Robust Malicious Communication Detection for Multi-layered Defense by Overcoming Anti-analysis Techniques

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2020

Toshiki SHIBAHARA

List of publication

Journal papers

1. Toshiki Shibahara, Kohei Yamanishi, Yuta Takata, Daiki Chiba, Taiga Hokaguchi, Mitsuaki Akiyama, Takeshi Yagi, Yuichi Ohsita, and Masayuki Murata. Event de-noising convolutional neural network for detecting malicious url sequences from proxy logs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 101, No. 12, pp. 2149–2161, 2018.

2. Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, Kunio Hato, and Masayuki Murata. Evasive malicious website detection by leveraging redirection subgraph similarities. *IEICE Transactions on Information and Systems*, Vol. 102, No. 3, pp. 430–443, 2019.

3. Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Kunio Hato. Efficient dynamic malware analysis for collecting http requests using deep learning. *IEICE Transactions on Information and Systems*, Vol. 102, No. 4, pp. 725–736, 2019.

Refereed Conference Papers

1. Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Yuta Takata, and Takeshi Yada. Poster: Detecting malicious web pages based on structural similarity of redirection chains. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1671–1673. 2015.

2. Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada. Efficient dynamic malware analysis based on network behavior using deep learning. In *Proceedings of the 59th Annual IEEE Global Communications Conference*, pp. 1–7, 2016.

3. 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. In *Proceedings of the 2017 IEEE International Conference on Communications*, pp. 1–7, 2017.

4. Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, and Takeshi Yada. Detecting malicious websites by integrating malicious, benign, and compromised redirection subgraph similarities. In *Proceedings of the 41st IEEE International Conference on Computers, Software and Applications*, Vol. 1, pp. 655–664. 2017.

5. Toshiki Shibahara, Daiki Chiba, Mitsuaki Akiyama, Kunio Hato, Daniel Dalek, and Masayuki Murata. Unknown family detection based on family-invariant representation. In *NeurIPS 2018 Workshop on Security in Machine Learning*, 2018.

6. Toshiki Shibahara, Hirokazu Kodera, Daiki Chiba, Mitsuaki Akiyama, Kunio Hato, Ola Söderström, Daniel Dalek, and Masayuki Murata. Cross-vendor knowledge transfer for managed security services with triplet network. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pp. 59–69. 2019.

Non-Refereed Technical Papers

1. Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Yuta Takata, and Takeshi Yada. Detecting malicious web pages based on structural similarity of redirection chains. *Computer Security Symposium*, Vol. 2015, No. 3, pp. 496–503, 2015 (in Japanese).

2. Toshiki Shibahara, Daiki Chiba, Mitsuaki Akiyama, Kunio Hato, and Masayuki Murata. Unknown family detection using adversarial training. *Computer Security Symposium*, Vol. 2018, No. 2, pp. 348–355, 2018 (in Japanese).

3. Toshiki Shibahara, Hirokazu Kodera, Daiki Chiba, Mitsuaki Akiyama, Kunio Hato, Ola Söderström, Daniel Dalek, Masayuki Murata. Efficient incident detection by predicting potential important alerts. *Computer Security Symposium*, Vol. 2019, pp. 1092–1099, 2019 (in Japanese).

Preface

Cybersecurity is actively studied because cyberattacks can damage enterprise reputations and cause enormous financial losses. Multi-layered defenses, which are combinations of various detection systems to improve detection capabilities, are generally deployed to mitigate diverse and dynamically changing attacks. Detection systems can be roughly categorized according to two perspectives: detection targets or input data. The systems can be divided according to detection targets into those for pre-infection and post-infection detection. The systems can be further divided according to input data into those for network-based and host-based detection.

In this thesis, we focus on network-based detection systems, because they are effective for both pre-infection and post-infection detection. Effective network-based detection requires exhaustive beforehand collection of malicious communications, which has become difficult because attackers are employing anti-analysis techniques. Specifically, attackers use cloaking to conceal malicious websites and use environment-aware malware to conceal communications between malware samples and attackers. In this thesis, we propose systems and a method for overcoming such anti-analysis techniques.

