall*), the rank of a domain name was first within 1M, fell, and finally delisted from Alexa1M. The Alexa1M-Fall pattern is intended to detect maliciously re-registered, parked, and hijacked domain names that changed from originally legitimate domain names to improve TPRs. In the rise pattern of Alexa1M (*Alexa1M-Rise*), the rank of a domain name was first outside 1M then increased to be within 1M. This Alexa1M-Rise pattern includes legitimate start-up websites' domain names during the specified time window to improve TNRs.

We use the domain names listed in the public blacklist hpHosts [57] as the malicious domain name list. The hpHosts provides malicious domain names of malicious websites engaged in exploits, malware distribution, and phishing. The TVPs for hpHosts are defined in a similar way for Alexa. Note that hpHosts does not have any continuous value, such as ranking, and only has information of whether domain names are listed. Figure 3.3 shows examples of typical domain names that fit the four patterns in hpHosts. In the null pattern of hpHosts (*hpHosts-Null*), a domain name has never been listed in hpHosts. This hpHosts-Null pattern can be used for improving TNRs because legitimate domain names are less likely to be listed in hpHosts. In the stable pattern of hpHosts (*hpHosts-Stable*), a domain name has always been listed in hpHosts. The hpHosts-Stable pattern includes domain names related to bullet-proof hosting providers, which provide network resources even to attackers, to improve TPRs. In the fall pattern of hpHosts

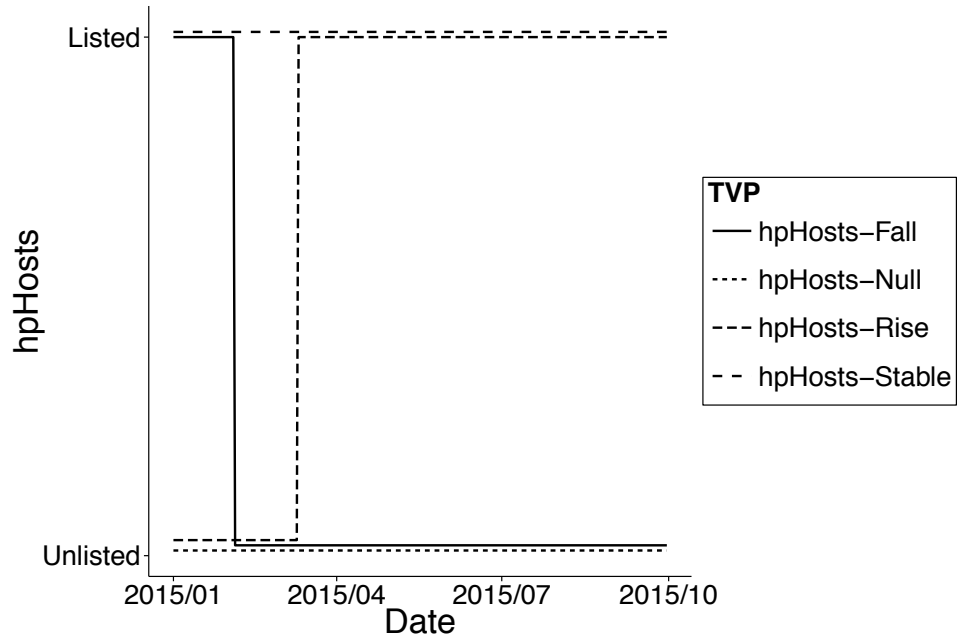


Figure 3.3: Example of TVPs in Malicious Domain Name List (hpHosts)

Table 3.1: Relationships between TVPs and Objectives

Objectives	**TVPs**
Improving True Positive Rates (TPRs)	Alexa-Null, Alexa-Fall, hpHosts-Stable, hpHosts-Rise
Improving True Negative Rates (TNRs)	Alexa-Stable, Alexa-Rise, hpHosts-Null, hpHosts-Fall

(*hpHosts-Fall*), a domain name was once listed then unlisted. For example, this pattern includes domain names that were once abused then sanitized to improve TNRs. In the rise pattern of hpHosts (*hpHosts-Rise*), a domain name was listed from the middle of the specified time window. This hpHosts-Rise pattern is intended to detect newly registered malicious domain names that attackers will use for a while. Specifically, many subdomain names can be created under the same domain name to bypass fully qualified domain names (FQDN)-level blacklists. Thus, the hpHosts-Rise pattern contributes to understanding the situation to increase TPRs.

As described above, these TVPs in both legitimate/popular and malicious

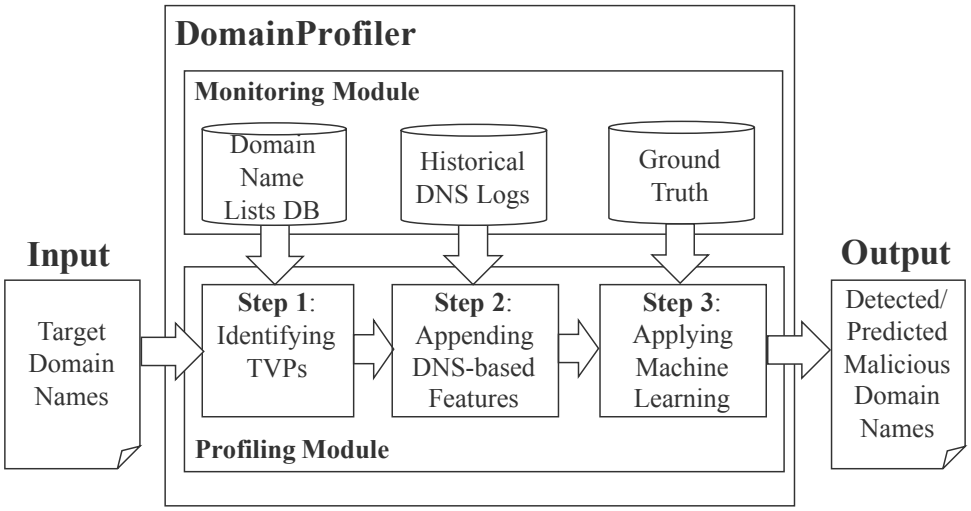


Figure 3.4: Overview of Our System

domain name lists contribute to boosting both TPR and TNR. Table 3.1 summarizes the relationships between the TVPs and their objectives. The effectiveness of using these patterns in real datasets is described later in Section 3.4.

3.3 Our System: DomainProfiler

DomainProfiler identifies the temporal variation patterns (TVPs) of domain names and detects/predicts malicious domain names. Figure 3.4 gives an overview of our system architecture. DomainProfiler is composed of two major modules: monitoring and profiling. The monitoring module collects various types of essential data to evaluate the maliciousness of unknown domain names. The profiling module detects/predicts malicious domain names from inputted target domain names by using the data collected with the monitoring module. The details of each module are explained step by step in the following sections.

3.3.1 Monitoring Module

The monitoring module collects three types of information that will be used later in the profiling module. The first type of information is domain name lists. As discussed in Section 3.2, we need to collect the legitimate/popular domain name list (Alexa) and malicious domain name list (hpHosts) on a daily basis to create a database of listed domain names and their time-series variations.

The second type of information is historical DNS logs, which means time-series collections of the mappings between domain names and IP addresses. A passive DNS [58] is one typical way to collect such mappings by storing resolved DNS answers at large caching name servers. Due to the privacy policy of our organization, we do not use the passive DNS approach. Instead, we actively send DNS queries to domain names to monitor and build a passive DNS-like database. On the plus side, this active monitoring contains no personally identifiable information of the senders. Moreover, we can control DNS queries so as not to contain disposable domain names [59], which are non-informative and negatively affect the database. For example, disposable domain names are automatically generated one-time domain names to obtain a user's environmental information by using certain anti-virus products and web services. Since these domain names are distinct, their mappings between domain names and IP addresses significantly increase the database size with non-informative information for evaluating the maliciousness of domain names. On the minus side of active monitoring, we can only query known domain names and cannot gather the mappings of unknown domain names. Thus, we have expanded known existing domain names as much as possible to partially address this issue. For example, we have extracted all domain names in domain name lists such as Alexa and hpHosts. Moreover, we crawl approximately 200,000 web pages every day to gather web content and extract domain names. Furthermore, we query a search engine API (2.5M queries/month) to expand the domain names based on the above results.

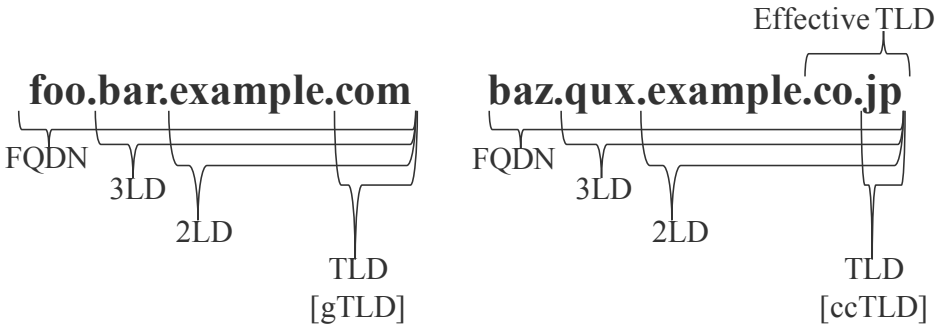


Figure 3.5: Definition of Domain Name Terms

The third type of information is the ground truth, which will be used to label the training dataset and evaluate the effectiveness of our system. Our ground truth includes the results of web client-based honeypots (honeyclients) and sandbox systems and some subscription-based data such as VirusTotal [60] and professional services by a security vendor. The details of the ground truth we used are given later in Section 3.4.1.

3.3.2 Profiling Module

The profiling module consists of three steps. Each step uses the information collected from the monitoring module to finally output malicious domain names from inputted target domain names.

Step 1: Identifying TVPs

Step 1 identifies the TVPs for each input target domain name. The definition was already introduced in Section 3.2. First, we query the input domain name to the domain name lists database to obtain the time-series data of listed domain names that match the second-level domain (2LD) part of the input domain name. The database consists of five domain name lists: Alexa1k, Alexa10k, Alexa100k, Alexa1M, and hpHosts.

To precisely define the TVP of every domain name, we define that the top-level domain (TLD) includes an effective TLD or public suffix [61] such

as `.com.au`, `.co.jp`, and `.co.uk`, as shown in Fig. 3.5. In general, TLDs are divided into generic top-level domains (gTLDs), such as `.com`, `.net`, and `.org`, and country code top-level domains (ccTLDs) such as `.au`, `.jp`, and `.uk`. If we do not use effective TLDs, there is a significant difference in the 2LD parts between gTLDs and ccTLDs. For example, in the gTLD case of `foo.bar.example.com`, the 2LD part is `example.com`; however, in the ccTLD case of `baz.qux.example.co.jp`, the 2LD part is `co.jp`. Our definition of including effective TLDs is intended to treat both gTLD and ccTLD identically, that is, the 2LD part in the above ccTLD example is `example.co.jp` in this chapter.

Second, the TVPs of the matched 2LD parts within a specified time window are identified using the predefined patterns (null, stable, fall, and rise), as shown in Section 3.2.

Third, the numbers of matched 2LD parts for the four patterns are counted and used as feature vectors in a machine learning algorithm. Specifically, the feature vectors created in step 1 correspond to Nos. 1–20 of the features listed in Table 3.2, that is, Nos. 1–4 are for Alexa1k, Nos. 4–8 are for Alexa10k, Nos. 9–12 are for Alexa100k, Nos. 13–16 are for Alexa1M, and Nos. 17–20 are for hpHosts.

Step 2: Appending DNS-based Features

Step 2 appends DNS-based features to the output of step 1, which are input target domain names with identified TVPs. This step is intended to detect malicious domain names that share common features in terms of IP addresses and domain names. We reviewed and analyzed the typical features proposed for known approaches to select the DNS-based features. The known approaches that are related to ours are summarized later in Section 3.6. As a result of verifying the availability and effectiveness of the features, we decided to follow the features proposed for Notos [4]. The DNS-based features are mainly divided into two types: related IP addresses (rIPs) and related domain names (rDomains).

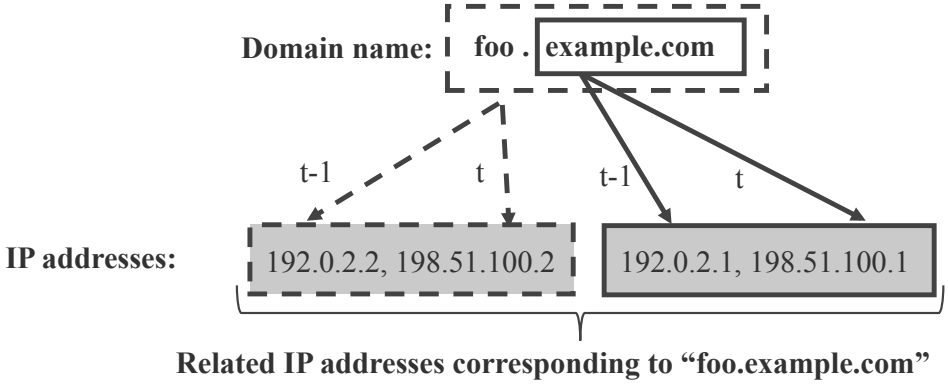


Figure 3.6: Graph for Related IP Addresses (rIPs)

To acquire features of rIPs, we need to first construct a graph of rIPs for each target domain name. Figure 3.6 shows an example of rIPs for `foo.example.com`. The graph is a union of every resolved IP address corresponding to each domain name at the FQDN level and its parent domain name levels, such as 3LD and 2LD, from historical DNS logs collected in the former monitoring module. In Fig. 3.6, the FQDN and 3LD (`foo.example.com`) correspond to the IP address `192.0.2.2` at time $t-1$ and `198.51.100.2` at t, and the 2LD (`example.com`) corresponds to the IP address `192.0.2.1` at $t-1$ and `198.51.100.1` at t. Thus, these four IP addresses are defined as rIPs for `foo.example.com`. Then, we extract the features from rIPs. These features consist of three subsets: border gateway protocol (BGP), autonomous system number (ASN), and registration.

The BGP features, Nos. 21–29 in Table 3.2, are created from the information of BGP prefixes corresponding to the related IP addresses (rIPs) of each target domain name. To obtain the required BGP information, we refer to the CAIDA dataset [62]. Specifically, we extract the number of rIPs' BGP prefixes of the target FQDN (No. 21), that of the 3LD part of the target (No. 22), and that of the 2LD part of the target (No. 23); the number of countries for the BGP prefixes of the target FQDN (No. 24), that of the 3LD part of the target (No. 25), and that of the 2LD part of the target (No. 26); the number of rIPs for the 3LD part of the target (No. 27) and that for the

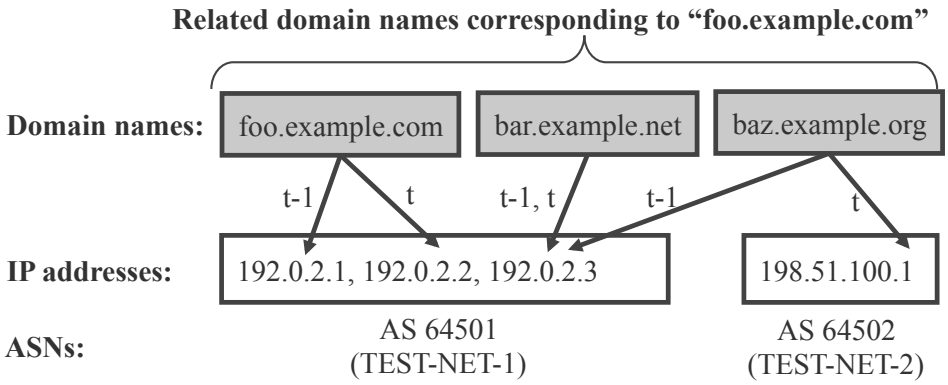


Figure 3.7: Graph for Related Domain Names (rDomains)

2LD part of the target (No. 28); and the number of organizations for the BGP prefixes of the target FQDN (No. 29).

The ASN features, Nos. 30–32 in Table 3.2, are created from the ASN information corresponding to the rIPs of each target domain name. To obtain the ASN information, we refer to the MaxMind GeoIP2 databases [63]. Specifically, we extract rIPs' ASNs of the target FQDN (No. 30), that of the 3LD part of the target (No. 31), and that of the 2LD part of the target (No. 32).

The registration features, Nos. 33–38 in Table 3.2, are created from the IP address registration information corresponding to the rIPs of each target domain name. To obtain the registration information, we refer to the information of delegated IP addresses [64] from all regional Internet registries (RIRs), namely AFRINIC, APNIC, ARIN, LACNIC, and RIPE NCC. Specifically, we extract the number of RIRs of the rIPs for the target FQDN (No. 33), that of the 3LD part of the target (No. 34), and that of the 2LD part of the target (No. 35); as well as the diversity or number of allocated dates of the rIPs for the target FQDN (No. 36), that of the 3LD part of the target (No. 37), and that of the 2LD part of the target (No. 38).

On the other hand, to acquire the features of related domain names (rDomains), we need to construct a graph of rDomains for each target domain

name using the historical DNS logs collected in the monitoring module. Figure 3.7 shows an example of rDomains for `foo.example.com`. The graph is a union of domain names pointing to IP addresses in the same autonomous system number (ASN) of the historical IP addresses of each target domain name. In Fig. 3.7, the ASN for the target `foo.example.com` is AS64501 and another IP address `192.0.2.3` in AS64501 is connected to the domain names `bar.example.net` and `baz.example.org`. Thus, these three domain names are defined as rDomains for `foo.example.com`, and we extract their features. These features consist of three subsets: FQDN string, n-grams, and top-level domain (TLD).

The FQDN string features, Nos. 39–41 in Table 3.2, are created from the set of rDomains for each target domain name. Specifically, we extract the number of FQDNs (No. 39) in the rDomains, mean length of the FQDNs (No. 40), and standard deviation (SD) of the length of the FQDNs (No. 41).

The n-gram features, Nos. 42–50 in Table 3.2, are created from the occurrence frequency of n-grams ($n = 1, 2, 3$) in the set of rDomains for each target domain name. Note that the units of n-grams in this chapter are denoted with characters; thus, 2-grams for `example.com` consists of pairs of characters such as `ex`, `xa`, and `am`. Specifically, we extract the mean, median, and SD of 1-gram (Nos. 42–44) in rDomains, those of 2-grams (Nos. 45–47), and those of 3-grams (Nos. 48–50).

The TLD features, Nos. 51–55 in Table 3.2, are created from TLDs in the set of rDomains for each target domain name. Specifically, we extract the distinct number of TLDs in the set of rDomains (No. 51), ratio of the .com TLD in the set (No. 52), and mean, median, and SD of the occurrence frequency of the TLDs in the set (Nos. 53–55).

Table 3.2: List of Features

Type	No.	Feature Name	Type	No.	Feature Name	Type	No.	Feature Name
TVP (Legitimate /Popular)	1	Alexa1k-Null	rIP (BGP)	21	# BGP Prefixes (FQDN)	rDomain (FQDN)	39	# FQDNs
	2	Alexa1k-Stable		22	# BGP Prefixes (3LD)		40	Mean Lengths
	3	Alexa1k-Fall		23	# BGP Prefixes (2LD)		41	SD Lengths
	4	Alexa1k-Rise		24	# Countries (FQDN)	rDomain (1-gram)	42	Mean Distribution
	5	Alexa10k-Null		25	# Countries (3LD)		43	Median Distribution
	6	Alexa10k-Stable		26	# Countries (2LD)		44	SD Distribution
	7	Alexa10k-Fall		27	# IP addresses (3LD)	rDomain (2-grams)	45	Mean Distribution
	8	Alexa10k-Rise		28	# IP addresses (2LD)		46	Median Distribution
	9	Alexa100k-Null		29	# Organizations (FQDN)		47	SD Distribution
	10	Alexa100k-Stable	rIP (ASN)	30	# ASNs (FQDN)	rDomain (3-grams)	48	Mean Distribution
	11	Alexa100k-Fall		31	# ASNs (3LD)		49	Median Distribution
	12	Alexa100k-Rise		32	# ASNs (2LD)		50	SD Distribution
	13	Alexa1M-Null	rIP (Registration)	33	# Registries (FQDN)	rDomain (TLD)	51	# TLDs
	14	Alexa1M-Stable		34	# Registries (3LD)		52	# Ratios of .com
	15	Alexa1M-Fall		35	# Registries (2LD)		53	Mean Distribution
	16	Alexa1M-Rise		36	# Dates (FQDN)		54	Median Distribution
TVP (Malicious)	17	hpHosts-Null		37	# Dates (3LD)		55	SD Distribution
	18	hpHosts-Stable		38	# Dates (2LD)			
	19	hpHosts-Fall						
	20	hpHosts-Rise						

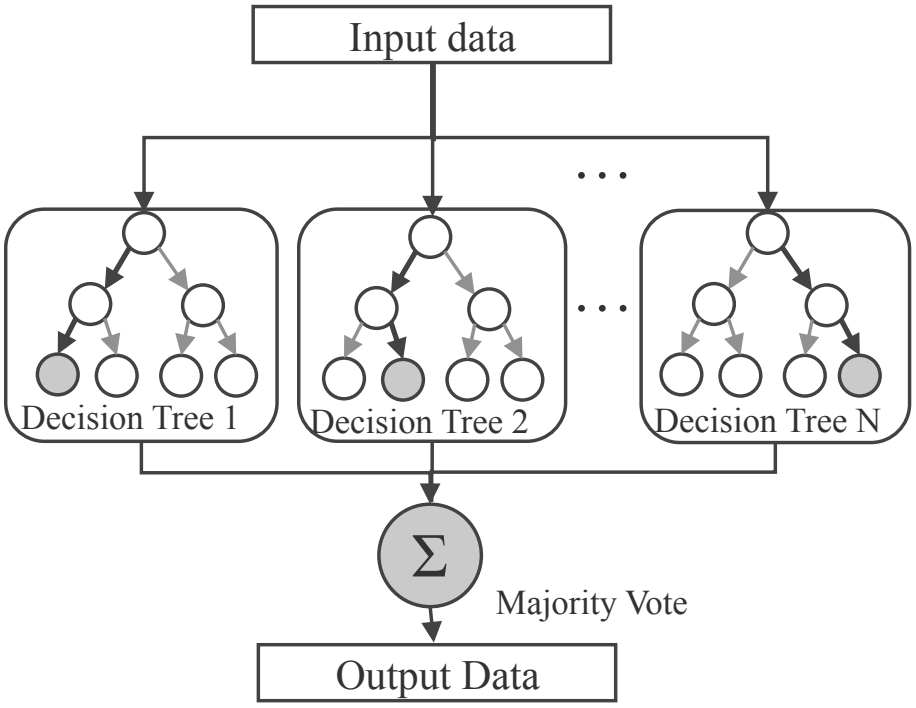


Figure 3.8: Concept Image of Random Forest

Step 3: Applying Machine Learning

Step 3 involves applying a machine learning algorithm to the outputs of step 2, which consist of input target domain names with all the features listed in Table 3.2. This step is designed to achieve our goal of detecting/predicting possible malicious domain names in future. To this end, we use supervised machine learning to effectively find possible malicious domain names from unvalued input domain names. Supervised machine learning basically consists of two phases: training and test. The training phase generates a learning model based on the labeled malicious and legitimate training data with extracted features. The test phase calculates the maliciousness of each input domain name with extracted features using the learning model generated in the training phase to detect/predict malicious domain names.

Among many supervised machine learning algorithms, we selected a *random forest* [65] because of its high accuracy, as identified in our preliminary

Table 3.3: Dataset

Type	**Dataset**	**Period**	**# FQDNs**
Target Domain Names (training set)	Legitimate-Alexa	2013-05-22–2015-02-28	89,739
	Malicious-hpHosts	2013-01-17–2015-02-28	83,670
Target Domain Names (test set)	Honeyclient-Exploit	2015-03-01–2015-10-07	537
	Honeyclient-Malware	2015-03-01–2015-10-07	68
	Sandbox-Malware	2015-03-01–2015-10-07	775
	Sandbox-C&C	2015-03-01–2015-10-07	8,473
	Pro-C&C	2015-03-01–2015-03-29	97
	Pro-Phishing	2015-03-01–2015-03-29	78,221
Domain Name Lists DB	AlexaDB	2013-05-22–2015-02-28	5,596,219
	hpHostsDB	2013-01-17–2015-02-28	1,709,836
Historical DNS Logs	DNSDB	2014-10-01–2015-02-28	47,538,966

experiments, and high scalability, which we can easily parallelize. The concept image of a random forest is shown in Fig. 3.8. A random forest consists of many *decision trees*, which are constructed from input data with randomly sampled features. The final prediction is output by the majority vote of the decision trees.

3.4 Evaluation

DomainProfiler was evaluated using real datasets including an extensive number of domain names. This section explains how we evaluated it in terms of the effectiveness of our approach of using temporal variation patterns (TVPs) and the detection/prediction performance of malicious domain names used in real cyberattacks.

3.4.1 Dataset

Our evaluations required three types of datasets, as shown in Table 3.3: target domain names, domain name list databases, and historical DNS logs.

The first dataset was target domain names, which were composed of training and test sets. The training set was labeled data for creating a learning model in a random forest. To create the *Legitimate-Alexa*, we extracted

fully qualified domain names (FQDNs) based on the domain names listed in Alexa100k. Since most of the domain names in Alexa are second-level domain (2LD) and do not have IP addresses, we used a search engine API to randomly extract existing FQDNs in the wild from each 2LD name. Moreover, as shown in Section 3.3.1, we used our ground truth, such as the results of honeyclients and subscription-based professional data, to eliminate the possibility that malicious domain names are in Legitimate-Alexa. As for *Malicious-hpHosts*, we used a similar process to Legitimate-Alexa; that is, we extracted FQDNs from 2LD names listed in hpHosts using a search engine and verified the maliciousness using our ground truth.

The test set was used for evaluating the predictive detection performance of our system. Note that there were no overlaps between the training and test sets and the collected period of the test set was after that of the training set. Thus, we are able to simulate the predictive performance of abused domain names in future by using the test set. For web client-based honeypot (honeyclient) datasets, we used our honeyclients to collect newly malicious FQDNs, particularly related to drive-by download attacks, from March 2015 to October 2015. *Honeyclient-Exploit* contained FQDNs of websites engaged in distributing exploits that target users' browsers and their plugins. *Honeyclient-Malware* was the collection of FQDNs used for malware distribution websites in drive-by download attacks. To create sandbox datasets, we used our sandbox systems to run 13,992 malware samples randomly downloaded from VirusTotal [60]. The *Sandbox-Malware* dataset contained FQDNs connected by malware samples (e.g., downloader) to download other malware samples. The *Sandbox-C&C* dataset was a collection of FQDNs of command and control (C&C) servers detected with our sandbox. The *Pro-C&C* and *Pro-Phishing* datasets were FQDNs used for C&C servers and phishing websites, respectively. Note that the Pro datasets were obtained from commercial and professional services provided by a security vendor, and the FQDNs we selected were only those with high confidence of being abused by attackers.

The second dataset was a domain name list database used for identifying

TVPs. As explained in Section 3.2, we selected Alexa top sites as a legitimate/popular list, and hpHosts as a malicious list since they are continuously obtainable daily.

The third dataset was historical DNS logs, which involved time-series collections of the domain name and IP address mappings. As discussed in Section 3.3.1, we actively sent DNS queries to the domain names we found by using domain name lists, our web crawler, and search engine API. That is, we extracted all domain names in domain name lists such as Alexa and hpHosts. Moreover, we crawled approximately 200,000 web pages every day to gather web content and extracted domain names from the content and their static and dynamic hyperlinks. Furthermore, we expanded the number of domain names using an external search engine API (2.5M queries/month) based on the above domain names. In our evaluations, we used over 47M distinct FQDNs and their time-series changes from October 2014 to February 2015, as shown in Table 3.3.

3.4.2 Parameter Tuning

Before we evaluated our DomainProfiler, we needed to tune two types of parameters: the size of the time window in TVPs (step 1) and the required parameters to run the random forest (step 3).

Here is the summary of evaluation criteria discussed in the following sections. A true positive (TP) is the number of correctly predicted malicious domain names, a false positive (FP) is the number of incorrectly predicted legitimate ones, a false negative (FN) is the number of incorrectly predicted malicious ones, and a true negative (TN) is the number of correctly predicted legitimate ones. The true positive rate (TPR), otherwise known as recall, is the ratio of correctly detected malicious domain names to actual malicious domain names. The precision is the ratio of actual malicious domain names to domain names detected as malicious with the system. The true negative rate (TNR) is the ratio of correctly determined legitimate domain names among actual legitimate domain names. The F-measure is the ratio that

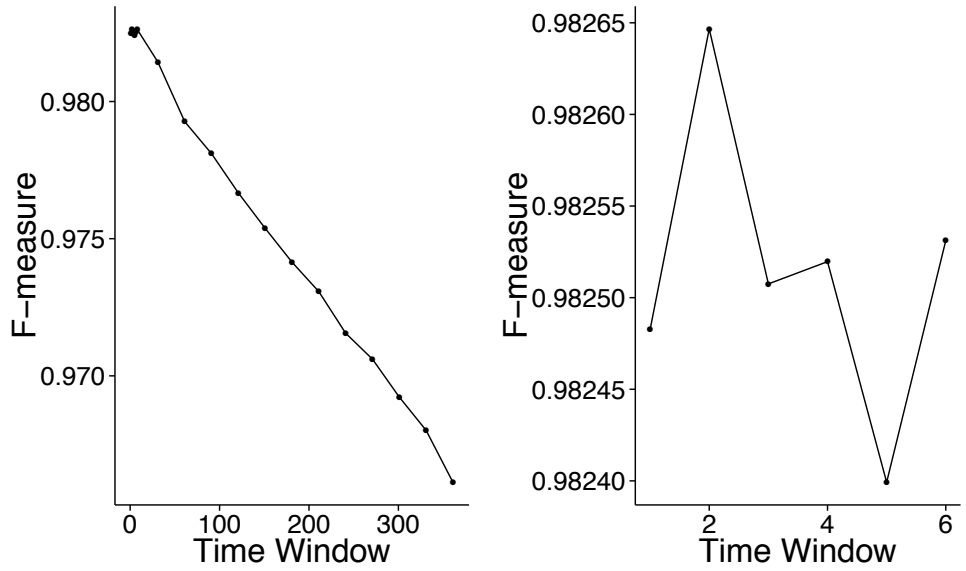


Figure 3.9: Tuning Time Window Size

combines recall and precision, namely, it is calculated as the harmonic mean of precision and recall.

Time Window Size

We conducted 10-fold cross-validations (CVs) using the training set with variable time window sizes (from 1 to 365 days) to select the time window size based on the evaluation criteria. Figure 3.9 shows two graphs: the left one corresponds to the time window sizes from 1 to 365 days and the right one corresponds to those from 1 to 7 days. These two graphs reveal that the best time window size for TVPs is only two days. This is not a surprising result for us in terms of the nature of domain names or TVPs. Since attackers abuse the DNS to generate a huge volume of distinct domain names from one day to the next, keeping old information over a long period decreases the F-measure of our system.

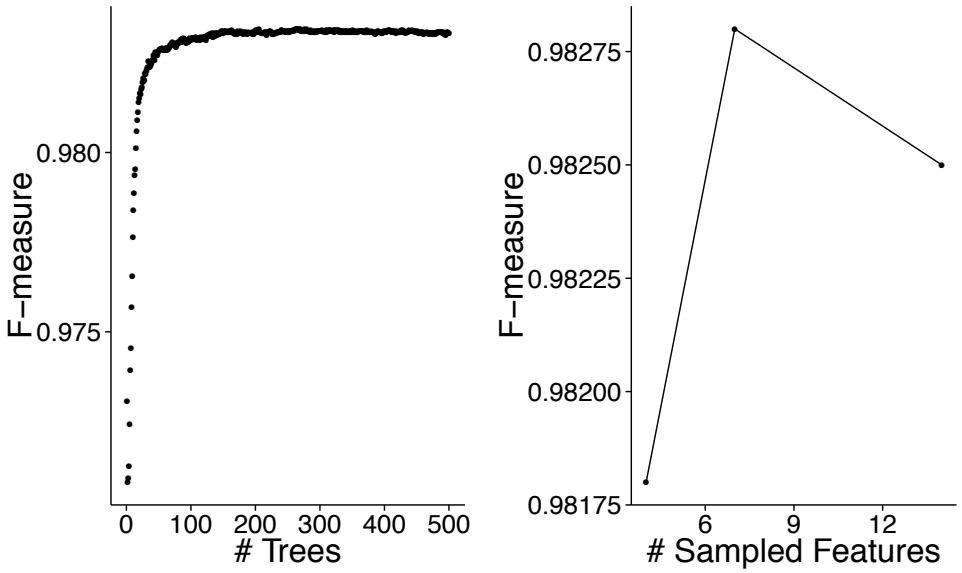


Figure 3.10: Tuning Random Forest Parameters

Random Forest

A random forest [65] requires two parameters to run. One parameter is the number of decision trees. As explained in Section 3.3.2, a random forest consists of multiple decision trees; thus, we needed to decide how many trees to make beforehand. As is the case with the aforementioned time window size, we conducted 10-fold CVs by changing the number of trees to determine the optimum number of decision trees. The left graph in Fig. 3.10 shows the relationships between the number of trees and F-measure. The graph shows that F-measures are stable over 100 trees. Thus, we decided to use 100 decision trees in the following evaluations.

The other parameter is the number of sampled features in each individual decision tree. A random forest constructs decision trees from input data with randomly sampled features to improve overall accuracy. We conducted 10-fold CVs again to search for the optimum number of sampled features. The right graph in Fig. 3.10 shows that the best F-measure is obtained when the number of sampled features is 7.

3.4.3 Feature Set Selection

Now that we selected the optimal parameters (the time window size, number of trees, and number of sampled features in each tree), this section compares the detection performance with different feature sets. The feature sets include the temporal variation pattern (TVP), related IP address (rIP), related domain name (rDomain), combination of rIP and rDomain (rIP+rDomain), and combination of TVP, rIP, and rDomain (TVP+rIP+rDomain). We conducted 10-fold CVs using the training set and the optimal parameters by changing the feature sets to estimate how accurately each feature set will perform in theory. Table 3.4 illustrates the detection performance using the above evaluation criteria. Note that the number of FQDNs varies with feature sets due to the availability of each feature. For example, some domain names have no rIPs and/or rDomains. Also, Fig. 3.11 shows the receiver operator characteristic (ROC) curves. An ROC curve shows a pair of FPR and TPR corresponding to a particular decision cut-off point. Thus, if the ROC curve of a feature set rises more rapidly, it means that the performance of the feature set is better. An area under curve (AUC) is the area under the ROC curve and is generally used to evaluate a binary classifier. We calculated the AUC for each ROC curve, and the results are also listed in Table 3.4. Table 3.4 and Fig. 3.11 illustrate that using our TVP features significantly contributes to achieving better detection performance than using only DNS-based features. Using only DNS-based features (rIP, rDomain, and rIP+rDomain) does not go beyond 0.90 in any evaluation criteria. These results show that only using common/typical DNS-based features proposed for known approaches proves to be insufficient for detecting malicious domain names in current attack ecosystems. However, combining the DNS-based features with our TVP features (TVP+rIP+rDomain) achieves the best result, namely, a TPR/recall of 0.975, TNR of 0.991, precision of 0.990, and F-measure of 0.983. These results indicate that our key idea based on using TVPs is effective for improving both TPR and TNR exactly as intended.

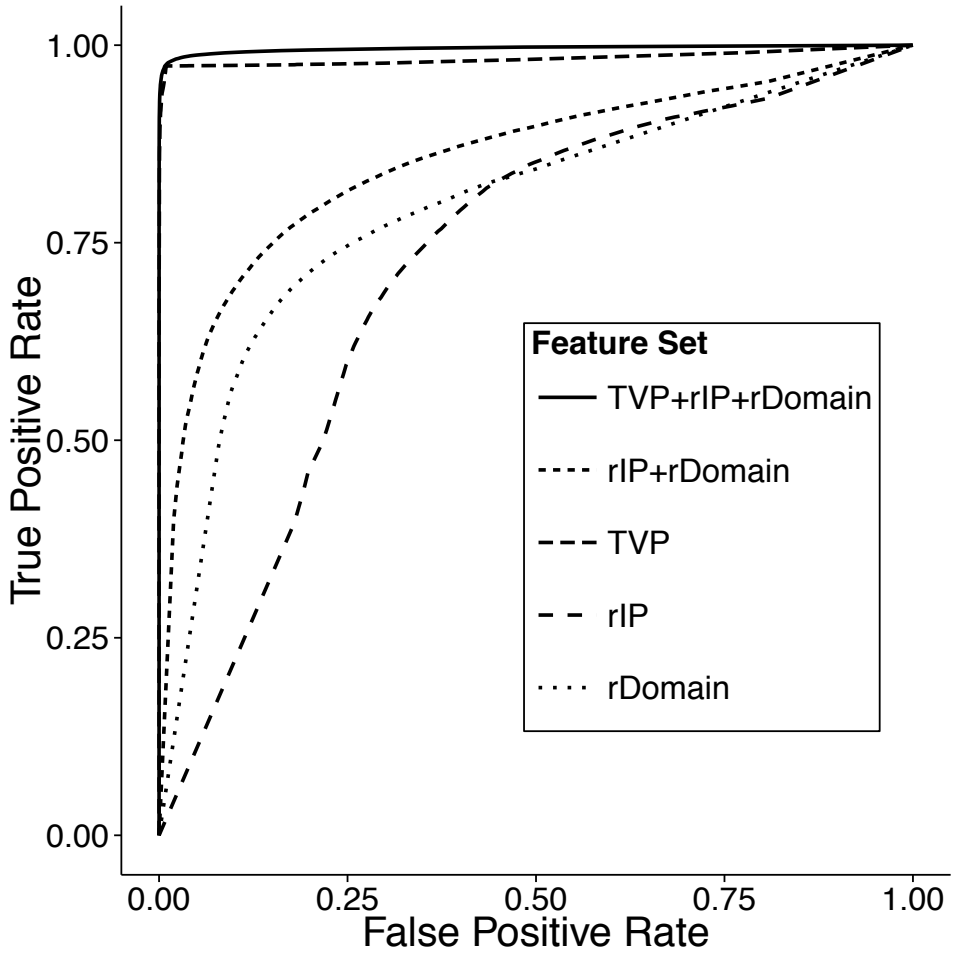


Figure 3.11: ROC curves

Table 3.4: Detection Performance with Different Feature Sets (Cross-Validation)

Feature Set	**TP**	**FP**	**FN**	**TN**	**# FQDNs**	**TPR/Recall**	**TNR**	**Precision**	**F-measure**	**AUC**
TVP	81,436	834	2,234	88,905	173,409	0.973	0.991	0.990	0.982	0.983
rIP	62,734	32,347	20,270	57,306	172,657	0.756	0.639	0.660	0.705	0.722
rDomain	58,095	16,523	24,873	72,893	172,384	0.700	0.815	0.779	0.737	0.794
rIP+rDomain	61,928	13,197	21,033	76,206	172,364	0.746	0.852	0.824	0.783	0.855
TVP+rIP+rDomain	80,879	798	2,082	88,605	172,364	0.975	0.991	0.990	0.983	0.996

3.4.4 System Performance

We evaluated the system performance of a prototype version of DomainProfiler. Specifically, we calculated the execution time and data size in each step when we conducted a 10-fold CV using the training set with the optimal parameters and best feature set (TVP+rIP+rDomain). Step 1 (identifying TVPs) was executed on a single server with a 10-core 2.2-GHz CPU and 128-GB RAM. The execution time for extracting TVP features from 173,409 FQDNs was 61 seconds, which was equivalent to 0.0004 seconds/FQDN. The file sizes of the domain name list database (SQL) were 1.4 GB in Alexa and 300 MB in hpHosts, respectively. Step 2 (appending DNS-based features) was executed as a MapReduce job on a Hadoop cluster, which had 2 master servers (16-core 2.4-GHz CPU, 128-GB RAM) and 16 slave servers (16-core 2.4-GHz CPU, 64-GB RAM). The execution time for extracting rIP features from 173,409 FQDNs was 20 hours (0.42 seconds/FQDN) and that for rDomain features was 96 hours (1.99 seconds/FQDN). The file size of the historical DNS logs used for extracting these DNS-based features was 212 GB. Step 3 (applying machine learning) was executed on the same server as step 1. The execution time for one-time training from 156,068 FQDNs was 28 seconds (0.0001 seconds/FQDN) and that for test from 17,341 FQDNs was 8 seconds (0.0005 seconds/FQDN). These evaluations prove the basic feasibility of our proposed system and reveal that step 2 requires far more resources and time to execute than steps 1 and 3. The reason for step 2's high cost is the size of the graphs for rIPs and rDomains. Today, some domain names used by hypergiants, such as Google, Amazon, and Akamai, have a huge number (over ten thousand) of rIPs and rDomains. This fact raises the problem of high cost for extracting common/typical DNS-based features. However, from the results explained in Section 3.4.3, this problem will be solved if our system makes a sacrifice of a TPR of 0.002 to use the feature set TVP instead of TVP+rIP+rDP. This is a trade-off between system performance and detection performance. Thus, we should configure our system based on this situation.

3.4.5 Predictive Detection Performance

We evaluated the predictive detection performance of DomainProfiler; that is, whether we can discover domain names that may be abused in *future*. The aforementioned evaluations were based on cross-validations (CVs); however, this section focuses on the evaluation of the detection performance of new malicious domain names that first appeared after March 1, 2015 by using only the information as of February 28, 2015. Specifically, we used the training set shown in Table 3.3 to create a learning model first then input the test set in Table 3.3 to evaluate the predictive detection performance. In this evaluation, we set the optimal parameters discussed in Section 3.4.2. The best feature set (TVP+rIP+rDomain) discussed in Section 3.4.3 was compared with the feature set (rIP+rDomain) that had only common/typical DNS-based features. Table 3.5 lists the evaluation results of using TVP+rIP+rDomain and Table 3.6 lists those of rIP+rDomain. Note that the test set only consists of malicious domain names; thus, there are no false positives (FPs) and true negatives (TNs) and their related evaluation criteria in the tables.