First, we propose a system for detecting malicious websites without collecting all malicious data. Compromised websites have similar traits because attackers use search engines to automatically discover vulnerable websites. We therefore build a classifier by leveraging both malicious and compromised websites. The proposed system detects 143 more malicious websites employing anti-analysis techniques than does a conventional system. This system enhances exhaustiveness of collected malicious websites and improves detection capabilities for network-based pre-infection detection.

Next, we propose a system for detecting communications to malicious websites from simple logs such as proxy logs. We focus on sequences of destination URLs, because some artifacts of malicious redirections can be extracted from simple logs by considering several nearby URLs. We compare three approaches for classifying URL sequences: an individual-based approach, a convolutional neural network (CNN), and a novel event de-noising CNN (EDCNN). Evaluation results show that only our EDCNN achieves practical classification performance, a true positive rate (TPR) of 99.1% and a false positive rate of 3.4%. Using detected malicious communications, we can improve capabilities for network-based pre-infection detection.

Then, we propose a system for efficiently collecting HTTP requests with dynamic malware analysis. Specifically, our system analyzes a malware sample over a short period, then determines whether analysis should be continued or suspended. In the proposed system, we apply a recursive neural network, which has recently exhibited high classification performance in the field of natural language processing. In an evaluation with 42,856 malware samples, our proposed system collects 94% of novel HTTP requests and reduces analysis time by 82% in comparison with a system that continues all analyses. We can improve network-based post-infection detection by using the collected HTTP requests.

We also propose a method for detecting dynamically changing attacks even when some malicious communications cannot be collected by anti-analysis techniques. Specifically, we investigate how to improve existing deep neural network-based systems in terms of detecting unknown families, namely new types of malicious websites or malware samples. We focus on the tendency that some features are inherent across different families because malicious data include similar code or produce similar behaviors to exploit vulnerabilities or to cost-effectively achieve a successful attack. Therefore, we build a classifier that prioritizes family-invariant features. Our evaluation results show that our method outperforms conventional optimization methods, which optimizes a classifier only so that it accurately classifies malicious and benign data, by at most 19%, 19%, and 7% in terms of TPR for malicious websites, Android applications, and PE files, respectively. This method can improve network-based detection and compensate for degradation caused by anti-analysis techniques.

Acknowledgments

This thesis could not have been accomplished without the assistance of many people, and I would like to acknowledge all of them.

First of all, I would like to express my great gratitude to my superviser, Professor Masayuki Murata, for his generous guidance and insightful comments throughout my Ph.D.

I am heartily grateful to the members of my thesis committee, Professor Takashi Watanabe, Professor Toru Hasegawa, Professor Teruo Higashino, and Professor Morito Matsuoka of Graduate School of Information Science and Technology, Osaka University, for their multilateral reviews and perceptive comments.

I express my appreciation to all of past and present colleagues and secretaries of the Advanced Network Architecture Research Laboratory, Graduate School of Information Science and Technology, Osaka University. Furthermore, I must acknowledge past and present managers and colleagues of NTT Secure Platform Laboratories and NTT Security (Sweden).

I cannot conclude my acknowledgement without expressing my thanks to my parents and family. Thank you for your giving me invaluable supports throughout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Advancement of information and communication technologies has increased the number of available Internet-based services. With the growing number of personal computers, Internet of Things (IoT) devices, and enterprise infrastructures connected to the Internet, malware-infected hosts have caused more and more critical damage. In 2016, for example, over 200,000 hosts in over 150 countries were infected with ransomware called WannaCry [23], and IoT malware called Mirai conducted serious distributed denial of service (DDoS) attacks [5]. In 2018, nearly $500 million in cryptocurrency was stolen from a cryptocurrency exchange [13]. Since such cyberattacks can disrupt enterprise reputations and cause enormous financial loss, cybersecurity is actively studied.