In terms of the true positive rate (TPR/recall), DomainProfiler and feature set (TVP+rIP+rDomain) achieved extremely high TPRs in all test sets; our system achieved TPRs of 0.985 in both Honeyclient-Exploit and Honeyclient-Malware. Moreover, our system accurately detected/predicted command and control (C&C) domain names in Sandbox-C&C and Pro-C&C, while our training set did not include labeled C&C domain names. This is not a surprising result because our TVP is designed to exploit the common characteristics of attackers' domain names. On the other hand, using only the typical features (rIP+rDomain) achieved a TPR of 0.402 at best. Comparing these results illustrates that our TVP features successfully contribute to predicting malicious domain names that will be used in future.

In terms of early detection of future malicious domain names, we investigated when the system can detect such domain names. Specifically, we analyzed the number of days that elapsed from February 28, 2015, when

the learning model was created, for malicious domain names to be detected with the system. For example, if the system correctly detected and identified a new malicious domain name on March 7, 2015, the elapsed number of days for the domain name is 7. Tables 3.5 and 3.6 also show the descriptive statistics of the elapsed days for malicious domain names for each feature set. Note that we only count domain names in the TP of each dataset. The descriptive statistics include the minimum (days_Min), first quartile (days_1stQu), which means the value cut off the first 25% of the data, second quartile (days_2stQu), which is also called the median and is the value cut off the first 50% of the data, the mean (days_Mean), the third quartile (days_3rdQu), which is the value cut off the first 75% of the data, and the maximum (days_Max). Table 3.5 shows that our proposed system (TVP+rIP+rDomain) can precisely predict future malicious domain names 220 days before the ground truth, such as honeyclients and sandbox systems, and identify them as malicious in the best case. Comparing the above results with Table 3.6 reveals that the common/typical feature set (rIP+rDomain) also detects malicious domain names early; however, the number of detected domain names (TP) is quite small. We conclude that our proposed system using TVPs outperforms the system using only the common/typical feature set from the perspective of both accuracy and earliness.

Table 3.5: Predictive Detection Performance of DomainProfiler (TVP +rIP+rDomain)

Dataset	**TP**	**FN**	**# FQDNs**	**TPR/ Recall**	**days_ Min**	**days_ 1stQu**	**days_ 2ndQu**	**days_ Mean**	**days_ 3rdQu**	**days_ Max**
Honeyclient-Exploit	529	8	537	0.985	16	140	176	164.5	197	220
Honeyclient-Malware	67	1	68	0.985	1	125	205	159.4	212	220
Sandbox-Malware	721	54	775	0.930	1	108	133	128.2	151	221
Sandbox-C&C	7,476	997	8,473	0.882	1	38	99	98.31	140	221
Pro-C&C	92	5	97	0.948	1	9	15	14.95	21	29
Pro-Phishing	75,583	2,638	78,221	0.966	1	9	14	14.04	19	28

Table 3.6: Predictive Detection Performance of Feature Set (rIP+rDomain)

Dataset	**TP**	**FN**	**# FQDNs**	**TPR/ Recall**	**days_ Min**	**days_ 1stQu**	**days_ 2ndQu**	**days_ Mean**	**days_ 3rdQu**	**days_ Max**
Honeyclient-Exploit	184	353	537	0.343	19	158	187	172.3	210	219
Honeyclient-Malware	5	63	68	0.074	1	60	63	108.4	205	213
Sandbox-Malware	197	578	775	0.254	3	112	133	128.3	153	221
Sandbox-C&C	2,491	5,982	8,473	0.294	1	47	108	103.9	144	221
Pro-C&C	39	58	97	0.402	1	9	12	14.18	20	29
Pro-Phishing	29,427	48,794	78,221	0.376	1	11	17	15.21	20	28

3.4.6 Effectiveness of Our Temporal Variation Patterns

We analyzed how our temporal variation pattern (TVP) features contribute to increasing the true positive rate (TPR) and true negative rate (TNR) simultaneously. We present some noteworthy case studies in our TVPs defined in Section 3.2. In this analysis, we used the TVP+rIP+rDomain feature set, and the setting and dataset were the same as the previous evaluation discussed in Section 3.4.5.

Alexa1M-Null: This TVP is intended to boost the TPR, as described in Section 3.2. Our analysis revealed that this TVP was especially effective for malicious domain names using a domain generation algorithm (DGA) and abusing new generic top-level domains (gTLDs) such as .xyz and .solutions. This is because these domain names are less likely to be within Alexa1M.

Alexa1M-Stable: This TVP successfully determined the characteristics of somewhat popular domain names to improve the TNR.

Alexa1M-Fall: This TVP is designed to detect the changing of malicious domain names to improve the TPR. We observed two major types of malicious domain names that fit this TVP. One type is expired domain names due to the termination of services or the merger and acquisition of companies. Some of these expired domain names were re-registered by third-party attackers to execute domain parking or cyberattacks. The other type is domain names that were changed from legitimate to malicious because the website of the domain name was not well-managed and in a poor state of security.

Alexa1M-Rise: This TVP attempts to grab the features of legitimate domain names of start-up websites to improve the TNR. We observed many domain names corresponding to this TVP such as those of new companies, services, movies, and products.

hpHosts-Null: This TVP successfully improved the TNR because the second-level domain (2LD) parts of popular/legitimate domain names were less likely to be listed in hpHosts.

hpHosts-Stable: This TVP is designed to determine the characteristics of malicious domain names abusing easy-to-use services, such as bullet-proof

hosting, to improve the TPR. For example, we observed many subdomains using a domain generation algorithm (DGA) under the same 2LD part such as `84c7zq.example.com`.

hpHosts-Fall: This TVP is intended to boost the TNR. We confirmed that some domain names under well-managed networks fit this TVP because the domain names were once abused and then quickly sanitized. In this case, the TVP contributed to the accurate prediction of future legitimate domain names.

hpHosts-Rise: This TVP is designed to help detect/predict malicious domain names more accurately to improve the TPR. We mainly observed two types of domain names that fit this TVP. One type is domain names that heavily used the DGA in both 2LD and third-level domain (3LD) parts of domain names, e.g., `14c2c5h8[masked].yr7w2[masked].com`. We observed that this type of 2LD will be continuously used for a while by attackers to create many subdomain names. The other type is domain names under free subdomain name services, which offer subdomain name creation under 2LD parts, such as .flu.cc and .co.nr. These services are easily abused by attackers in creating distinct domain names.

3.5 Discussion

This section discusses possible evasion techniques against DomainProfiler and issues when using the predicted malicious domain names generated from our system as countermeasures to protect users from cyberattacks.

3.5.1 Evading DomainProfiler

DomainProfiler is designed to exploit the temporal variation patterns (TVPs) of malicious domain names used by attackers. There are three possible evasion techniques of our system. One technique is to avoid using domain names as attack infrastructure. If attackers do not use domain names, it would be easier for us to take countermeasures such as just blocking them us-

ing IP addresses. The cost of changing IP addresses is much higher than that of changing domain names due to the limited address space. For instance, the address space of IPv4 is limited to 32 bits and that of IPv6 is to 128 bits. However, domain names can consist of 255 or less octets/characters [66], which means a maximum of a 2040-bit space.

Another evasion technique is to avoid all our features in a TVP, related IP address (rIP), and related domain name (rDomain) to hide malicious domain names from our system. For example, attackers can operate their domain names as real legitimate/popular services for a long time to evade our TVPs then use the domain names as their malicious infrastructure. However, this situation drives up the cost for implementing any attacks using domain names. Another example is border gateway protocol (BGP) hijacking, which potentially enables attackers to divert user traffic from real IP addresses to their IP addresses. In such a case, attackers may bypass our rIP or rDomain features; however, the BGP is basically used between Internet service providers, and it is difficult for normal Internet users/attackers to take effective control of it.

The other possible evasion technique is to use legitimate web services as legitimate users. For example, attackers would create dedicated accounts for some web services and use them as their command and control (C&C) channels. In such a case, only legitimate domain names are observed and our system cannot detect them. However, these accounts could be easily banned by the administrator of the web services, and the content sent/received by attackers could be easily analyzed to develop a new countermeasure.

3.5.2 DNS-based Blocking

DomainProfiler predicts and outputs malicious domain names. However, these domain names cannot always be blocked with a domain-name level. For example, malicious and legitimate websites can exist under the same domain name. Thus, blocking based on domain names instead of URLs may excessively block legitimate websites. To examine an actual condition, we

extracted and checked URLs under each predicted malicious domain name using a search engine API and commercial ground truth. In this examination, we manually analyzed 250 domain names randomly selected from the Honeyclient-Exploit dataset, as shown in Table 3.3, by using the following simple heuristics. If multiple URLs are found under the domain name, we consider that the domain name cannot be blocked with a domain-name level. On the other hand, if at most one URL is under the domain name, we determine that the domain name can be blocked with a domain name level. The examination results suggest that 72% (=180/250) of domain names can be effectively blocked using DNS-based blocking without excessive blocking of legitimate websites. Therefore, we conclude that malicious domain names output from our system can contribute to expanding DNS-based block lists if we consider the situation of URLs under the domain names.

3.6 Related Work

We summarize known approaches related to ours in terms of evaluating attack infrastructure or resources owned by attackers. Most of the studies are broadly divided into three approaches: lexical/linguistic, user-centric, and historic relationship. Note that these approaches were often combined in most of the studies we reviewed; thus, we classify them based on the main idea of each study.

3.6.1 Lexical/Linguistic Approach

The lexical/linguistic approach is focused on lexical or linguistic features obtained from malicious attack resources such as URLs and domain names.

Ma et al. proposed a learning approach using features from the lexical structure of malicious phishing URLs [26]. The focus with the approach for our system is not URLs but domain names, and our system can detect not only phishing but also other attacks.

Yadav et al. focused on linguistic features in command and control

(C&C) domain names generated using a domain generation algorithm (DGA) and developed an approach for detecting such malicious domain names [67]. While this approach detects C&C domain names containing random strings, our approach targets broader malicious domain names.

Szurdi et al. analyzed the nature of typosquatting domain names [68]. Typosquatting is generally defined as a technique to register similar domain names to popular domain names to profit from advertisements and perform phishing attacks. We take a different or more general approach with our system, and our temporal variation patterns (TVPs) can also take into account the nature of typosquatting domain names.

Felegyazhi et al. proposed to use WHOIS information of domain names such as registration and name servers to detect malicious domain names [25]. Our approach does not use WHOIS information due to the cost of retrieving it; however, we achieve a high true positive rate (TPR) of malicious domain names without WHOIS.

3.6.2 User-centric Approach

The user-centric approach focuses on user behavior of DNS traffic by observing passive DNS logs. Sato et al. used the co-occurrence characteristics of DNS queries to C&C domain names from multiple malware-infected hosts in a network to extend domain name blacklists [28]. Also, Rahbarinia et al. proposed a system called Segugio to detect new C&C domain names from DNS query behaviors in large ISP networks [6]. These systems require malware-infected hosts in a network; however, our approach works without malware-infected hosts.

Bilge et al. proposed a system called Exposure to detect malicious domain names based on the time-series changes of the number of DNS queries in passive DNS data [69]. Perdisci et al. proposed a system, FluxBuster, which detects previously unknown fast-flux domain names by using large-scale passive DNS data [70]. The cost of Exposure or FluxBuster for retrieving and analyzing large-scale passive DNS logs is much larger than that of our TVPs

in DomainProfiler.

Antonakakis et al. proposed the system Kopis, which uses user behavior observed in passive DNS logs on authoritative DNS servers [71]. Today, the number of new generic top-level domains (gTLDs) is rapidly increasing; thus, it is more difficult to exhaustively gather such information on a TLD's authoritative DNS servers. DomainProfiler does not require such logs and is designed to use publicly available information.

Antonakakis et al. also proposed a system called Pleiades that is focused on DNS queries to non-existent domain names observed on recursive DNS servers to detect DGA domain names used for C&C [72]. In addition, Thomas et al. proposed a system similar to Pleiades to determine the characteristics of non-existent domain names [73]. Our system does not require such DNS logs and is focused on not only C&C domain names but also other malicious domain names such as drive-by download and phishing.

3.6.3 Historic Relationship Approach

The historic relationship approach is focused on the historic or time-series information of domain names, IP addresses, and web content.

Antonakakis et al. proposed a system called Notos to detect malicious domain names that have similar patterns to past malicious domain names [4]. This was one of the most successful studies on domain-name evaluation or reputation systems. Notos uses historic IP addresses and historic domain names to extract effective features to discriminate malicious domain names from legitimate ones. As stated in Section 3.3.2, we use these features as some of our features in related IP addresses (rIPs) and related domain names (rDomains). Moreover, our TVP features dramatically expand detection and prediction performance, as discussed in Section 3.4.

Manadhata et al. proposed a method for detecting malicious domain names from event logs in an enterprise network by using graph-based analysis [74]. Boukhtouta et al. proposed an analyzing method for creating graphs from sandbox results to understand the relationships among domain names,

IP addresses, and malware family names [75]. Kührer et al. proposed a method for identifying parked and sinkhole domain names from websites and blacklist content information by using graph analysis [5]. DomainProfiler strongly relies on the TVP or time-series information, which these studies did not use, to precisely predict future malicious domain names.

Chiba et al. used the characteristics of past malicious IP addresses to detect malicious websites [76]. Our system uses not only IP address features (rIPs) but also TVPs to precisely detect malicious domain names.

Venkataraman et al. developed a method for inferring time-series shifting of IP address prefixes to detect malicious IP addresses used for spam or botnet [77]. DomainProfiler is also focused on the idea of shifting malicious resources; however, the target and method are completely different.

The closest concept to ours is that proposed by Soska et al. [78]. They focused on the idea of variations in compromised websites using a popular content management system and proposed a method for predicting vulnerable websites before they turn malicious. The main features they rely on are content-based features obtained from compromised websites. The concept of DomainProfiler seems to be similar; however, our system has an advantage in scalability because it does not need to access websites and extract features from them. Moreover, the focus with our system is wider; that is, our system can detect websites related to drive-by download and phishing attacks.

Lever et al. pointed out a problem in re-registration of expired domain names and developed an algorithm called Alembic to find potential domain-name ownership changes using passive DNS data [79]. We have also focused on the temporal changes of domain names including such re-registered domain names [80]. However, our system does not rely on passive DNS data and the goal with our system is not only finding re-registered domain names but also specifying truly malicious domain names abused by attackers.

Recently, Hao et al. proposed a system called Predator to predict future malicious domain names when they are registered [81]. Their system

uses domain registration information directly obtained from the .com TLD registry (VeriSign, Inc). However, more and more new gTLDs (e.g., .xyz and .top) have been started to use since October 2013. Now the number of such new gTLDs is 1,184 as of September 2016 [82]. Attackers also leverage new gTLDs for their cyberattacks. For example, Halvorson et al. showed that domain names using new gTLDs are twice as likely to appear on blacklists [83]; that means attackers actively make use of new gTLDs nowadays. Obviously, to keep up with such situations, their system needs to obtain real-time access privileges to highly confidential data inside the each new gTLD's registry. While the concept of Predator resembles DomainProfiler, the mechanism is totally different because our system does not require any data only owned by a registrar, registry, and authoritative name server.

3.7 Conclusion

We proposed a system called DomainProfiler to detect/predict potential malicious domain names in future. Our key idea behind the system is to exploit temporal variation patterns (TVPs) of malicious domain names. A TVP of domain names includes information about how and when a domain name has been listed in legitimate/popular and/or malicious domain name lists. Our system actively collects DNS logs, identifies their TVPs, and predicts whether a given domain name will be used for a malicious purpose. Our evaluation with large-scale data revealed that our system can predict malicious domain names 220 days beforehand with a true positive rate (TPR) of 0.985. DomainProfiler will be one of the ways to track the trend in ever-changing cyber security threats.

Chapter 4

Profiling Generated Structures of Domain Names

4.1 Introduction

Domain names came into existence in the 1980s and have become an integral part of today's Internet. While the Internet cannot virtually function without domain names, attackers also use domain names and the domain name system (DNS) as reliable, instantaneous, and distributed infrastructure for conducting attacks. For example, attackers register similar domain names to legitimate services or popular brands to deceive users into downloading malware or unwanted programs. Another common example is the so-called command and control (C&C), wherein attackers use domain names as rendezvous points of malware-infected hosts to control them and launch other attacks such as denial-of-service (DoS) attacks and spam emails.

Countermeasures such as detection and filtering of malicious domain names owned/used by attackers have been studied and implemented for many years [4, 7, 80]. Nevertheless, abuse of new domain names has continued and remains a significant threat even today. Moreover, there is no single common defense solution against domain name abuses because each malicious domain name has different characteristics. If we fail to choose the right countermeasure for each malicious domain name, it will be basically ineffective in practice. For example, some malicious domain names are created by abus-

ing legitimate services including online advertising and web hosting. If we filter domain names used in such legitimate services, we may prevent users from accessing legitimate services and disrupt legitimate businesses. Other malicious domain names are purposely set up by an algorithm called domain generation algorithm (DGA) or an intention to deceive users. If we hesitate to filter these domain names, we cannot decrease the threat of cyberattacks.

A key challenge is to determine the optimal defense solution for each malicious domain name. This chapter is intended to reveal *what, where*, and *how* countermeasures need to be taken against such malicious domain names. We focus on the relationships between domain name categories and practical defense solutions to determine how best to utilize detected malicious domain name information. In reality, there is a significant distance from simply detecting malicious domain names to utilizing them for making the Internet safer. In particular, malicious domain names differ significantly depending on their characteristics such as the hierarchical structure of the domain names, the back-end services offering them, and the operational situations. Thus, we need to consider these characteristics for each malicious domain name and determine the best defense solution to prevent filtering any legitimate services or businesses. Therefore, we should first categorize or classify malicious domain names according to their characteristics and then determine the defense solution suitable for each domain name.

In this chapter, we design and implement a unified and objective analysis pipeline combining existing defense solutions for realizing practical and optimal defenses against malicious domain names. Our novel analytical approach is referred to as malicious domain names' *chromatography* that is used for the separation of mixtures comprising various types of malicious domain names for websites. On the basis of this concept, we do not create a hodgepodge of existing solutions but design separation of malicious domain names and offer defense information by considering the characteristics of the malicious domain names as well as the possible defense solutions and points of defense.

Our main contributions are summarized as follows.

- We develop a taxonomy of malicious domain names and provide systematized knowledge of the latest defense solutions with points of defense.
- We design and implement a new analytical method, which systematically determines the optimal defense solution for each malicious domain name.
- We evaluate our analysis pipeline and output defense information using a large and real dataset to show both the effectiveness and validity of our proposed approach. In particular, we are the first to show that over 70% of malicious domain names need only DNS-level defense with no collateral damage of legitimate accesses.

The rest of this chapter is organized as follows. In Section 4.2, we show a taxonomy of malicious domain names and consider the possible defense solutions and points of defense. We discuss our proposed analysis pipeline DomainChroma in Section 4.3. Section 4.4 shows the detail explanation of datasets and the evaluation results. Finally, we conclude this chapter in Section 4.5.

4.2 Chromatography of Domain Names

We define a new concept of malicious domain names' *chromatography*. Traditionally applied in chemistry, chromatography separates a mixture into its components based on their different chemical characteristics. Our chromatography handles mixtures comprising various types of malicious domain names for websites. We explore the separation of malicious domain names and offer defense information by considering both the characteristics of the malicious domain names and possible points of defense.

4.2.1 Characteristics of Malicious Domain Names

In most cyberattacks, domain names are used to deliver malicious content (e.g., exploit content and malware) and to command/control malware-

infected hosts because domain names and DNS are easy to use, reliable, and distributed systems. We investigated the characteristics of malicious domain names to further classify them into two categories, i.e., *compromised* and *dedicated*.

Compromised. This category contains malicious domain names abusing legitimate services. We define this category because such malicious domain names are originally intended to offer legitimate services to Internet users, and we should not simply stop such domain names and filter accesses to them. We explore this compromised category and show how to classify this category later in Section 4.3.2.

Dedicated. This category contains malicious domain names prepared exclusively for malicious purposes. We consider this category because malicious domain names differ significantly in character relative to compromised domain names. Based on this idea, we should clearly distinguish dedicated malicious domain names from compromised domain names in terms of providing mitigations or countermeasures. We show the details of this dedicated category and illustrate how to classify this category later in Section 4.3.2.

4.2.2 Points of Defense

We summarize possible points of defense and corresponding defense methods against malicious domain names in terms of building a realistic remedy strategy. Specifically, we divide these points into two levels: *HTTP-level* and *DNS-level*.

HTTP-level Points of Defense. This level includes three components that are involved in web communication using HTTP/HTTPS, i.e., security appliances, web servers, and search engines. Although this chapter primarily focuses on domain names and DNS, we also consider HTTP-level defenses because there is a close connection between HTTP and DNS. For example, most HTTP communication employs DNS name resolution; thus, we can defend against malicious HTTP communication at both DNS and HTTP levels. The following three components can only defend against attacks us-

ing malicious domain names if attacks use HTTP, e.g., distributing exploit content or malware using websites, hosting phishing websites, and operating malware-infected hosts using C&C.

Security appliances include HTTP proxies, network intrusion detection or prevention systems (NIDS/NIPS), and deep packet inspection (DPI). They can filter HTTP communication pointing to malicious domain names.

Web servers on the Internet can be critical points of defense if they serve malicious content. The defense on web servers is straightforward, i.e., deleting corresponding malicious content on the web servers. To deploy this defense, abuse reports can be sent to content owners, server administrators [84], and national or regional computer emergency response team (CERT) organizations.

Search engines are one of the most frequently used web applications. Most users access websites based on search results generated by a search engine. If search engines display a link to malicious content, users are susceptible to cyberattacks. Thus, search engines should filter links that point to malicious content.

DNS-level Points of Defense. This level includes three major components in DNS, i.e., caching name servers, authoritative name servers, and domain registries/registrars.

Caching name servers are generally deployed in local area networks or Internet service provider (ISP) networks. The defense in caching name servers is primarily filtering user access to malicious domain names based on domain name blacklists.

Authoritative name servers are deployed in each DNS zone, primarily at the second level domain (2LD) level. The defenses in authoritative name servers include filtering and updating zone data. Filtering can be used to block DNS queries pointing to a blacklisted domain name to prevent users from accessing a malicious domain name. Updating zone data involves deleting a malicious domain name record such that the domain name cannot be resolved.

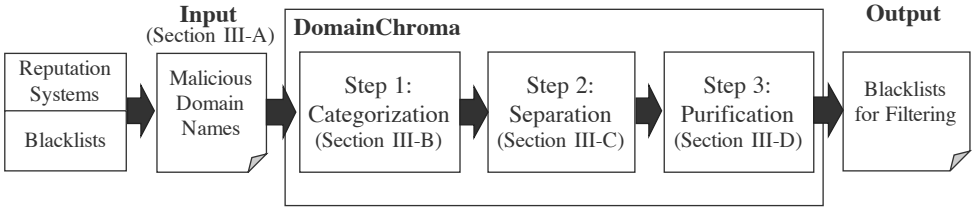


Figure 4.1: Overview of Our Analysis Pipeline DomainChroma

Domain registries/registrars manage domain name registrations. The registry manages the registration of domain names within large DNS zones, such as top-level domains (TLDs). The registrar is a service provider that connects the registries to manage domain databases. Malicious domain names can be defended by anti-abuse actions on the master database of each registry, such as deleting a domain name. For example, VeriSign, Inc., which is a `.com` and `.net` registrar, can take actions by complying with court orders and law enforcement [85].

4.3 Analysis Pipeline: DomainChroma

We design a new analysis pipeline called DomainChroma that considers both characteristics of malicious domain names shown in Section 4.2.1 and each point of defense shown in Section 4.2.2. Our design aims to systematize the knowledge of state-of-the-art defense techniques, enabling detection of malicious domain names and optimal defense actions. Specifically, we attempt to reveal *what, where,* and *how* countermeasures need to be taken against malicious domain names, thus securing domain names and DNS for legitimate Internet users. To this end, we implement the *chromatography* concept shown in Section 4.2. Fig. 4.1 is an overview of our analysis pipeline DomainChroma. The DomainChroma pipeline consists of three steps: *categorization*, *separation*, and *purification*. The order of these steps is designed to reflect the demand in defense operations against malicious domain names. The step-by-step details of each step are presented in the following

sections.

4.3.1 Input: Malicious Domain Names

The DomainChroma input is expected to contain malicious domain names provided by domain reputation systems [4, 7, 80] or any kinds of domain name blacklists, with a low probability of obvious legitimate domain names. Such malicious domain names are engaged in various types of cyberattacks (e.g., drive-by download, malware download and C&C, and phishing). At this point, we know *what* domain names to be targeted for actions, but *where* and *how* these actions should be implemented are unknown. Our analysis pipeline proceeds through the following steps to decide the actions for optimal defense against malicious domain names.

4.3.2 Step 1: Categorization

The first step of our analysis pipeline DomainChroma is categorization. The input malicious domain names are categorized into *compromised* or *dedicated* since these two categories have different characteristics in terms of deciding countermeasures as shown in Section 4.2.1.

Compromised

We define that compromised malicious domain names compose of the following three sub-categories, i.e., advertising, content delivery network (CDN), and web hosting. This section explains definitions of these sub-categories and how to identify them.

Advertising. Online advertisements are commonly used in most websites to generate revenue. Online Advertising is not intended to abuse domain names nor engage in cyberattacks. However, attackers have used this ecosystem as an attack vector to reach target users effectively. Thus, we categorize advertising domain names not to filter legitimate advertisements excessively. To identify domain names in advertising ecosystems, we use pre-defined information used in previous work [86].

CDN. A CDN delivers web content to end users with a distributed and efficient infrastructure. It is essentially used by legitimate users or companies; however, attackers have used it as reliable infrastructure to distribute malicious content. Thus, we should filter specific content or the URLs rather than the CDN domain names to prevent users from accessing malicious content while maintaining legitimate services. To do this, we identify such CDN domain names using the previously reported information [87].

Web Hosting. We define that web hosting involves domain names that use shared hosting services, including cloud services and file sharing services. Since web hosting is an economical option for users to host a website, the number of web hosting services is increasing dramatically. Attackers use web hosting services as an attack infrastructure, e.g., to host malicious websites or malware. In this chapter, we define web hosting as domain names that have multiple different owner's URLs and identify such domain names based on our heuristic rules about known web hosting providers.

Dedicated

We define that dedicated malicious domain names compose of the following nine sub-categories, i.e., domain generation algorithm (DGA), re-registration, sinkholing, parking, typosquatting, no-URLs, dynamic DNS, free, and domain hosting. We illustrate definitions of these sub-categories and how to classify them.

DGA. A DGA dynamically produces domain names primarily used as rendezvous points between attackers and victims or malware-infected hosts. A domain name generated by the algorithm is called an automatically generated domain (AGD) [88]. DGAs generate a huge number of distinct AGDs and then use only a small subset of generated domain names for their actual malicious activities, such as C&C communication. Attackers use a DGA to make blacklisting or taking down C&C infrastructure infeasible. DGAs are only used for malicious purposes, and we should filter domain names generated by DGAs to specify malware-infected hosts and prevent users from

cyberattacks. To identify DGAs, we follow the previous work that leveraged the linguistic features of DGAs [89].

Re-registration. This sub-category contains maliciously re-registered domain names that were originally legitimate domain names. For a detailed explanation of the re-registration process, the reader can refer to recent work [81]. Expired domain names, particularly popular domain names, tend to be targeted and immediately re-registered by attackers for malicious purposes, such as phishing attacks. To detect such re-registered malicious domain names, we use crawling/parsing WHOIS data and detecting re-registration events based on our heuristic rules.

Sinkholing. Security researchers and organizations often use a countermeasure called sinkholing to take control of malware C&C domain names. In terms of defending users, we should identify sinkholed malicious domain names because such domain names will not be used for legitimate services. To identify such sinkholed domain names, we use sinkholing-specific features proposed in precious work [5].

Parking. Parking is a service used to monetize currently unused domain names that display advertisements. Attackers tend to use such parking services to monetize malicious traffic resulting from malware infection or phishing attacks. Parked domain names used for malicious purposes should be filtered to hamper monetization related to cyberattacks. To identify such parked domain names, we use recently published parking-specific information [90].

Typosquatting. Typosquatting is generally defined as an attacking technique to register similar domain names to popular or legitimate services. Such typosquatting domain names are almost always created for non-legitimate purposes; thus, we should detect and filter such domain names and the traffic directed to them. To detect such malicious domain names engaged in typosquatting, we design a typosquatting classifier based on previous work [68].

No-URLs. We newly define no-URLs as domain names that are considered only in the DNS protocol and have no URLs under the domain name. A ma-

licious domain name that has no URLs can immediately be filtered because there is no risk of excessively filtering legitimate websites. To identify this sub-category, we use a search engine API to check whether the domain name has any URLs.

Dynamic DNS. Dynamic DNS services allow Internet users to register subdomains under their specific domain names and resolve the names of subdomains and IP addresses. Dynamic DNS is used by legitimate users and abused by attackers. Thus, when considering defenses against malicious domain names that employ dynamic DNS, we should take care not to stop dynamic DNS services. To this end, we refer to previous work [91] to detect such dynamic DNS domain names based on pre-defined rules.

Free. This sub-category contains domain names created by free domain registration services. For example, a freely available domain name can be registered under some TLDs or some second level domains (2LDs). Such services are easily abused by attackers to create malicious domain names. To identify domain names created for free, we use prior knowledge offered by previous work [91].

Domain Hosting. Domain hosting is very similar to the previously described web hosting sub-category. In this chapter, the difference between web hosting and domain hosting is the structure of domain names and URLs. In a web hosting case, an individual directory can be created (e.g., `/~user1`). On the other hand, in a domain hosting case, subdomains or fully qualified domain names (FQDNs) can be created under the hosting service's domain names (e.g., `user1.example.org`). We should recognize these two patterns because we consider optimal countermeasures for each domain name in order to avoid filtering legitimate accesses. To classify domain hosting, we use our heuristic rules of known domain hosting patterns.

4.3.3 Step 2: Separation of Mixtures

The second step of our analysis pipeline DomainChroma is separation of mixtures comprising malicious domain names. Once the input domain names

are categorized in step 1, we can determine *where* the actions against each input domain name should be performed. We use a conditional procedure based on our pre-defined rules. These rules assign the points of defense to the corresponding categories. Specifically, the input domain names are separated into two groups: one *requiring HTTP-level defenses*, the other *requiring DNS-level defenses*. These groups correspond to the points of defense summarized in Section 4.2.2.

Domain names requiring HTTP-level defenses fall into the *compromised* category, namely, *advertising*, *CDN*, and *web hosting*, as defined in Section 4.3.2. These domain names are defined as compromised because they originally referred to legitimate services. Thus, within this group, our actions should not target domain names alone but should use additional information such as URLs pointing to specific malicious content or files.

Conversely, domain names requiring DNS-level defenses fall into the *dedicated* category, namely, *DGA*, *re-registration*, *sinkholing*, *parking*, *typosquatting*, *no-URLs*, *dynamic DNS*, *free*, and *domain hosting*, as defined in Section 4.3.2. These domain names are defined as dedicated because they are exclusively prepared for malicious purposes and can be directly targeted at DNS-level points of defense.

From the category/sub-category identification result of each input domain name and our rules, we can decide *where* to apply defense solutions for the domain name. If a domain name falls into multiple sub-categories in both HTTP and DNS levels (e.g., web hosting and typosquatting), we assign it to the HTTP-level to reduce the risk of collateral damage caused by excessive filtering of legitimate communication. In an organization, the assignment will depend on the operational policy of the organization. Our conditional procedure is easily tunable for this purpose.

4.3.4 Step 3: Purification

Step 3, *purification*, decides *how* each domain name should be used for the optimal defense. Similarly to step 2, we match defense strategies to categories

through our pre-defined rules. Specifically, we purify the domain names requiring DNS-level defenses or further separate them into two subgroups: those *requiring FQDN-level defenses* and those *requiring 2LD-level defenses*.

Here, we stipulate that domain names in the sub-categories *dynamic DNS*, *free* and *domain hosting* require FQDN-level defenses, whereas those in the sub-categories *DGA*, *re-registration*, *sinkholing*, *parking*, *typesquatting* and *no-URLs* require 2LD-level defenses. Domain names in the former group are defined as requiring FQDN-level defenses because their hierarchical structure means that users can create subdomains or FQDNs under the 2LDs owned by the providers of the three sub-categories in this group, as explained in Section 4.3.2. Meanwhile, domain names in the latter group (which includes all other categories requiring DNS-level defenses in Step 2), are defined as requiring 2LD-level defenses because they are almost certainly registered with malice. Within the subgroup requiring 2LD-level defenses, we extract and process the 2LD parts of the input domain names, which can be more effectively filtered than the FQDNs.

4.3.5 Output: Blacklists for Filtering

DomainChroma outputs blacklists for filtering for each point of defense defined in Section 4.2.2. The defense information in blacklists filters the user's accesses to malicious domain names at both HTTP-level and DNS-level points of defense.

HTTP-level points of defense include security appliances, web servers, and search engines, as explained in Section 4.2.2. Security appliances, which mainly monitor the HTTP protocol, can identify URLs requiring HTTP-level defenses. As explained in Section 4.3.3, such URLs are created using the additional information of specific malicious content or files. The URLs referred by security appliances can also be referred by web servers and search engines, which can then filter malicious contents.

DNS-level points of defense include caching name servers, authoritative name servers, and the domain registry/registrar as explained in Section 4.2.2.

At caching name servers in a local organization or an ISP, blacklists can prevent users from accessing malicious domain names. The blacklists used in caching name servers include both FQDN and 2LD lists respectively correspond to the domain names requiring FQDN-level and 2LD-level defenses output by Step 3 of DomainChroma. To minimize collateral damage of legitimate accesses, DomainChroma selects only the malicious domain names that are defendable at the DNS-level. Authoritative name servers can also filter answers to DNS queries pointing to blacklisted domain names, preventing their accesses by users. However, because authoritative name servers are deployed at each DNS zone, the efficiency of blacklisting is much lower for authoritative name servers than for caching name servers. At the domain registry/registrar, filtering blacklists can also be referred to manage the domain-name database directly.

4.4 Evaluation

We implemented and evaluated our analysis pipeline DomainChroma on real datasets containing numerous malicious domain names.

4.4.1 Dataset

We prepared an input dataset of malicious domain names presented in Table 4.1. As explained in Section 4.3.1, the expected input to DomainChroma is a dataset of malicious domain names provided by domain reputation systems [4, 7, 80] or any domain name blacklists. We used the six types of malicious domain names listed in Table 4.1. These malicious domain names were composed of *truly* malicious domain names confirmed by a client-based honeypot (honeyclient), a sandbox system, and commercial and professional services provided by a security vendor. The *Honeyclient-Exploit* and *Honeyclient-Malware* types contained malicious domain names related to drive-by download attacks detected by our honeyclient which regularly crawls public blacklists [57], some commercial blacklists from March 2015

Table 4.1: Dataset of Malicious Domain Names

Type	Period	# FQDNs
Honeyclient-Exploit	2015-03-01–2015-10-07	537
Honeyclient-Malware	2015-03-01–2015-10-07	68
Sandbox-Malware	2015-03-01–2015-10-07	775
Sandbox-C&C	2015-03-01–2015-10-07	8,473
Pro-C&C	2015-03-01–2015-03-29	97
Pro-Phishing	2015-03-01–2015-03-29	78,221
Total		88,171

to October 2015. More precisely, the malicious domain names collected in Honeyclient-Exploit were distributing exploit content during drive-by download attacks. Honeyclient-Malware was composed of malicious domain names responsible for distributing malware samples. The malicious domain names in *Sandbox-Malware* and *Sandbox-C&C* were observed in a sandbox system running malware samples. These samples were randomly downloaded from VirusTotal [60] daily and consisted of newly submitted (within 24 hours) malicious executable files used in Microsoft Windows. Specifically, the malicious domain names in Sandbox-Malware were connected by malware downloader samples, enabling the download of other malware samples. Sandbox-C&C contained the C&C servers' domain names detected in the sandbox. The malicious domain names in *Pro-C&C* and *Pro-Phishing* were captured from C&C and phishing websites in March 2015 by a commercial and professional security service.

4.4.2 Output of DomainChroma

The six types of malicious domain names shown in Table 4.1 were input to DomainChroma, and the outputs were evaluated. This evaluation reveals the relationships between our defined domain-name categories/sub-categories and the various attack types (e.g., drive-by download, malware download and C&C, and phishing).

Table 4.2 summarizes the output of DomainChroma; namely, the identified points of defense and levels of defense information of the input do-

Table 4.2: Summary of Output of DomainChroma

Dataset	# FQDNs (HTTP-level)	# FQDNs (DNS-level /2LD-level)	# FQDNs (DNS-level /FQDN-level)	# Total Input FQDNs
Honeyclient-Exploit	16	369	152	537
Honeyclient-Malware	2	59	7	68
Sandbox-Malware	106	549	120	775
Sandbox-C&C	1,939	5,015	1,519	8,473
Pro-C&C	30	52	15	97
Pro-Phishing	22,031	48,589	7,601	78,221
Total	24,124	54,633	9,414	88,171

main names. The points of defense are HTTP-level (security appliances, web servers, and search engines) and DNS-level (caching name servers, authoritative name servers, and domain registries/registrars), as explained in Section 4.2.2. The level of defense information is the required granularity level (2LD-level or FQDN-level). From the DomainChroma output, we can provide blacklists to points of defense in real operations. As mentioned in Section 4.3.3, if a domain name belongs to multiple sub-categories in both HTTP and DNS levels, DomainChroma selects the HTTP-level to reduce the risk of collateral damage of legitimate accesses. Among the FQDNs, 24,124 (27.4% of the input FQDNs) required HTTP-level defenses, 54,633 (62.0% of the input FQDNs) required DNS-level defenses with 2LD-level domain information, and 9,414 (10.7% of the input FQDNs) required DNS-level defenses with FQDN-level domain information. These results indicate that when applying the output of DomainChroma, over 70% of the domain names engaged in various types of cyberattacks could be effectively defended only at DNS-level points of defense. This result is surprising and newly observed. In the Honeyclient-Exploit and Honeyclient-Malware cases, which correspond to drive-by download attacks, only approximately 3% of the input FQDNs require HTTP-level defenses, although both types are web-based attacks targeting web browsers and their plugins. This finding is possibly explained by the activities of recent attackers, who tend to prepare new

dedicated domain names for hosting their exploit kits and malware samples in drive-by download attacks. In the malware activities Sandbox-Malware, Sandbox-C&C, and Pro-C&C, up to 86% of the input FQDNs require DNS-level defenses. In the Pro-Phishing case, which corresponds to web phishing attacks, around 70% of the input FQDNs require DNS-level defenses, and only 9.7% require FQDN-level defense information. The low percentage of inputs requiring FQDN-level information is attributed to the self-registration and self-use of dedicated 2LDs in most of the recent phishing sites.

We investigated the risk of collateral damage when applying the DNS-level blacklist output from DomainChroma. Specifically, we checked the existence of legitimate domain names and URLs under each blacklisted domain name by leveraging passive DNS database (DNSDB) [92] and search engine APIs. We confirmed that the blacklists generated by DomainChroma incurred no collateral damage; that is, no legitimate domain names or URLs were falsely filtered by DomainChroma. Recall that DomainChroma was designed to prevent such situations by analyzing the characteristics of the domain names and points of defense. These results indicated that, by combining various techniques and considering both defense solutions and points of defense, DomainChroma generates optimal blacklists that do not inconvenience legitimate users.

4.5 Conclusion

In this chapter, we designed and implemented a unified and objective analysis pipeline called DomainChroma to reveal *what, where,* and *how* countermeasures should be taken against malicious domain names for websites. DomainChroma applied malicious domain names' *chromatography* that is newly defined here to mean the separation of mixtures comprising various types of malicious domain names. Based on this idea and systematized knowledge, we combined state-of-the-art research efforts and developed the analysis pipeline to offer practical and optimal defenses against today's malicious domain names without collateral damage of legitimate services. We

evaluated DomainChroma using a large and real dataset to show that over 70% of domain names need only DNS-level defense with no collateral damage of legitimate accesses. We hope that the knowledge and results in this chapter can be used to improve both the techniques and operations in DNS-level and HTTP-level points of defense to defend attacks using domain names and DNS in the future.

Chapter 5

Profiling Invariable Keywords in HTTP Communications

5.1 Introduction

Ever-evolving malware is a root cause of recent cyberattacks. Malware-infected hosts are controlled by attackers in such a way that they become accomplices in various cyberattacks. Communications and infrastructure between attackers and infected hosts are called command and control (*C&C*). An infected host controlled by C&C is called a *bot*, and a group of bots connected via C&C is called a *botnet*. Attackers transmit attack instructions to bots to carry out cyberattacks, and the results of attacks by bots are transmitted to attackers via C&C. Botnets enable attackers to conduct cyberattacks while keeping their existence untraceable. Thus, botnets are one of the most serious threats in the cyber security field.

C&C is an essential function of a botnet, i.e., C&C communications from bots must occur in a network. C&C communications have involved various protocols such as IRC, HTTP, P2P, and HTTP+P2P [93]. Recently, attackers have tended to use more general protocols for their C&C communications to prevent them from being analyzed or detected. Thus, over 60% of botnets use HTTP or HTTP+P2P as their C&C protocol [94]. In enterprise networks, in particular, filtering for outbound traffic is applied to limit their protocol only to HTTP and HTTPS. Therefore, using HTTP as a C&C

protocol is effective for attackers.