Cybersecurity systems must accurately detect malware infections, but delivery campaigns and malware behaviors are diverse and changing. Delivery campaigns can be roughly divided into two types: drive-by download attacks that lure victims to malicious websites and exploit browser or plugin vulnerabilities [34, 86] and social engineering attacks that use email or malicious web advertisements to make victims install malware themselves [57, 79]. Malware behavior differs depending on attacker goals, such as data exfiltration [16], file encryption [23], or DDoS attacks [5]. Moreover, delivery campaigns and malware behaviors adapt to evade detection systems. Specifically, attackers continuously modify their exploited vulnerabilities, malicious website domains, spam email

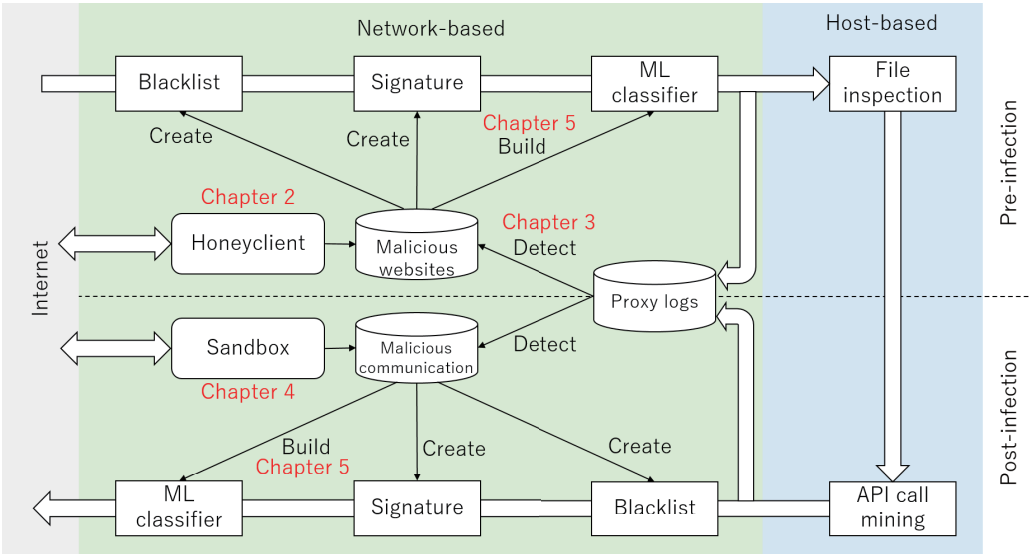


Figure 1.1: Multi-layered defense and contributions of this thesis.

text, utilized application programming interfaces (API), and domains for communication by infected hosts [11, 38, 49].

Multi-layered defenses, which are combinations of various detection systems to improve detection capabilities, are generally deployed to mitigate diverse and dynamically changing attacks (Fig. 1.1). Detection systems can be roughly categorized according to two perspectives: detection targets or input data. The systems can be divided according to detection targets into those for pre-infection and post-infection detection, where pre-infection detection focuses on delivery campaigns and malware themselves, and post-infection detection focuses on malware behavior after an infection. The systems can be further divided according to input data into those for network-based and host-based detection, where network-based detection uses communications monitored on a network as input data, while host-based detection uses API calls and registry accesses on a host. By combining different types of detection systems, multi-layered defense becomes robust against diverse and changing attacks. Even if attackers evade some systems, damage can be prevented when at least one system can detect a malware infection.

In this thesis, we focus on network-based detection systems, because they are effective for both

pre-infection and post-infection detection. Host-based detection is not always effective for post-infection detection, because it may be disabled by malware that terminate processes and services associated with anti-virus software [1, 45]. In contrast, network-based detection monitors network communications and is inaccessible from infected hosts, and thus cannot be disabled by malware.