Countermeasures against botnet consist of defeating botnets themselves, referred to as a *takedown*, or detecting bots or C&C communications in a botnet. For example, takedowns of the Zeus botnet, the Citadel botnet, and the GameOver Zeus botnet were conducted in 2012, 2013, and 2014, respectively [95, 96, 97]. Takedowns can help prevent cyberattacks. However, they are not easily implemented because it is necessary to keep precise track of C&C infrastructure and to ensure cooperation among relevant organizations.

Therefore, detecting infected hosts on a particular network is necessary to mitigate cyberattacks, and the importance of this has significantly increased recently. This countermeasure can be divided into two categories: host-based and network-based. When an infected host is under the control of an attacker, any host-based countermeasure, such as antivirus software, has been disabled. In this case, network-based countermeasures are more effective, and many such countermeasures have been focused on C&C communications. One countermeasure is to blacklist known C&C domain names or URLs. Matching the communications with the blacklists makes it possible to detect C&C communications and infected hosts in a network.

Matching communications with blacklists is not always successful because attackers evade blacklists by changing all or part of their C&C domain names and URLs, namely their hostnames, domain names, URL paths, and URL queries. For example, some attackers use a domain generation algorithm (DGA) to change domain names effectively. Polymorphic URLs, which attackers generate to evade blacklists, tend to have similar patterns because attackers reuse their web servers or use the same toolkit.

Research on generating network-based signatures or templates, which involves the use of regular expressions to detect polymorphic patterns, has been conducted to use these patterns in URLs. For example, Xie et al. proposed a system AutoRE to generate signatures with regular expressions to detect the polymorphic URLs in spam emails sent from botnets [98]. Perdisci et al. proposed a method of generating signatures with regular expressions

to detect the URLs used in C&C communications based on HTTP traffic captured in a controlled environment, which is a sandbox system to dynamically analyze malware samples [1]. Nelms et al. improved upon the above research [1] and proposed a system called ExecScent, which introduced the concept of *templates* that can cover not only URLs but also HTTP request headers [13].

However, a potential problem with such signatures or templates is that they may falsely regard benign communications as malicious, resulting in false positives, due to an inherent aspect of regular expressions. Given that the cost of dealing with malware infection is high, false positives should be kept to a minimum. The cost can be enormous in organizations. A study showed that an average of 395 hours a week is spent responding to false alerts of malware infection, which costs about $1.27 million per year [99].

We therefore propose a system, called BotProfiler, to generate templates that cause fewer false positives than with the conventional system ExecScent [13] in order to achieve more accurate detection of malware-infected hosts. We focused on the key idea that malicious infrastructures, such as malware and C&C, are prone to be reused instead of created from scratch. Our research verifies this idea and proposes here BotProfiler to profile the variability of substrings in HTTP requests. BotProfiler identifies invariable keywords based on the same malicious infrastructures and makes it possible to generate more accurate templates.

Our main contributions are as follows:

- We propose a system called BotProfiler that generates templates to detect infected hosts after profiling invariable substrings of URL paths, URL queries, and user agents in HTTP requests from infected hosts.

- Our research presents the first-ever analysis of the existence and its reason of invariable substrings used by attackers and contributes to automatically generating more accurate and valuable templates than ExecScent.

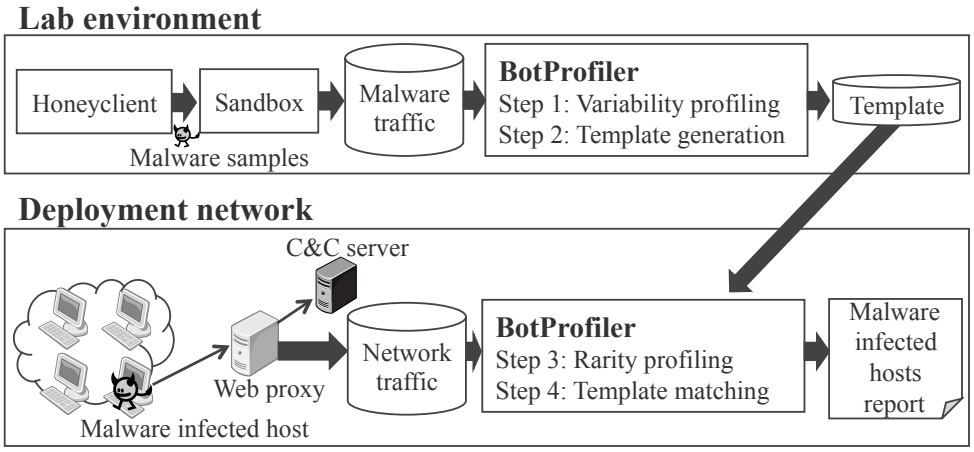


Figure 5.1: BotProfiler system overview

- The effectiveness of BotProfiler was validated with actual large traffic datasets. The results indicated that it reduced the number of false positives by up to two-thirds compared to ExecScent and even increased the detection rate of infected hosts.

The rest of this chapter is organized as follows. The detection methodology of BotProfiler is introduced in Section 5.2. The datasets and the results of our evaluation are described in Section 5.3. The limitation of our system is discussed in Section 5.4. Section 5.5 reviews related work. Finally, Section 5.6 concludes this chapter.

5.2 Detection Methodology

5.2.1 System Overview

BotProfiler generates templates to detect infected hosts in a network. Figure 5.1 is an overview of the system. It involves four steps; step 1: variability profiling, step 2: template generation, step 3: rarity profiling, and step 4: template matching. This matching concept was originally introduced with the conventional system ExecScent; however, BotProfiler generates

more precise and valuable templates than ExecScent. Steps 1 and 2 involve generating templates from outbound traffic captured in our sandbox system [100] running malware samples (malware traffic). Our malware samples were obtained from our high-interaction honeyclient [18, 101]. Steps 3 and 4 involve matching traffic with templates based on two criteria: the similarity to the templates and the *rarity* of each element in the templates. For example, an element that has high rarity means that it appears very infrequently in a deployment network. BotProfiler determines the rarity to use the characteristics of modern malware samples, which tend to affect a limited percentage of all hosts in a network, and to reduce false positives. Figure 5.1 also shows that the architecture of BotProfiler is divided into a lab environment and deployment network. The lab environment requires a honeyclient and a sandbox; however, the deployment network does not require them and only uses templates from the lab environment. This architecture readily enables us to deploy BotProfiler in multiple deployment networks. The details of BotProfiler are explained step by step in the following sections.

5.2.2 Step 1: Variability Profiling

Step 1 enables us to generate templates with regular expressions that cause fewer false positives than ExecScent. We focused on the key idea that malicious infrastructures, such as malware and C&C, tend to be reused instead of created from scratch. On the basis of this idea, in step 1, the variability of substrings in malware-generated HTTP requests is profiled to identify *invariable keywords* and variable substrings. Figure 5.2 shows the detailed procedure of step 1. First, substrings composed of two or more characters in URL paths, URL queries, and user agents in HTTP requests of malware traffic are extracted as *candidate keywords*. From the candidate keywords, *invariable keywords* are then detected based on the number of malware samples using the keywords. If there are more malware samples that use the same candidate keyword, it is more likely that the keyword is invariable and

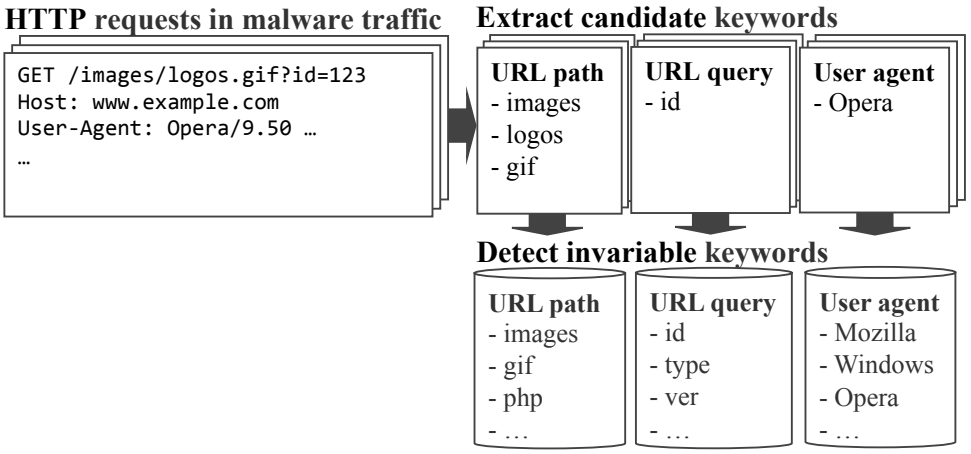


Figure 5.2: Example of variability profiling

reused based on the same malicious infrastructure.

It is claimed that ExecScent [13] observed the existence of stable URL paths in some malware-generated HTTP requests. However, it does not reveal how to detect and use them. On the other hand, we focused on invariable keywords not only in URL paths but also in URL queries, and user agents. Moreover, step 1 contributes to automatically detecting such invariable keywords to generate more accurate and valuable templates without using any particular ground truth data. The effectiveness of BotProfiler is validated later in Section 5.3.

5.2.3 Step 2: Template Generation

Replacing Substrings in HTTP Requests with Regular Expressions

Step 2 involves generating templates using invariable keywords produced in step 1. These templates contain features of URL paths, URL queries, and user agents, based on the result of clustering HTTP requests in malware traffic. Our templates include not only URLs but also user agents in HTTP request headers to reduce false positives with considering only URLs. To detect polymorphic patterns used by attackers, HTTP requests are segmented

Table 5.1: Example of patterns in regular expressions

Data type	Regular expression
String	`<str; length>`
Integer	`<int; length>`
Hexadecimal	`<hex; length>`
Base64	`<base64; length>`

into substrings with symbols (e.g., `/`, `?`, `=`, `-`, `.`) and replaced with regular expressions (e.g., `<str; length>`) containing the data type (e.g., string (`str`), integer (`int`), hexadecimal (`hex`), `base64`) and length indicated in Table 5.1. Figure 5.3 shows examples of how substrings are replaced with regular expressions using both ExecScent and BotProfiler. ExecScent basically replaces all substrings with regular expressions to effectively aggregate HTTP requests into templates. However, this system has a potential problem of falsely matching benign HTTP requests with templates, which results in false positives, due to an inherent aspect of regular expressions. For example, even if the structure of the URL path in a benign HTTP request (e.g., `/foobar/index.htm`) and that in malware traffic (e.g., `/images/logos.gif`) are totally different, the regular expressions that correspond to these structures are the same (e.g., `/<str;6>/<str;5>.<str;3>`). BotProfiler avoids such false positives based on regular expressions. Thus, it does *not* replace the substrings that match any invariable keywords produced in step 1 since these substrings tend to be reused by the same malicious infrastructure. The efficiency for aggregating HTTP requests into templates in ExecScent is higher than that of BotProfiler because more HTTP requests are converted into the same regular expression patterns. However, BotProfiler generates more specific templates since it uses the reused nature of malicious infrastructures.

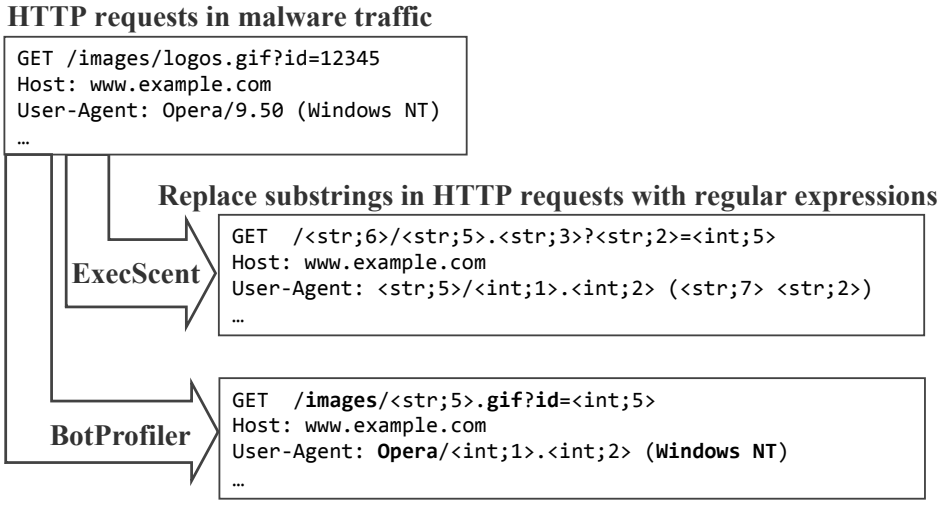


Figure 5.3: Example of replacing substrings in HTTP requests with regular expressions

Aggregating HTTP Requests

To reduce the number of templates and matching cost using the templates, ExecScent aggregates similar HTTP requests into one template. BotProfiler also uses this concept to generate practical templates. Specifically, it introduces two stages of clustering of HTTP requests with regular expressions. An overview of this is given in Fig. 5.4.

The first stage consists of clustering using the criterion of destination IP addresses. This clustering process groups HTTP requests that share the same destination IP address range or prefix to generate *IP range clusters*. The second stage consists of applying agglomerative hierarchical clustering to HTTP requests within each IP range cluster. The agglomerative hierarchical clustering sequentially combines similar requests based on the predefined similarity metric to output a dendrogram, which is a tree-like diagram representing the distance between clusters [102]. Cutting the dendrogram at a certain height, which is called the *cut height* and is determined empirically, divides the IP range cluster into *similar request clusters* that include mul-

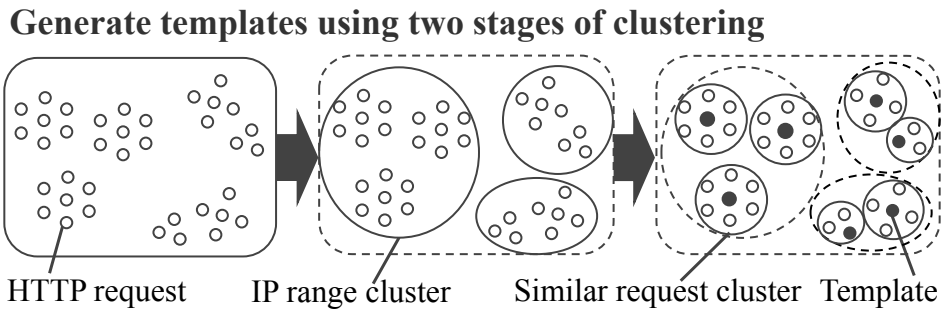


Figure 5.4: Overview of process to generate templates using clustering

tiple similar HTTP requests. BotProfiler finally selects 0.5 as the cut height based on the best result of preliminary experiments. In each similar request cluster, BotProfiler extracts the centroid, which refers to one of the HTTP requests that maximizes the sum of similarities between the request and all other requests. From the centroids, the *templates* that contain URL paths, URL queries, and user agents are extracted.

The similarity metric $Sim(h_a, h_b)$ between HTTP requests h_a and h_b is defined by the following equation.

$$Sim(h_a, h_b) = \frac{1}{n} \cdot \sum_{k=1}^{n} \sigma_k(h_a, h_b) \tag{5.1}$$

Here, σ_k is the function to calculate similarities between elements in h_a and h_b, and n is the number of considered elements; we set $n = 4$. Specifically, σ_1 is the similarity between URL paths and is calculated using the normalized edit distance, σ_2 is the similarity between the combination of parameter names in the URL queries and is calculated using the Jaccard similarity, σ_3 is the similarity between the values in the URL queries and is calculated using the ratio of having the same data type and length, and σ_4 is the similarity between user agents and is calculated using the normalized edit distance. The above definitions result in $\sigma_k \in [0, 1], \forall k$ and $Sim(h_a, h_b) \in [0, 1]$.

Note that the host header of HTTP requests, such as a domain name or an IP address, is not included as an element in the templates. Under

the assumption that attackers reuse their malicious infrastructure, such as web servers and toolkits, to conduct other attacks at a different domain name or IP address, a host header might be changed. For example, using a domain generation algorithm (DGA) enables attackers to change domain names frequently. However, elements in our templates (URL paths, URL queries, and user agents) are more stable than the host header. Thus, our templates can be used to detect attacks caused by the reuse of malicious infrastructure by attackers.

5.2.4 Step 3: Rarity Profiling

Step 3 contributes to reducing false positives when matching templates with traffic in a deployed network. This step involves calculating the rarities of elements in our templates generated in step 2. That is, this step involves finding the *rare* elements that appear very infrequently in benign traffic. Given that the number of infected hosts is far lower than that of non-infected hosts in a deployment network, the infrequent elements might be presumed to be sent from infected hosts in the network. Rarities of elements vary from network to network; thus, rarities should be calculated on each deployment network. We focused on the rarities of URL paths, URL queries, and user agents. ExecScent also uses a set of other headers in HTTP requests in addition to the above. However, we did not use it since such other headers are not available in the traffic capture environment in the deployment network. The rarity $\rho_{t,k}$ of an element k in a template t is calculated using the following equation.

$$\rho_{t,k} = 1 - \frac{n_{t,k}}{\max_i n_i} \tag{5.2}$$

Here, $n_{t,k}$ is the number of hosts that send HTTP requests containing k in t, and $\max_i n_i$ is the maximum number of hosts in all elements of the same type (e.g., URL paths, URL queries, and user agents). The definition results in $\rho_{t,k} \in [0, 1], \forall t, \forall k$.

5.2.5 Step 4: Template Matching

Step 4 involves matching traffic to be evaluated with both the templates generated in step 2 and the rarities in step 3 to detect malware-infected hosts. Specifically, a matching score $Score(h,t)$ between an HTTP request h and a template t is calculated from the following equation.

$$Score(h,t) = \frac{\sum_{k=1}^{n} \sigma_k(h,t) \cdot \omega(\sigma_k(h,t), \rho_{t,k})}{\sum_{k=1}^{n} \omega(\sigma_k(h,t), \rho_{t,k})} \cdot \rho_{h,d} \tag{5.3}$$

Here, $\sigma_k(h,t)$ is defined in the same way as explained in Section 5.2.3, $\rho_{t,k}$ is the rarity of k in t, $\rho_{h,d}$ is the rarity of a destination fully qualified domain name (FQDN) d in h and is calculated in the same way as step 3, and ω is the weight function between $\sigma_k(h,t)$ and $\rho_{t,k}$ and is defined by the following equation.

$$\omega(\sigma_k(h,t), \rho_{t,k}) = 1 + \frac{1}{(2 - \sigma_k(h,t) \cdot \rho_{t,k})^m} \tag{5.4}$$

The value m is a fixed parameter and determined empirically. The above definitions result in $Score(h,t) \in [0,1]$. A matching score $Score(h,t)$ is designed to be high when the similarity between an HTTP request h and a template t is high, and the rarities of elements in t are high in a deployment network. If $Score(h,t)$ exceeds a predefined threshold (matching score threshold), which means an HTTP request closely matches a template and the elements in the request have rarely appeared in the deployment network, BotProfiler determines h to be generated by an infected host.

5.3 Evaluation

5.3.1 Evaluation Overview

BotProfiler was evaluated using extensive actual datasets. This section explains how we evaluated it in terms of feasibility and detection performance in comparison to the conventional system ExecScent. The feasibility is based on the effectiveness of invariable keywords and the results of our variability profiling and template generation. Detection performance was measured by

Table 5.2: Malware traffic datasets

Dataset	# Malware samples	# Detected samples	# Malware families	# HTTP requests
Current (Aug. 2011–Dec. 2012)	2,507	431 (17%)	44	598,534
Future_1 (Jan. 2013)	426	366 (86%)	25	11,427
Future_2 (Feb. 2013)	444	396 (89%)	17	67,030
Future_3 (Mar. 2013)	451	282 (63%)	45	32,113
Future_4 (Apr. 2013)	511	301 (59%)	69	17,996
Future_5 (May. 2013)	616	376 (61%)	55	19,385
Future_6 (Jun. 2013)	438	344 (79%)	37	8,259
Future_7 (Jul. 2013)	695	465 (67%)	40	9,567
Future_8 (Aug. 2013)	1,477	932 (63%)	49	23,020

the detection rate of infected hosts and the false positive rate in the deployed network. Note that ExecScent is closed-source software; thus, we reimplement it based on the paper [13] to compare the detection performance in both systems.

5.3.2 Datasets

Malware traffic captured in the sandbox and benign traffic in the deployment network were used to evaluate the effectiveness of BotProfiler when deployed in a real network. We simulated the situation in which infected hosts exist in the deployment network. Our evaluation involved dividing both malware traffic and benign traffic based on the date of January 1, 2013. Specifically, malware and benign traffic *before* that date was used to generate templates or calculate rarities, whereas those kinds of traffic *after* that date were used to evaluate the detection rate or false positive rate. This situation is equivalent to evaluating the *future* detection performance of BotProfiler because it uses only the information as of January 1, 2013. The details of our datasets are described further below.

Malware traffic was captured from the sandbox system [100] running mal-

Table 5.3: Malware families in malware traffic dataset (Current)

Malware family	# Malware samples
`Backdoor.Win32.ZAccess`	157
`Trojan-Ransom.Win32.PornoAsset`	50
`Backdoor.Win32.PMax`	38
`Trojan-PSW.Win32.Tepfer`	31
`Trojan-Downloader.Win32.Agent`	25
`Trojan.Win32.Bublik`	16
`Trojan-Downloader.Win32.Andromeda`	12
`Trojan-Spy.Win32.Zbot`	12
`Trojan.Win32.FakeAV`	8
`Trojan-FakeAV.Win32.SmartFortress`	8

ware samples. The sandbox supports executable files only in Microsoft Windows environments. These malware samples were collected using the honeyclient [18, 101] crawling public blacklists such as MalwareDomainList [43] and hpHosts [57] and some commercial blacklists from August 2011 to August 2013. Table 5.2 lists the number of malware samples in each dataset, number and ratio of samples detected by antivirus software, number of unique malware family names, and number of HTTP requests. The *Current* dataset was used for generating templates and composed of HTTP requests generated by 2,507 unique malware samples. Datasets labeled *Future* were used to evaluate detection performance and consisted of datasets divided into months *Future_1–8*. Note that there were no overlaps in malware samples between the Current and Future_1–8 datasets.

All malware samples were checked with multiple antivirus software programs using the VirusTotal [60] as of January 6, 2015. Kaspersky was selected to label samples with malware family names for the following two reasons. One reason is that it achieved the best detection rate excepting the uninformative labels (e.g., `generic`, `heuristic`). The other reason is that the rule for family names is openly available [103]. The latest malware definition file of the software as of January 6, 2015 was used for the evaluation. Table 5.2

Table 5.4: Malware families in malware traffic datasets (Future_1–8)

Malware family	# Malware samples
AdWare.Win32.Agent	541
Trojan-PSW.Win32.Tepfer	272
Trojan-Ransom.Win32.Foreign	242
Backdoor.Win32.ZAccess	174
Trojan-Downloader.Win32.Agent	166
RiskTool.Win32.Agent	155
Downloader.Win32.LMN	132
Trojan-Ransom.Win32.PornoAsset	128
AdWare.NSIS.Indirect	127
AdWare.Win32.iBryte	100

indicates that the detection rate for antivirus software was not very good in any of the datasets even if the latest definition file was used. The reason is that most of the malware samples were not suitably collected or analyzed by the antivirus vendors. Tables 5.3 and 5.4 list the top 10 malware family names detected by the antivirus in both the Current and Future_1–8 datasets. This result indicates that the malware samples in our datasets were unbiased in terms of malware families. Moreover, the result shows the difference in distribution of families between Current and Future_1–8. Note that malware family information was used as reference in our evaluation and was *not* used in BotProfiler.

Benign traffic was captured in a large, real enterprise network from December 2012 to August 2013. The traffic was inspected by security engineers using commercial ground truth data to filter out the possibility of containing malicious traffic in January 2015. Table 5.5 shows the number of source IP addresses and number of outbound HTTP requests. The Training dataset was used to calculate rarities, and the Testing dataset was used to evaluate the false positive rate.

Table 5.5: Benign traffic datasets

Dataset	# Src IP addresses	# HTTP requests
Training (Dec. 2012)	5,261	95,438,564
Testing (Jan. 2013–Aug. 2013)	8,055	723,903,639

Table 5.6: Classification of URL path structure

URL path class	# Malware families	Effectiveness of BotProfiler
Fixed path structure	30 (68%)	Effective
Random path structure	14 (32%)	Ineffective

5.3.3 Verifying the Effectiveness of Invariable Keywords

In this section, we validate the effectiveness of invariable keywords used with BotProfiler. Specifically, URL paths, URL queries, and user agents in HTTP requests of the Current dataset were analyzed to reveal effective and ineffective cases when introducing invariable keywords to the templates. To the best of our knowledge, this is the first analysis focusing on the structure of malware-generated HTTP requests. The following results are the basis of the superiority of BotProfiler.

Table 5.6 lists the results of analyzing the structure of the *URL paths* in each malware family. The results indicate that 68% of malware families in the Current dataset send HTTP requests that have a fixed URL path structure, meaning that it contains fixed and fixed-length substrings. Invariable keywords in BotProfiler are effective for such fixed URL path structures. Two reasons can be considered for the large portion of fixed URL path structures. One is the use of publicly available APIs by malware samples. Some attackers use APIs (e.g., GeoIP) to obtain information on infected hosts such as IP addresses and locations. Using such APIs forces attackers to comply with the requirements of the APIs and to send HTTP requests that have a fixed URL path. The other reason is the high cost for attackers to accept

Table 5.7: Classification of URL query structure

URL query class	HTTP Method	# Malware families	Effectiveness of BotProfiler
Fixed field name	GET	14 (32%)	Effective
Random field name	GET	5 (11%)	Ineffective
No queries	GET	11 (25%)	-
No queries	POST	14 (32%)	-

random URL path structures. For example, attackers need to receive and interpret HTTP requests with such random URL paths on their C&C servers.

Table 5.7 lists the result of analyzing the structure of the *URL queries* in each malware family. The results indicate that 32% of malware families in the Current dataset send HTTP GET requests that have fixed URL query field names. Invariable keywords in BotProfiler are effective for such fixed URL query field names. The reasons for the fixed URL query field name are the same as those for having a fixed URL path, namely, the utilization of publicly available APIs and the high cost for attackers to accept a random URL query field name. In Table 5.7, the item listed as *no queries* in the HTTP GET method includes instances of checking the Internet connection using malware samples, and *no queries* in the HTTP POST method includes instances of sending information using the body of the request.

The results of analyzing the structure of *user agents* in each malware family are listed in Table 5.8. The results indicate that 77% of malware families in the Current dataset send HTTP requests that have the same user agent as general web browsers such as Internet Explorer and Firefox. User agents of general web browsers consist of fixed substrings, which represent the name of the browser or OS, and the version number (e.g., `Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)`). Therefore, using such fixed substrings as invariable keywords in BotProfiler is effective in generating accurate templates. The reason that most attackers use general web browsers as user agents is that they try to hide their C&C communications in legit-

Table 5.8: Classification of user agent structure

User agent class	# Malware families	Effectiveness of BotProfiler
Common web browsers	34 (77%)	Effective
Unusual user agents	5 (11%)	Effective
No user agents	5 (11%)	-

Table 5.9: Summary of variability profiling

	URL path	URL query	User agent
# Candidate keywords	6,521	1,365	601
# Invariable keywords	483	259	142

imate HTTP communications. On the other hand, 11% of malware families send HTTP requests with unusual user agents (e.g., `_Converter_agent`, `VBTagEdit`). However, most of these user agents include fixed substrings. Thus, using such fixed substrings as invariable keywords is also considered to be effective.

5.3.4 Results of Variability Profiling

This section explains the results of conducting variability profiling (step 1) with the Current dataset listed in Table 5.2. Specifically, we analyzed the relationship between candidate keywords in URL paths, URL queries, and user agents, and the number of malware samples that use each candidate keyword. The more malware samples that use a candidate keyword, the more likely it is that the candidate keyword is the invariable keyword that is based on the same malicious infrastructure. Figure 5.5 is the complementary cumulative distribution function (CCDF) showing the relationship between candidate keywords and the number of malware samples that use the keywords. In this case, CCDF corresponds to the probability of the candidate keywords being greater than or equal to the specified number of malware samples.

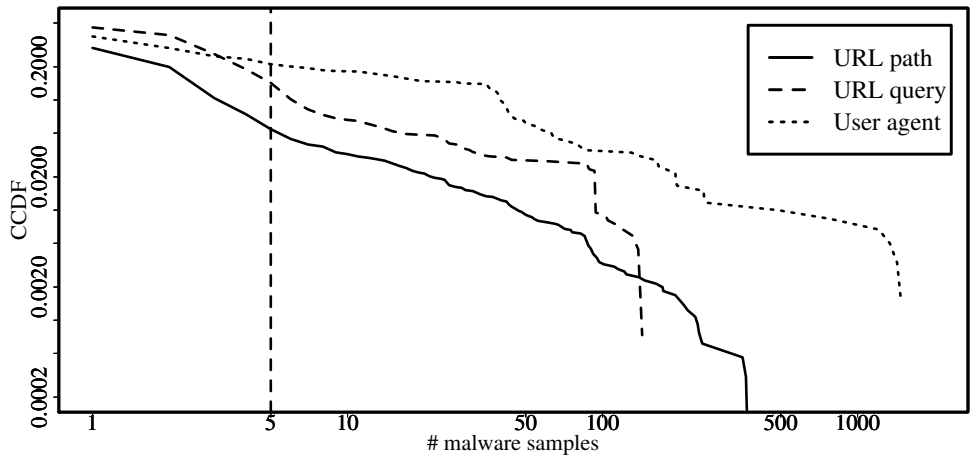


Figure 5.5: CCDF of number of malware samples for each keyword

Figure 5.5 reveals the existence of invariable keywords that are reused across multiple malware samples in the URL paths, URL queries, and user agents of HTTP requests generated by infected hosts. BotProfiler sets a threshold for the number of malware samples, and any candidate keyword that exceeds the threshold is identified as an invariable keyword. In our further evaluation, the threshold was set to five malware samples based on the best result in our preliminary verification. Table 5.9 lists the number of candidate keywords and that of invariable keywords in each URL path, URL query, and user agent. Moreover, Table 5.10 gives examples of invariable keywords identified with BotProfiler.

5.3.5 Results of Generating Templates

This section describes the results of generating templates (step 2) with the Current dataset and invariable keywords output from step 1. Table 5.11 lists the number of templates output from ExecScent, which does *not* use invariable keywords in templates, and that from BotProfiler, which uses invariable keywords. As explained in Section 5.2.3, the number of templates in BotProfiler exceeds that in ExecScent because the efficiency of aggre-

Table 5.10: Example of detected invariable keywords

Element	Examples of invariable keywords
URL path	`php, js, exe, txt, app, geo, geoid, images, up, stat, city, jpg, html, img, png, gif, install, htm, css, track`
URL query	`id, type, affid, ver, name, ts, event, short, currency, group, fail, ini, cmd, version, file, page, source, os, subid, step`
User agent	`Windows, Mozilla, NT, compatible, MSIE, SV, Opera, Agent, User, Presto, Gecko, Lang, ID, rv, Firefox, NET, CLR, JP, en, US`

Table 5.11: Summary of template generation

	# Input HTTP requests	# Output templates
ExecScent	598,534	1,749
BotProfiler	598,534	2,098

gating requests in BotProfiler is lower than that in ExecScent.

Examples of templates generated with BotProfiler are shown in Fig. 5.6. For example, Template #1 was created by keep-alive C&C communications used by the Sality botnet. Template #2 was produced by C&C communications that involved counting the number of infected hosts by the ZeroAccess botnet, which mainly uses P2P as a C&C protocol. However, BotProfiler succeeded in generating templates from a limited percentage of HTTP requests in ZeroAccess. Template #3 was generated by communications in which infected hosts are forced to download and install fake antivirus software. Note that these templates are automatically generated with BotProfiler *only* from HTTP requests in the Current dataset without using any other information such as malware family names obtained by antivirus software. With BotProfiler, generated templates are delivered to the deployment network and matched with traffic using pre-calculated rarities in

Template #1 (Generated from Trojan-Dropper.Win32.Agent)
[URL path] `/logos_<str;1>.gif`
[URL query] `<hex;6>=<int;7>`
[User agent] `Opera/<int;1>.<int;2> (Windows NT <int;1>.<int;1>; <str;1>; en)`

Template #2 (Generated from Backdoor.Win32.ZAccess)
[URL path] `/count.php`
[URL query] `id=<int;3>&<str;1>=<int;1>&<str;1>=<int;1>&<str;1>=<int;1>`
[User agent] `Opera/<int;1>.<int;2> (Windows NT <int;1>.<int;1>; JP; <base64;3>)`

Template #3 (Generated from Trojan:Win32:FakeAV)
[URL path] `/api/stats/install/`
[URL query] `ts=<hex;8>&affid=<int;5>&ver=<int;7>&group=<str;3>`
[User agent] `Mozilla/<int;1>.<int;1> (compatible; MSIE <int;1>.<int;1>; Windows NT)`

Figure 5.6: Examples of generated templates

the deployment network to reduce false positives.

The relationship between created templates and malware family names was analyzed, revealing that templates were generated from 39 out of 44 malware families in the Current dataset. There are two reasons that templates were not generated for five malware families. One reason is that some malware families do *not* use HTTP as their C&C protocol. BotProfiler is only focused on HTTP requests; therefore, such families were out of the scope of this study. However, this is not a major problem for us because our deployment network does not allow protocols other than web protocols (HTTP, HTTPS) using outbound traffic filtering. The other reason is the limitation of dynamic analysis in the sandbox. That is, malware samples that do not run or do not generate any HTTP requests in the sandbox are out of the scope of this study.

Table 5.12: Malware families in generated templates

Malware family name	# Templates
`Trojan-Downloader.Win32.Agent`	310
`Trojan-PSW.Win32.Tepfer`	193
`Trojan-Downloader.Win32.Andromeda`	64
`Trojan-Spy.Win32.Zbot`	30
`Backdoor.Win32.Hlux`	22
`Backdoor.Win32.Simda`	18
`Trojan.Win32.Jorik`	17
`Trojan.Win32.FakeAV`	9
`Trojan-FakeAV.Win32.FakeSysDef`	7
`Trojan.Win32.Genome`	7

Table 5.12 lists the top 10 malware family names detected by the antivirus software in the generated templates. A comparison between Table 5.3 and Table 5.12 reveals that the composition of malware families in the Current dataset and that in templates generated from the Current dataset differ widely. This is due to the characteristics of C&C communications. For example, the number of inputted ZeroAccess (`Backdoor.Win32.ZAccess`) malware samples was large; however, it mainly uses P2P as the C&C protocol and sends a few variations of HTTP requests. Thus, the number of generated templates is small.

5.3.6 Detection Rate

This section compares the detection rate in both ExecScent and BotProfiler. The detection rate is defined by the ratio of correctly detected infected hosts. For simplicity, an infected host is only infected with one malware sample at a time. That is, a malware sample in a dataset was considered to be detected if at least one of the HTTP requests in each malware sample was detected with BotProfiler. In our evaluation, we determined the detection rate of BotProfiler, which generates templates with invariable keywords output from step 1, and that of ExecScent, which does *not* use invariable

Table 5.13: Detection rate on malware traffic datasets

Matching score threshold		0.75	0.80	0.85	0.90	0.95
Future_1	ExecScent	91.08%	90.14%	88.97%	60.56%	60.33%
	BotProfiler	91.55%	91.08%	89.91%	69.95%	62.91%
Future_2	ExecScent	88.96%	88.96%	84.01%	79.28%	79.28%
	BotProfiler	89.64%	89.64%	84.68%	79.50%	79.28%
Future_3	ExecScent	68.74%	65.41%	52.55%	26.39%	26.39%
	BotProfiler	67.85%	63.41%	54.55%	27.49%	26.83%
Future_4	ExecScent	37.18%	28.57%	15.07%	2.94%	2.74%
	BotProfiler	35.81%	23.48%	14.68%	3.33%	3.13%
Future_5	ExecScent	32.31%	21.92%	11.53%	1.14%	0.81%
	BotProfiler	28.90%	18.34%	10.88%	1.14%	0.81%
Future_6	ExecScent	26.03%	14.38%	4.57%	0.23%	0.23%
	BotProfiler	22.37%	11.64%	6.39%	0.23%	0.23%
Future_7	ExecScent	23.31%	16.83%	4.46%	0.43%	0.43%
	BotProfiler	17.41%	9.78%	7.19%	0.43%	0.43%
Future_8	ExecScent	19.96%	14.83%	3.10%	0.27%	0.27%
	BotProfiler	14.16%	9.37%	7.35%	0.27%	0.27%

keywords.

Table 5.13 lists the results of detecting malware samples in *Future* datasets using templates generated by the *Current* dataset. This table shows detection rates with a variable matching score threshold. If $Score(h, t)$ exceeds the matching score threshold, BotProfiler determines the HTTP request h as being generated by an infected host, as described in Section 5.2.5. The Future datasets consisted of datasets divided by month *Future_1–8*. Each Future dataset contains malware samples collected after the Current dataset was compiled. Our evaluation using the Future datasets enabled us to track changes in the detection rate of templates from the Current dataset over time.

The results reveal three facts concerning detection rates in both systems. First, the detection rates of both systems decreased linearly over time (from Future_1 to Future_8) in each matching score threshold. This is due to matching Future datasets with the same templates generated from the Current

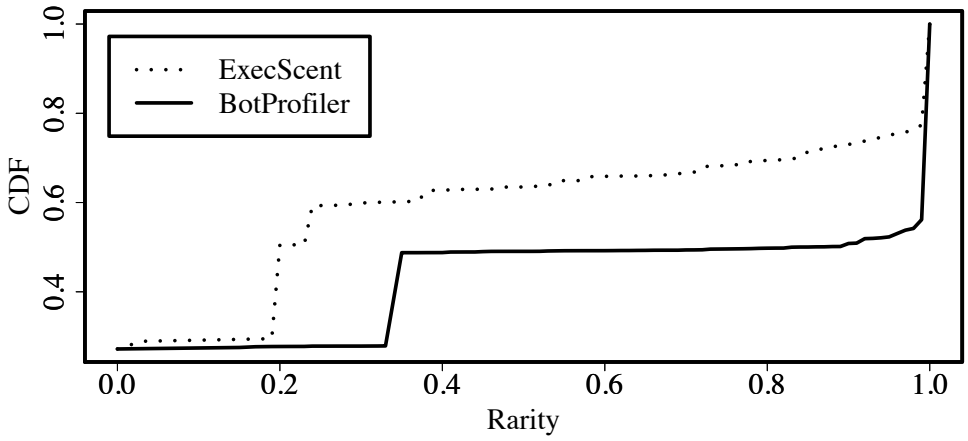


Figure 5.7: CDF of URL path rarities in malware traffic datasets

dataset, even if the malware samples in the Future datasets vary over time. This fact led us to conclude that templates should be updated using the latest malware samples on a regular basis. In this case, the templates should be updated after two months (Future_2) to obtain a detection rate of at least 80% at a matching score threshold of 0.85.

Second, the detection rates of BotProfiler tended to be lower than those of ExecScent as time advanced. For example, at a matching score threshold of 0.75, the detection rates in Future_1 and Future_2 were more than those of ExecScent. On the other hand, the detection rates between Future_3 and Future_8 were lower than those of ExecScent. These results indicate that the effective period of templates in ExecScent is longer than that of BotProfiler since the matching area covered by the regular expressions of ExecScent is wider than that of BotProfiler. However, the actual operation of BotProfiler in the deployment network includes downloading templates from the lab environment at least once every month. In such a case, the detection rate of BotProfiler is superior to that of ExecScent.

Finally, BotProfiler improved the detection rates with all thresholds between 0.75 and 0.95 in Future_1 and Future_2, even when it introduced invariable keywords to generate more specific templates. This is not a sur-

Table 5.14: False positive rate on benign traffic datasets

Matching score threshold	0.75	0.80	0.85	0.90	0.95
ExecScent	3.89%	0.46%	0.08%	0.04%	0.01%
BotProfiler	1.18%	0.20%	0.06%	0.03%	0.01%

prising result and is reasonable. The only reason is that BotProfiler determines not only the similarity to the templates but also the rarity of each element in the templates. For example, the rarity of the URL path in BotProfiler (e.g., `/images/<str;5>.gif`) is higher than that of ExecScent (e.g., `/<str;6>/<str;5>.<str;3>`) because the latter matches various types of HTTP requests in the deployment network. Figure 5.7 shows the CDF of URL path rarities in the deployment network when calculating a matching score of Future datasets. This graph illustrates that URL path rarities in BotProfiler are higher than those of ExecScent. This difference contributes to raising the matching scores of HTTP requests to improve the detection rates in BotProfiler.

5.3.7 False Positive Rate

This section compares the false positive rates for both ExecScent and BotProfiler. The false positive rate is defined by the ratio of falsely detected benign HTTP requests. Table 5.14 presents the false positive rates with variable matching score thresholds. The results indicate that the false positive rates of BotProfiler are always lower than those of ExecScent for all matching score thresholds between 0.75 and 0.95. In particular, at the threshold of 0.75, the false positive rate of BotProfiler (1.18%) was less than one-third that of ExecScent (3.89%). This is due to the effect of using invariable keywords in BotProfiler. Our invariable keywords enable BotProfiler to generate more accurate templates, which causes fewer false positives. We conclude that BotProfiler achieves a higher detection rate and lower false positive rate simultaneously under the condition in which

templates are regularly updated. In operating BotProfiler in the deployment network, a matching score threshold is set based on an acceptable false positive rate in the network. For example, setting the threshold at 0.85 enables BotProfiler to reduce the false positive rate to 0.06%.

5.4 Limitation

5.4.1 Detecting Invariable Keywords

Our variability profiling (step 1) heuristically set the threshold of the number of malware samples to five, based on the result of the preliminary verification. That threshold is used to identify invariable keywords; that is, any candidate keywords that exceed the threshold are identified as invariable keywords. Specifically, increasing the threshold reduces the number of invariable keywords and vise versa, as shown in Fig. 5.5. Reducing the number of invariable keywords causes an expansion of the matching area of regular expressions in our templates. This leads to an increase in both the detection rate and the false positive rate. Our future task is to automatically set the optimal threshold to meet the requirement for detection rate or false positive rate in each deployment network.