There are three main approaches by which network-based detection detects malicious communications based on URLs, domains, or IP addresses: blacklists [19, 91], regular expression signatures [77, 135], and machine learning classifiers [25, 122]. These approaches are used for different purposes, depending on their generalization performance and false positive rates. Blacklists do not produce false positives but cannot detect malicious communications slightly different from known ones. Signatures detect more malicious communications than do blacklists, but produce some false positives. Machine learning classifiers have the highest potential for generalization, but tend to produce the most false positives. Since these approaches detect communications that are identical or similar to known malicious ones, effective network-based detection requires exhaustive beforehand collection of malicious communications. For pre-infection detection, a web crawler called a honeyclient is a typical collection tool. A honeyclient accesses many websites and identifies malicious ones to collect malicious communications [19, 121]. A typical tool for post-infection detection is a sandbox, which collects malicious communications by executing malware in a safe environment [77, 127]. Missed malicious communications can be detected in communication logs recorded for post-analysis, where we can apply more complicated approaches than are possible in real-time detection. Post-analysis is more useful for pre-infection detection, because there will be more missed malicious communications associated with earlier attack stages than those associated with later stages. We must continuously collect malicious communications in these ways, because attackers continuously change the URLs, domains, or IP addresses they use for malicious purposes. The collected malicious communications are applied to network-based detection. As mentioned above, the detection capabilities of network-based detection depend on the exhaustiveness of collected malicious communications and generalization performance of detection approaches.

Collecting malicious communications is crucial for making network-based detection effective, but has become difficult because attackers now employ anti-analysis techniques developed for subverting detection analyses and systems. Specifically, attackers use cloaking [55] to conceal

malicious websites and use environment-aware malware [54] to conceal communications between malware and attackers. This thesis proposes systems and a method for overcoming anti-analysis techniques by two approaches: improving the exhaustiveness of collected malicious communications and the generalization performance of detection approaches. In the former, we propose systems for minimizing the negative effects of anti-analysis techniques on malicious communication collection. In the latter, we propose a method for detecting dynamically changing attacks even when some malicious communications cannot be collected by anti-analysis techniques. Specifically, we focus on unknown families, which are new types of malicious websites or malware. To make multi-layered defense robust, we need component systems that robustly detect malicious communications without being disturbed by anti-analysis techniques. This thesis therefore considers improvements to detection capabilities for pre-infection or post-infection network-based detection.

Anti-analysis techniques are not limited to those described above. Attackers commonly use obfuscation (so-called packing), anti-disassembly, and anti-debugging to disturb reverse engineering, which is a manual analysis for determining malware functioning [109]. To thwart blacklist-based detection systems, attackers use domain-flux and fast-flux, which are techniques for rapidly changing malicious domains and IP addresses [41, 94]. To conceal malicious content in Web-based attacks, attackers use IP-based cloaking that blocks accesses from pre-defined IP addresses [88]. They also use malvertising [61] to deliver malware through Web advertisements. There has been recent study of the threat of adversarial examples, which apply small perturbations to evade machine learning classifiers [27]. Among these anti-analysis techniques, we focus on cloaking and environment-aware malware, because overcoming these techniques is promising for improving detection capabilities of network-based detection, as described below. The following sections describe in detail our research problems, related work, and the proposed systems and method.

1.2 Cloaking

Attackers have applied drive-by download attacks to distribute malware through the Web. When a client accesses a landing URL that is the starting point of an attack, it is redirected to an exploit URL via multiple redirection URLs. Browser or plugin vulnerabilities are exploited at the exploit

URL, infecting the client with malware [86]. To prevent drive-by download attacks, security researchers and vendors use a honeyclient to access malicious websites and collect malicious data such as malicious URLs, web content, and redirections [24, 120]. Using the collected data, they create signatures for anti-virus software and build machine learning classifiers for malicious web content [19, 25, 113], redirection chains [72, 78, 135], and exploit kits [122].

To defeat these detection systems, attackers conceal malicious data through an anti-analysis technique called cloaking [55], making it difficult to collect all data from malicious websites. Specifically, attackers may change the destination URL according to browser fingerprinting such as IP addresses and client environments (e.g., the family and version of the browser and whether it is running on an actual operating system (OS) or being emulated as a virtual machine) [51, 52]. If the client environment differs from that of the attacker's target, the attacker modifies client accesses by altering server responses from malicious websites.