BotProfiler needs to update not only templates but also invariable keywords to catch up the latest trend of malware samples. The cost for updating invariable keywords is not so large and is small enough to update monthly at the same timing as updating templates. For example, in our preliminary experiment, it took only 18 seconds to create invariable keywords for *Current* dataset that contains 598,534 HTTP requests. Therefore, if we set the threshold preliminary, our system can update invariable keywords on a regular basis to support the latest attacks.

Dynamic analysis with code tainting can be used as another implementation of the concept of our variability profiling. Code tainting is an approach to track data propagation on a running system and is combined with dynamic analysis or a sandbox system [104]. Under the assumption that the

data propagation reveals the existence of fixed substrings in HTTP requests generated by malware samples, such fixed substrings can be utilized as invariable keywords. However, code tainting is a resource-hungry technique to run. Therefore, our lightweight approach, which is based on counting the number of malware samples, has a competitive advantage over code tainting.

5.4.2 Generating Templates from Malware Traffic

As shown in Fig. 5.1, BotProfiler generates templates from malware traffic. That is, to generate *ideal* templates, our lab environment needs to collect malware samples exhaustively and to analyze the collected samples adequately. These tasks may be problematic regarding cyber security for two reasons. One reason is that it is virtually impossible to collect all malware samples because attackers mass-produce their malware samples using a toolkit, and they have recently been targeting particular environments or organizations using drive-by downloads or advanced persistent threats (APTs). The other reason is that some malware samples can evade detection by identifying the sandbox environment. If malware samples include such a function, they cannot be dynamically analyzed using the sandbox. Kirat et al. recently proposed a malware analysis system using a *real* environment that does not include a monitoring component inside the system [105]. Such an analysis method may be a solution to analyze sophisticated malware samples.

5.4.3 Update of Rarities on Deployment Network

BotProfiler generates rarities based on one month of traffic in the deployment network to reduce false positives. However, the rarities should be updated with the appropriate timing. For example, the rarities of URL paths or URL queries might be dynamically changed when websites or web applications, which members in the deployment network usually use, are updated. Moreover, user agents might be changed if new software is introduced in the deployment network or if regularly used browsers are updated by their vendors. As stated above, the appropriate timing for updating the rarities

depends on the situation in the deployment network. Thus, our future task might include developing a method for calculating rarities in diverse or multiple deployment networks.

5.5 Related Work

5.5.1 Generating Network-based Templates

Much research has been done on generating network-based signatures or templates as a network-based countermeasure against infected hosts. Xie et al. proposed a system called AutoRE for generating signatures with regular expressions to detect polymorphic URLs in spam emails sent from botnets based on the nature of similar substrings in malicious URLs [98]. BotProfiler differs from AutoRE in that the focus is not spam emails but infected hosts, and the suffix-array algorithm proposed for AutoRE cannot appropriately be applied to substrings in HTTP requests. Perdisci et al. proposed a method of generating signatures with regular expressions to detect the URLs used in C&C communications based on HTTP traffic captured in a controlled environment [1]. BotProfiler targets not only URLs but also HTTP requests and determines not only similarities but also rarities to reduce false positives. Nelms et al. improved Perdisci et al.'s method [1] in their system called ExecScent, which covers HTTP requests [13]. ExecScent is one of the most advanced systems for generating templates for infected hosts using regular expressions and was a significant influence in developing BotProfiler. We expanded ExecScent to introduce the concept of invariability in substrings in HTTP requests. Zarras et al. proposed the BotHound system to focus on the sequence of components in HTTP headers to generate templates [106]. BotProfiler does not use the sequence of HTTP headers, that is, it is a more lightweight system than BotHound. Zand et al. proposed a method of generating signatures to detect C&C communications by focusing on frequent words in C&C communications [107]. The idea is similar to our invariable keywords, although this method cannot generate accurate templates com-

posed of URL paths, URL queries, and user agents and cannot determine both invariable and variable features in templates.

5.5.2 Modeling and Detecting Botnets

Another countermeasure against infected hosts includes modeling and detecting multiple infected hosts by using characteristics of botnets. Gu et al. proposed BotSniffer [108] and BotMiner [109] to detect simultaneous and similar network behaviors between multiple hosts in order to identify the activities of infected hosts based on the key idea that infected hosts of the same botnet have similar characteristics. Unlike with BotProfiler, controlling the false positive rate is generally difficult with these anomaly-based systems, and they are also difficult to deploy in a large and real network. Rossow et al. proposed a method to observe and model P2P botnets such as Zeus, ZeroAccess, and Kelihos [110]. Also, Zhang et al. proposed a method to detect such P2P botnet activities in a network [111]. BotProfiler does not focus on P2P botnets because our deployment networks basically only accept web protocols and do not accept P2P protocol. Caballero et al. proposed a method called protocol reverse engineering to analyze messages in unknown or undocumented protocols used in C&C communications [112]. BotProfiler only focuses on HTTP request headers, as HTTP is a known and documented protocol. Thus, our system does not need to use such techniques.

5.5.3 Detecting C&C Domain Names

The other approaches focus on the characteristics of domain names used in C&C communications to detect accesses to such domain names as infected hosts' activities. Holz et al. and Passerini et al. proposed methods to utilize features of Fast-Flux, which is a technique used by attackers to frequently change the mappings of their C&C domain names and a lot of IP addresses, and to identify C&C domain names [113, 114]. Also, Schiavoni et al. proposed a method focusing on the characteristics of DGAs, which are frequently

used in C&C to generate multiple domain names, and to detect such DGA domain names [89]. These methods focus exclusively on the relationship of domain names and differ from BotProfiler in the way that our templates only focus on URL paths, URL queries, and user agents. Therefore, these methods could be combined with our templates to detect infected hosts more accurately.

5.6 Conclusion

We proposed a system called BotProfiler to generate templates to detect infected hosts in a network. The key idea of our proposal is that malicious infrastructures such as malware and C&C tend to be reused instead of created from scratch. On the basis of this key idea, BotProfiler profiles invariable substrings in HTTP requests and generates more accurate templates than a conventional system. Our evaluation with large actual datasets revealed that BotProfiler reduced false positives by up to two thirds compared with the conventional system, and it even increased the detection rate of infected hosts. We also described a limitation of BotProfiler and the problems that remain to be solved regarding cyber security.

Chapter 6

Conclusion

This thesis focused on data analysis methods for network-based countermeasures against various types of cyberattacks. The goal of this thesis was to solve the four fundamental problems that hamper conventional countermeasures: malicious web content is dispersed by attackers, malicious domain names change over time, attackers generate differently-structured malicious domain names, and malicious HTTP communications blend in to evade detection. To achieve this goal, this thesis proposed four new analysis methods in the following chapters.

In Chapter 2, a new scheme to detect malicious websites by profiling their IP address features was developed and evaluated. The experimental results show that features extracted only from IP addresses are distinct indicators that enable us to compensate for the limitations of existing approaches; i.e., the scheme can detect even *unknown* malicious websites with a low error rate.

In Chapter 3, this thesis proposed a system called DomainProfiler to predict which domain names have the potential to be malicious in future. The key idea behind the system is to profile the temporal variation patterns (TVPs) of malicious domain names. The TVP of a domain name includes information about how and when the domain name has been listed in legitimate/popular and/or malicious domain name lists. The system actively collects DNS logs, identifies their TVPs, and predicts whether a given do-

main name will be used for a malicious purpose. Testing with large-scale data revealed that the system can predict which domain names will be malicious 220 days beforehand with a true positive rate (TPR) of 0.985.

In Chapter 4, this thesis designed and implemented a unified and objective analysis pipeline called DomainChroma to reveal *what, where,* and *how* countermeasures should be taken against malicious domain names for websites. DomainChroma applies malicious domain name *chromatography*, a new term we have defined to mean the separation of mixtures comprising various types of malicious domain names. Based on this idea and systematized knowledge, combined with state-of-the-art research, this thesis developed an analysis pipeline to offer practical and optimal defenses against today's malicious domain names without collateral damage to legitimate services. This thesis evaluated DomainChroma using a large, real dataset and showed that over 70% of domain names need only DNS-level defense and that there was no collateral damage to legitimate accesses.

In Chapter 5, this thesis proposed a system called BotProfiler, which generates templates for detecting infected hosts in a network. The key idea is that malicious infrastructure, such as malware samples and command and control (C&C) servers, tends to be reused instead of created from scratch. On the basis of this key idea, BotProfiler profiles fixed substrings in HTTP requests and generates more accurate templates than a conventional system would. The evaluation with large actual datasets revealed that BotProfiler reduced false positives by up to two-thirds compared to the conventional system, and it even increased the rate of detection of infected hosts.

As described above, this thesis proposed four new analysis methods to solve the four problems with conventional countermeasures. Test results from several different perspectives prove the effectiveness of the proposals in real-world settings. The knowledge and results presented in this thesis will help us to keep up with the trend of ever-changing cyberattacks and enhance the capabilities of cybersecurity systems on the Internet.

Acknowledgments

This doctoral thesis would not have been possible without the support of many people around me. I would like to take this opportunity to acknowledge those who have helped me with my research activities over the years.

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Shigeki Goto, for his kind and consistent support during my undergraduate, master's, and doctoral courses at Goto Laboratory in Waseda University. He provided me with his insightful knowledge and guided me to complete this thesis.

I would also like to thank my sub-advisor, Prof. Tatsuya Mori, for his invaluable support. He was a researcher at NTT before he moved to Waseda University. At that time, we conducted joint research on network security. His invaluable and practical advice has always helped me improve the quality of my research and papers.

In addition, I would like to thank Prof. Masato Uchida for undertaking to referee my doctoral thesis. His valuable comments, based on his expertise, enabled me to improve the quality of this thesis.

I truly feel grateful to all members of Prof. Goto's laboratory or *Team Goto Love*. In particular, working with Dr. Akihiro Shimoda, Mr. Kazuhiro Tobe and Mr. Yuta Takata was a great opportunity and helped me to become a real security researcher.

In addition, I would like to extend my gratitude to all my colleagues at NTT. In particular, I was fortunate enough to work with Dr. Takeshi Yagi and Dr. Mitsuaki Akiyama. Dr. Yagi was my mentor during my summer internship at NTT while I was a master's student, and he became my mentor

again after I joined NTT. His tireless effort, guidance, and encouragement helped me to grow as a researcher in this field. Dr. Akiyama helped me learn how to think like a security researcher and explore new research projects through daily discussions with him. I would also like to thank Dr. Keisuke Ishibashi, Mr. Kazufumi Aoki, Mr. Yuta Takata, and Mr. Toshiki Shibahara for their valuable comments, based on their deep expertise. Mr. Makoto Otsuka and Mr. Nobuharu Nitta helped me to implement and operate my proposed systems. In addition, I would like to thank my supervisors, Mr. Takeo Hariu and Mr. Takeshi Yada, for encouraging my research activities and giving me the chance to earn a doctoral degree while working at NTT.

Finally, special thanks must go to my family. My parents have always encouraged and supported me. My wife Yuki and my daughter Emily are my biggest sources of encouragement.

Bibliography

[1] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA, pp.391–404, USENIX Association, 2010.

[2] M.A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos, "CAMP: content-agnostic malware protection," 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, The Internet Society, 2013.

[3] L. Invernizzi, S. Miskovic, R. Torres, C. Kruegel, S. Saha, G. Vigna, S. Lee, and M. Mellia, "Nazca: Detecting malware distribution in large-scale networks," 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014, The Internet Society, 2014.

[4] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings, pp.273–290, USENIX Association, 2010.

[5] M. Kührer, C. Rossow, and T. Holz, "Paint it black: Evaluating the effectiveness of malware blacklists," Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings, ed. A. Stavrou, H. Bos,

and G. Portokalidis, Lecture Notes in Computer Science, vol.8688, pp.1–21, Springer, 2014.

[6] B. Rahbarinia, R. Perdisci, and M. Antonakakis, "Segugio: Efficient behavior-based tracking of malware-control domains in large ISP networks," 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015, pp.403–414, IEEE Computer Society, 2015.

[7] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: finding malicious domains using passive DNS analysis," Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011, The Internet Society, 2011.

[8] M. Akiyama, T. Yagi, and M. Itoh, "Searching structural neighborhood of malicious urls to improve blacklisting," 11th Annual International Symposium on Applications and the Internet, SAINT 2011, Munich, Germany, 18-21 July, 2011, Proceedings, pp.1–10, IEEE Computer Society, 2011.

[9] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: a fast filter for the large-scale detection of malicious web pages," Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011, ed. S. Srinivasan, K. Ramamritham, A. Kumar, M.P. Ravindra, E. Bertino, and R. Kumar, pp.197–206, ACM, 2011.

[10] L. Invernizzi and P.M. Comparetti, "Evilseed: A guided approach to finding malicious web pages," IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA, pp.428–442, IEEE Computer Society, 2012.

[11] C. Grier, L. Ballard, J. Caballero, N. Chachra, C.J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis,

N. Provos, M.Z. Rafique, M.A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G.M. Voelker, "Manufacturing compromise: the emergence of exploit-as-a-service," the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012, ed. T. Yu, G. Danezis, and V.D. Gligor, pp.821–832, ACM, 2012.

[12] M.Z. Rafique and J. Caballero, "FIRMA: malware clustering and network signature generation with mixed network behaviors," Research in Attacks, Intrusions, and Defenses - 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings, ed. S.J. Stolfo, A. Stavrou, and C.V. Wright, Lecture Notes in Computer Science, vol.8145, pp.144–163, Springer, 2013.

[13] T. Nelms, R. Perdisci, and M. Ahamad, "Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates," Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013, ed. S.T. King, pp.589–604, USENIX Association, 2013.

[14] G.D. Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Pexy: The other side of exploit kits," Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings, ed. S. Dietrich, Lecture Notes in Computer Science, vol.8550, pp.132–151, Springer, 2014.

[15] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna, "The dark alleys of madison avenue: Understanding malicious advertisements," Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014, ed. C. Williamson, A. Akella, and N. Taft, pp.373–380, ACM, 2014.

[16] T. Taylor, K.Z. Snow, N. Otterness, and F. Monrose, "Cache, trigger, impersonate: Enabling context-sensitive honeyclient analysis on-the-wire," 23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016, The Internet Society, 2016.

[17] M.A. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt, "Trends in circumventing web-malware detection," tech. rep., Google, 2011.

[18] M. Akiyama, M. Iwamura, Y. Kawakoya, K. Aoki, and M. Itoh, "Design and implementation of high interaction client honeypot for drive-by-download attacks," IEICE Transactions, vol.93-B, no.5, pp.1131–1139, 2010.

[19] "How to avoid, remove Facebook malware." `https://www.cnet.com/how-to/how-to-avoid-remove-facebook-malware/`.

[20] J. Levine, "DNS Blacklists and Whitelists." RFC 5782 (Informational), Feb. 2010.

[21] "URIBL." `http://www.uribl.com/`.

[22] "OpenDNS." `http://www.opendns.com/`.

[23] "SmartScreen Filter: FAQ." `https://support.microsoft.com/en-us/help/17443/windows-internet-explorer-smartscreen-filter-faq`.

[24] "Google Safe Browsing." `https://developers.google.com/safe-browsing/`.

[25] M. Félegyházi, C. Kreibich, and V. Paxson, "On the potential of proactive domain blacklisting," 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '10, San Jose, CA, USA, April 27, 2010, ed. M. Bailey, USENIX Association, 2010.

[26] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009, ed. J.F.E. IV, F. Fogelman-Soulié, P.A. Flach, and M.J. Zaki, pp.1245–1254, ACM, 2009.

[27] S. Yadav, A.K.K. Reddy, A.L.N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010, ed. M. Allman, pp.48–61, ACM, 2010.

[28] K. Sato, K. Ishibashi, T. Toyono, and N. Miyake, "Extending black domain name list by using co-occurrence relation between DNS queries," 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '10, San Jose, CA, USA, April 27, 2010, ed. M. Bailey, USENIX Association, 2010.

[29] C. Curtsinger, B. Livshits, B.G. Zorn, and C. Seifert, "ZOZZLE: fast and precise in-browser javascript malware detection," 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings, USENIX Association, 2011.

[30] S. Shin and G. Gu, "Conficker and beyond: a large-scale empirical study," Twenty-Sixth Annual Computer Security Applications Conference, ACSAC 2010, Austin, Texas, USA, 6-10 December 2010, ed. C. Gates, M. Franz, and J.P. McDermott, pp.151–160, ACM, 2010.

[31] V. Paxson, "Bro: A system for detecting network intruders in real-time," Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998, ed. A.D. Rubin, USENIX Association, 1998.

[32] M. Roesch, "Snort: Lightweight intrusion detection for networks," Proceedings of the 13th Conference on Systems Administration (LISA-99), Seattle, WA, November 7-12, 1999, ed. D.W. Parter, pp.229–238, USENIX, 1999.

[33] M. Ishida, H. Takakura, and Y. Okabe, "High-performance intrusion detection using optigrid clustering and grid-based labelling," 11th Annual International Symposium on Applications and the Internet, SAINT 2011, Munich, Germany, 18-21 July, 2011, Proceedings, pp.11–19, IEEE Computer Society, 2011.

[34] C. Seifert, I. Welch, and P. Komisarczuk, "HoneyC - the low-interaction client honeypot," 2006.

[35] "Capture-HPC." `https://projects.honeynet.org/capture-hpc/`.

[36] L. Lu, V. Yegneswaran, P.A. Porras, and W. Lee, "BLADE: an attack-agnostic approach for preventing drive-by malware infections," Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010, ed. E. Al-Shaer, A.D. Keromytis, and V. Shmatikov, pp.440–450, ACM, 2010.

[37] M. Akiyama, Y. Kawakoya, and T. Hariu, "Scalable and performance-efficient client honeypot on high interaction system," 12th IEEE/IPSJ International Symposium on Applications and the Internet, SAINT 2012, Izmir, Turkey, July 16-20, 2012, pp.40–50, IEEE Computer Society, 2012.

[38] Y. Niu, H. Chen, F. Hsu, Y. Wang, and M. Ma, "A quantitative study of forum spamming using context-based analysis," Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA, 28th February - 2nd March 2007, The Internet Society, 2007.

[39] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pisa, Italy, September 11-15, 2006, ed. L. Rizzo, T.E. Anderson, and N. McKeown, pp.291–302, ACM, 2006.

[40] S. Hao, N.A. Syed, N. Feamster, A.G. Gray, and S. Krasser, "Detecting spammers with SNARE: spatio-temporal network-level automatic reputation engine," 18th USENIX Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings, ed. F. Monrose, pp.101–118, USENIX Association, 2009.

[41] M.P. Collins, T.J. Shimeall, S. Faber, J. Janies, R. Weaver, M.D. Shon, and J.B. Kadane, "Using uncleanliness to predict future botnet addresses," Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007, ed. C. Dovrolis and M. Roughan, pp.93–104, ACM, 2007.

[42] "Alexa Top Sites." `http://www.alexa.com/topsites/`.

[43] "Malware Domain List." `http://malwaredomainlist.com/`.

[44] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures," Theor. Comput. Sci., vol.181, no.1, pp.3–15, 1997.

[45] X. Cai and J.S. Heidemann, "Understanding block-level address usage in the visible internet," Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010, ed. S. Kalyanaraman, V.N. Padmanabhan, K.K. Ramakrishnan, R. Shorey, and G.M. Voelker, pp.99–110, ACM, 2010.

[46] C. Bishop, Pattern recognition and machine learning, Springer, New York, 2006.

[47] C. Chang and C. Lin, "LIBSVM: A library for support vector machines," ACM TIST, vol.2, no.3, p.27, 2011.

[48] J.C. Platt, "Fast training of support vector machines using sequential minimal optimization," Advances in Kernel Methods - Support Vector Learning, MIT Press, January 1998.

[49] J.C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," ADVANCES IN LARGE MARGIN CLASSIFIERS, pp.61–74, MIT Press, 1999.

[50] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel, "Internet Registry IP Allocation Guidelines." RFC 2050 (Historic), Nov. 1996. Obsoleted by RFC 7020.

[51] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[52] M. Hatada, Y. Nakatsuru, M. Akiyama, and S. Miwa, "Datasets for anti-malware research MWS 2010 datasets." IPSJ Computer Security Symposium (CSS) 2010 (in Japanese), 2010.

[53] M. Hatada, Y. Nakatsuru, and M. Akiyama, "Datasets for anti-malware research MWS 2011 datasets." IPSJ Computer Security Symposium (CSS) 2011 (in Japanese), 2011.

[54] "Anti malware engineering workshop 2012 (MWS 2012)." `http://www.iwsec.org/mws/2012/en.html`.

[55] "rblcheck." `http://sourceforge.net/projects/rblcheck/`.

[56] Verisign, "Domain name industry brief." http://www.verisign.com/en_US/innovation/dnib/index.xhtml.

[57] "hpHosts." http://www.hosts-file.net/.

[58] F. Weimer, "Passive DNS replication," Proceedings of the 17th Annual FIRST Conference, 2005.

[59] Y. Chen, M. Antonakakis, R. Perdisci, Y. Nadji, D. Dagon, and W. Lee, "DNS noise: Measuring the pervasiveness of disposable domains in modern DNS traffic," 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014, pp.598–609, IEEE Computer Society, 2014.

[60] "VirusTotal." https://www.virustotal.com/.

[61] Mozilla foundation, "Public suffix list." https://publicsuffix.org/list/.

[62] CAIDA, "Routeviews prefix to AS mappings dataset (pfx2as) for IPv4 and IPv6." http://www.caida.org/data/routing/routeviews-prefix2as.xml.

[63] MaxMind, "GeoIP2 databases." https://www.maxmind.com/en/geoip2-databases.

[64] P. Maigron, "Regional Internet registries statistics." http://www-public.tem-tsp.eu/~maigron/RIR_Stats/index.html.

[65] L. Breiman, "Random forests," Machine Learning, vol.45, no.1, pp.5–32, 2001.

[66] P. Mockapetris, "Domain names - implementation and specification." RFC 1035 (INTERNET STANDARD), Nov. 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766.

[67] S. Yadav, A.K.K. Reddy, A.L.N. Reddy, and S. Ranjan, "Detecting algorithmically generated domain-flux attacks with DNS traffic analysis," IEEE/ACM Trans. Netw., vol.20, no.5, pp.1663–1677, 2012.

[68] J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Félegyházi, and C. Kanich, "The long "taile" of typosquatting domain names," Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014., ed. K. Fu and J. Jung, pp.191–206, USENIX Association, 2014.

[69] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A passive DNS analysis service to detect and report malicious domains," ACM Trans. Inf. Syst. Secur., vol.16, no.4, pp.14:1–14:28, 2014.

[70] R. Perdisci, I. Corona, and G. Giacinto, "Early detection of malicious flux networks via large-scale passive DNS traffic analysis," IEEE Trans. Dependable Sec. Comput., vol.9, no.5, pp.714–726, 2012.

[71] M. Antonakakis, R. Perdisci, W. Lee, N.V. II, and D. Dagon, "Detecting malware domains at the upper DNS hierarchy," 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings, USENIX Association, 2011.

[72] M. Antonakakis, R. Perdisci, Y. Nadji, N.V. II, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of dga-based malware," Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012, ed. T. Kohno, pp.491–506, USENIX Association, 2012.

[73] M. Thomas and A. Mohaisen, "Kindred domains: detecting and clustering botnet domains using DNS traffic," 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume, ed. C. Chung, A.Z. Broder, K. Shim, and T. Suel, pp.707–712, ACM, 2014.

[74] P.K. Manadhata, S. Yadav, P. Rao, and W. Horne, "Detecting malicious domains via graph inference," Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I, ed. M. Kutylowski and J. Vaidya, Lecture Notes in Computer Science, vol.8712, pp.1–18, Springer, 2014.

[75] A. Boukhtouta, D. Mouheb, M. Debbabi, O. Alfandi, F. Iqbal, and M.E. Barachi, "Graph-theoretic characterization of cyber-threat infrastructures," Digital Investigation, vol.14, no.1, pp.S3–S15, 2015.

[76] D. Chiba, K. Tobe, T. Mori, and S. Goto, "Detecting malicious websites by learning IP address features," 12th IEEE/IPSJ International Symposium on Applications and the Internet, SAINT 2012, Izmir, Turkey, July 16-20, 2012, pp.29–39, IEEE Computer Society, 2012.

[77] S. Venkataraman, D. Brumley, S. Sen, and O. Spatscheck, "Automatically inferring the evolution of malicious activity on the internet," 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013, The Internet Society, 2013.

[78] K. Soska and N. Christin, "Automatically detecting vulnerable websites before they turn malicious," Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014., ed. K. Fu and J. Jung, pp.625–640, USENIX Association, 2014.

[79] C. Lever, R.J. Walls, Y. Nadji, D. Dagon, P.D. McDaniel, and M. Antonakakis, "Domain-z: 28 registrations later measuring the exploitation of residual trust in domains," IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pp.691–706, IEEE Computer Society, 2016.

[80] D. Chiba, T. Yagi, M. Akiyama, T. Shibahara, T. Yada, T. Mori, and S. Goto, "Domainprofiler: Discovering domain names abused in fu-

ture," 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016, pp.491–502, IEEE Computer Society, 2016.

[81] S. Hao, A. Kantchelian, B. Miller, V. Paxson, and N. Feamster, "PREDATOR: proactive recognition and elimination of domain abuse at time-of-registration," Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, ed. E.R. Weippl, S. Katzenbeisser, C. Kruegel, A.C. Myers, and S. Halevi, pp.1568–1579, ACM, 2016.

[82] ICANN, "ICANN new gTLDs delegated strings." `https://newgtlds.icann.org/en/program-status/delegated-strings`.

[83] T. Halvorson, M.F. Der, I.D. Foster, S. Savage, L.K. Saul, and G.M. Voelker, "From .academy to .zone: An analysis of the new TLD land rush," Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015, ed. K. Cho, K. Fukuda, V.S. Pai, and N. Spring, pp.381–394, ACM, 2015.

[84] F. Li, G. Ho, E. Kuan, Y. Niu, L. Ballard, K. Thomas, E. Bursztein, and V. Paxson, "Remedying web hijacking: Notification effectiveness and webmaster comprehension," Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016, pp.1009–1019, 2016.

[85] "Verisign anti-abuse domain use policy." `https://www.icann.org/en/system/files/files/verisign-com-net-name-request-10oct11-en.pdf`.

[86] M.A. Bashir, S. Arshad, W.K. Robertson, and C. Wilson, "Tracing information flows between ad exchanges using retargeted ads," 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., ed. T. Holz and S. Savage, pp.481–496, USENIX Association, 2016.

[87] J. Chen, X. Zheng, H. Duan, J. Liang, J. Jiang, K. Li, T. Wan, and V. Paxson, "Forwarding-loop attacks in content delivery networks," 23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016, The Internet Society, 2016.

[88] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A comprehensive measurement study of domain generating malware," 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., ed. T. Holz and S. Savage, pp.263–278, USENIX Association, 2016.

[89] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: Dga-based botnet tracking and intelligence," Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings, ed. S. Dietrich, Lecture Notes in Computer Science, vol.8550, pp.192–211, Springer, 2014.

[90] T. Vissers, W. Joosen, and N. Nikiforakis, "Parking sensors: Analyzing and detecting parked domains," 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015, The Internet Society, 2015.

[91] Z. Li, S.A. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures," 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, pp.112–126, IEEE Computer Society, 2013.

[92] Farsight Security, Inc., "DNSDB." `https://www.dnsdb.info/`.

[93] R. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and taxonomy of botnet research through life-cycle," ACM Comput. Surv., vol.45, no.4, p.45, 2013.

[94] Intel Security, "Periodic connections to control server offer new way to detect botnets." https://securingtomorrow.mcafee.com/mcafee-labs/periodic-links-to-control-server-offer-new-way-to-detect-botnets/.

[95] Microsoft, "Microsoft and financial services industry leaders target cybercriminal operations from Zeus botnets." http://blogs.microsoft.com/blog/2012/03/25/microsoft-and-financial-services-industry-leaders-target-cybercriminal-operations-from-zeus-botnets/.

[96] Microsoft, "Microsoft works with financial services industry leaders, law enforcement and others to disrupt massive financial cybercrime ring." https://blogs.microsoft.com/blog/2013/06/05/microsoft-works-with-financial-services-industry-leaders-law-enforcement-and-others-to-disrupt-massive-financial-cybercrime-ring/.

[97] Microsoft, "Microsoft helps FBI in GameOver Zeus botnet cleanup." http://blogs.microsoft.com/blog/2014/06/02/microsoft-helps-fbi-in-gameover-zeus-botnet-cleanup/.

[98] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: signatures and characteristics," Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008, ed. V. Bahl, D. Wetherall, S. Savage, and I. Stoica, pp.171–182, ACM, 2008.

[99] Ponemon Institute, "The cost of malware containment." http://www.ponemon.org/library/the-cost-of-malware-containment.

[100] K. Aoki, T. Yagi, M. Iwamura, and M. Itoh, "Controlling malware HTTP communications in dynamic analysis system using search engine," CSS, pp.1–6, 2011.

[101] M. Akiyama, T. Yagi, Y. Kadobayashi, T. Hariu, and S. Yamaguchi, "Client honeypot multiplication with high performance and precise detection," IEICE Transactions, vol.98-D, no.4, pp.775–787, 2015.

[102] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data clustering: A review," ACM Comput. Surv., vol.31, no.3, pp.264–323, 1999.

[103] Securelist, "Rules for naming." `https://securelist.com/threats/rules-for-naming/`.

[104] G. Jacob, R. Hund, C. Kruegel, and T. Holz, "JACKSTRAWS: picking command and control connections from bot traffic," 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings, USENIX Association, 2011.

[105] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: Bare-metal analysis-based evasive malware detection," Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014., ed. K. Fu and J. Jung, pp.287–301, USENIX Association, 2014.

[106] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of http-based malware," 2014 Twelfth Annual International Conference on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24, 2014, ed. A. Miri, U. Hengartner, N. Huang, A. Jøsang, and J. García-Alfaro, pp.249–256, IEEE, 2014.

[107] A. Zand, G. Vigna, X. Yan, and C. Kruegel, "Extracting probable command and control signatures for detecting botnets," Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014, ed. Y. Cho, S.Y. Shin, S. Kim, C. Hung, and J. Hong, pp.1657–1662, ACM, 2014.

[108] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," Proceedings of the Network

and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008, The Internet Society, 2008.

[109] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA, ed. P.C. van Oorschot, pp.139–154, USENIX Association, 2008.

[110] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C.J. Dietrich, and H. Bos, "Sok: P2PWNED - modeling and evaluating the resilience of peer-to-peer botnets," 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013, pp.97–111, IEEE Computer Society, 2013.

[111] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, "Building a scalable system for stealthy p2p-botnet detection," IEEE Trans. Information Forensics and Security, vol.9, no.1, pp.27–38, 2014.

[112] J. Caballero, P. Poosankam, C. Kreibich, and D.X. Song, "Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering," Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009, ed. E. Al-Shaer, S. Jha, and A.D. Keromytis, pp.621–634, ACM, 2009.

[113] T. Holz, C. Gorecki, K. Rieck, and F.C. Freiling, "Measuring and detecting fast-flux service networks," Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008, The Internet Society, 2008.

[114] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi, "Fluxor: Detecting and monitoring fast-flux service networks," Detection of Intru-

sions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings, ed. D. Zamboni, Lecture Notes in Computer Science, vol.5137, pp.186–206, Springer, 2008.

List of Research Achievements

Journal Papers

[A-1] Daiki Chiba, Takeshi Yagi, Mitsuaki Akiyama, Kazufumi Aoki, Takeo Hariu, and Shigeki Goto, "BotProfiler: Detecting Malware-Infected Hosts by Profiling Variability of Malicious Infrastructure," IEICE Transactions on Communications, vol.E99.B, no.5, pp.1012–1023, May 2016.
DOI:10.1587/transcom.2015AMP0001
（電子情報通信学会 通信ソサイエティ論文賞 Best Paper Award 受賞）

[A-2] Daiki Chiba, Kazuhiro Tobe, Tatsuya Mori, and Shigeki Goto, "Analyzing Spatial Structure of IP Addresses for Detecting Malicious Websites," Journal of Information Processing (JIP), vol.21, no.3, pp.539–550, July 2013.
DOI:10.2197/ipsjjip.21.539

Conference Papers

[B-1] Daiki Chiba, Mitsuaki Akiyama, Takeshi Yagi, Takeshi Yada, Tatsuya Mori, and Shigeki Goto, "DomainChroma: Providing Optimal Countermeasures against Malicious Domain Names," Proceedings of the 41st Annual IEEE Computer Software and Applications Conference (COMPSAC), Turin, Italy, July 2017. (in printing)

[B-2] Daiki Chiba, Takeshi Yagi, Mitsuaki Akiyama, Toshiki Shibahara, Takeshi Yada, Tatsuya Mori, and Shigeki Goto, "DomainProfiler: Discovering Domain Names Abused in Future," Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp.491–502, Toulouse, France, June 2016. DOI:10.1109/DSN.2016.51

[B-3] Daiki Chiba, Takeshi Yagi, Mitsuaki Akiyama, Kazufumi Aoki, Takeo Hariu, and Shigeki Goto, "BotProfiler: Profiling Variability of Substrings in HTTP Requests to Detect Malware-Infected Hosts," Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp.758–765, Helsinki, Finland, August 2015. DOI:10.1109/Trustcom.2015.444

[B-4] Daiki Chiba, Kazuhiro Tobe, Tatsuya Mori, and Shigeki Goto, "Detecting Malicious Websites by Learning IP Address Features," Proceedings of the 12th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), pp.29–39, Izmir, Turkey, July 2012.
DOI:10.1109/SAINT.2012.14

[B-5] Toshiki Shibahara, Kohei Yamanishi, Yuta Takata, Daiki Chiba, Mitsuaki Akiyama, Takeshi Yagi, Yuichi Ohsita, and Masayuki Murata, "Malicious URL Sequence Detection using Event De-noising Convolutional Neural Network," Proceedings of the IEEE International Conference on Communications (ICC), Paris, France, May 2017. (in printing)

[B-6] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada, "Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning," Proceedings of the IEEE Global Communications Conference (GLOBECOM), pp.1–7, Washington, D.C., USA, December 2016. DOI:10.1109/GLOCOM.2016.7841778

[B-7] Naomi Kuze, Shu Ishikura, Takeshi Yagi, Daiki Chiba, and Masayuki Murata, "Detection of Vulnerability Scanning Using Features of Collective Accesses Based on Information Collected from Multiple Honeypots," Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), pp.1067–1072, Istanbul, Turkey, April 2016.
DOI:10.1109/NOMS.2016.7502962

[B-8] Naomi Kuze, Shu Ishikura, Takeshi Yagi, Daiki Chiba, and Masayuki Murata, "Crawler Classification using Ant-based Clustering Scheme," Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST), pp.84–89, London, UK, December 2015.
DOI:10.1109/ICITST.2015.7412063

[B-9] Ryo Sato, Daiki Chiba, and Shigeki Goto, "Detecting Android Malware by Analyzing Manifest Files," Proceedings of the Asia Pacific Advanced Network (APAN), vol.36, pp.23–31, Daejeon, Korea, August 2013.
DOI:10.7125/APAN.36.4

Book

[C-1] 八木毅, 青木一史, 秋山満昭, 幾世知範, 高田雄太, 千葉大紀, "実践サイバーセキュリティモニタリング," コロナ社, 2016.

Others

[D-1] 千葉大紀, 八木毅, 秋山満昭, 森達哉, 矢田健, 針生剛男, 後藤滋樹, "攻撃インフラの時系列変動特性に基づく悪性ドメイン名の検知法," 電子情報通信学会 技術研究報告（信学技報）, vol.115, no.81, ICSS2015-10, pp.51–56, 2015年6月.（電子情報通信学会 情報通信システムセキュリティ研究賞 受賞）

[D-2] 千葉大紀, 八木毅, 秋山満昭, 青木一史, 針生剛男, “感染後通信検知のための通信プロファイリング技術の設計と評価,” 情報処理学会 コンピュータセキュリティシンポジウム（CSS）2014 論文集, vol.2014, no.2, pp.960–967, 2014 年 10 月.

[D-3] Daiki Chiba, Takeshi Yagi, Mitsuaki Akiyama, Kazufumi Aoki, and Takeo Hariu, “Profiling HTTP Requests to Detect Malware-infected Hosts,” USENIX Security Symposium Poster Session, August 2014.

[D-4] 千葉大紀, 中田健介, 秋山満昭, 青木一史, 神谷和憲, 八木毅, “マルウェア感染端末検知のための HTTP 通信プロファイル技術の設計,” 電子情報通信学会 技術研究報告（信学技報）, vol.113, no.502, ICSS2013-84, pp.155–160, 2014 年 3 月.

[D-5] 千葉大紀, 森達哉, 後藤滋樹, “悪性 Web サイト探索のための優先巡回順序の選定法,” 情報処理学会 コンピュータセキュリティシンポジウム（CSS）2012 論文集, vol.2012, no.3, pp.805–812, 2012 年 10 月.

[D-6] 千葉大紀, 八木毅, 秋山満昭, 森達哉, 後藤滋樹, “多種多様な攻撃に用いられる IP アドレス間の相関解析,” 情報処理学会 コンピュータセキュリティシンポジウム 2011（CSS）論文集, vol.2011, no.3, pp.185–190, 2011 年 10 月.

[D-7] 千葉大紀, 森達哉, 後藤滋樹, “SVM による IP 攻撃通信の判別法,” 情報処理学会 第 73 回全国大会講演論文集, vol.2011, no.1, pp.491–492, 2011 年 3 月.

[D-8] 古谷諭史, 芝原俊樹, 千葉大紀, 秋山満昭, 八木毅, 会田雅樹, “ネットワークの運用形態に着目した同一攻撃基盤に属する悪性ドメイン名推定技術,” 電子情報通信学会 暗号と情報セキュリティシンポジウム（SCIS）, 2017 年 1 月.

[D-9] 山本幸二, 大石和臣, 櫻井幸一, 須崎有康, 千葉大紀, 松本晋一, 森達哉, 吉岡克成, “第 25 回 USENIX Security Symposium 調査報告,” 電子情報通信学会 暗号と情報セキュリティシンポジウム（SCIS）, 2017 年 1 月.

[D-10] 山西宏平, 芝原俊樹, 高田雄太, 千葉大紀, 秋山満昭, 八木毅, 大下裕一, 村田正幸, “畳み込みニューラルネットワークを用いた URL 系列に基づくドライブバイダウンロード攻撃検知,” 情報処理学会 コンピュータセキュリティシンポジウム 2016（CSS）論文集, vol.2016, no.2, pp.811–818, 2016 年 10 月.

[D-11] 松本晋一, 千葉大紀, 須崎有康, 朴美娘, “2016 Network and Distributed System Security Symposium (NDSS 2016) 参加報告,” 電子情報通信学会 技術研究報告（信学技報）, vol.116, no.80, ICSS2016-8, pp.39–44, 2016 年 6 月.

[D-12] 久世尚美, 石倉秀, 八木毅, 千葉大紀, 村田正幸, “複数のハニーポットにおいて観測された情報に基づく通信のネットワーク上の特徴を考慮したぜい弱性スキャン識別,” 電子情報通信学会 技術研究報告（信学技報）, vol.115, no.488, ICSS2015-55, pp.47–52, 2016 年 3 月.

[D-13] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada, “Malware Classification based on Time Series of Network Behavior Using Deep Learning,” Annual Computer Security Applications Conference (ACSAC) Poster Session, December 2015.

[D-14] Takeo Hariu, Keiichi Yokoyama, Mitsuhiro Hatada, Takeshi Yada, Takeshi Yagi, Mitsuaki Akiyama, Tomonori Ikuse, Yuta Takata, Daiki Chiba, and Yasuyuki Tanaka, “Security Intelligence for Malware Countermeasures to Support NTT Group’s Security Business,” NTT Technical Review, vol.13, no.12, pp.1–7, December 2015.

[D-15] 針生剛男, 横山恵一, 畑田充弘, 矢田健, 八木毅, 秋山満昭, 幾世知範, 高田雄太, 千葉大紀, 田中恭之, “NTT グループのセキュリティビジネスを支えるマルウェア対策用セキュリティインテリジェンス,” NTT 技術ジャーナル, vol.27, no.10, pp.18–22, 2015 年 10 月.

[D-16] 久世尚美, 石倉秀, 八木毅, 千葉大紀, 村田正幸, “通信の多様化に向けた生物の環境適応性に基づく Web サイトへのぜい弱性スキャン検知,” 情報処理学会 コンピュータセキュリティシンポジウム（CSS）2015 論文集, vol.2015, no.3, pp.512–519, 2015 年 10 月.

[D-17] 知識友紀江, 千葉大紀, 後藤滋樹, “ブラウザ間通信を用いたアクセス集中時のコンテンツ配信法,” 情報処理学会 第 75 回全国大会講演論文集, vol.2013, no.1, pp.705–707, 2013 年 3 月.

[D-18] 佐藤亮, 千葉大紀, 後藤滋樹, “マニフェストファイルの分析による Android マルウェア検出法,” 情報処理学会 第 75 回全国大会講演論文集, vol.2013, no.1, pp.557–559, 2013 年 3 月.

[D-19] 永井信弘, 千葉大紀, 後藤滋樹, “HTTP 通信の時間軸解析による Web 感染型マルウェア検知,” 情報処理学会 第 75 回全国大会講演論文集, vol.2013, no.1, pp.549–551, 2013 年 3 月.

[D-20] 戸部和洋, 森達哉, 千葉大紀, 下田晃弘, 後藤滋樹, “実行ファイルに含まれる文字列の学習に基づくマルウェア検出方法,” 情報処理学会 コンピュータセキュリティシンポジウム（CSS）2010 論文集, vol.2010, no.2, pp.777–782, 2010 年 10 月.