One approach for detecting malicious websites without being affected by cloaking is aggregating large-scale user traffic [46, 114]. Attackers redirect clients to the same redirection URL from various landing URLs, then redirect them again to an exploit URL that targets the detected environment. Geographical diversity and uniform client environments can be used as traits of malicious websites. However, these systems require logs provided by anti-virus vendors or large ISPs, making them generally difficult to deploy. From the perspective of deployment, we designed our systems to use data collected using a honeyclient and proxy.

Another approach is detecting compromised websites. Li et al. [60] detected compromised JavaScript code that triggers malicious redirections by comparing code with a clean counterpart. However, this system requires both a clean version and a sample of the compromised JavaScript code. We consider detecting malicious websites by leveraging only already compromised websites.

In Chapter 2, we propose a system for detecting malicious websites without collecting all malicious data [100, 101, 104, 105]. Even if we cannot observe some malicious data, such as exploit code and malware, we can always observe the compromised websites into which attackers inject redirection code to malicious data. Since attackers use search engines to automatically discover vulnerable websites, compromised websites have similar traits. For example, attackers use specific search queries called "search engine dorking" [60, 112] to discover vulnerable websites

built using old versions of content management systems (CMSs). We therefore build a classifier by leveraging both malicious and compromised websites. Specifically, we convert all websites observed during an access into a redirection graph whose vertices are URLs and edges are redirections between two URLs, and classify it with graph mining. To perform this classification, we integrate similarities between the redirection graph's subgraphs and redirection subgraphs shared across malicious, benign, and compromised websites. Evaluating our system with crawling data from 455,860 websites, we find that it achieves a 91.7% true positive rate (TPR) for malicious websites containing exploit URLs and a low false positive rate (FPR) of 0.1%. Moreover, it detects 143 more malicious websites using anti-analysis (evasion) techniques than a conventional system does. These detected evasive websites are built by, for example, compromising a vulnerable CMS. This system enhances exhaustiveness of collected malicious websites and improves detection capabilities of network-based pre-infection detection.

In Chapter 3, we propose a system for detecting communications with malicious websites from simple logs such as proxy logs [106,107]. We focus on sequences of destination URLs, because some artifacts of malicious redirections can be extracted from simple logs by considering several nearby URLs. Specifically, simple logs contain malicious landing, redirection, and exploit URLs with their sequential order preserved. We call these *URL sequences*, and URL sequences including accesses to malicious websites *malicious URL sequences*. To find an effective approach for classifying URL sequences, we compare three approaches: an individual-based approach, a convolutional neural network (CNN), and a novel event de-noising CNN (EDCNN). Our EDCNN reduces the negative effects of benign URLs redirected from compromised websites included in malicious URL sequences. Evaluation results show that only our EDCNN achieved practical classification performance: a TPR of 99.1%, and FPR of 3.4%. By using detected malicious communications for creating blacklists, regular expression signatures, and machine learning classifiers, we can improve detection capabilities of network-based pre-infection detection.

1.3 Environment-aware Malware

For infected host detection, network-based systems such as malicious communication detection [22, 77] and blacklist-based detection have played an important role. These systems are difficult to evade because malicious network behavior is definitely observed. For example, attackers coordinate infected hosts to accomplish their mission by distributing configuration files or sending commands from command and control (C&C) servers.

To maintain high detection rates in these systems, novel HTTP requests that have not been collected in previous analyses are collected by executing new malware in a sandbox [8, 127]. However, there are now malware that circumvent HTTP request collection by confirming analysis environments, Internet connections, or execution dates before disclosing malicious behavior [48, 62, 73]. Malware thus do not always disclose their malicious behavior.

To counter environment-aware malware, Kirat et al. proposed an effective analysis system that runs on actual hardware [54]. However, attackers can still thwart this system, so not all malware are effectively analyzed [73]. In this way, environment-aware malware decreases efficiency in collecting HTTP requests even though efficiency of dynamic analysis has already been a problem. Specifically, long-term analysis of the more than 350 million new malware detected in 2016 [118] is infeasible in limited amounts of time. Hence, malware is typically analyzed over fixed short periods, such as 5 min [31]. If more characteristic patterns can be identified from collected requests, we can increase the detection capabilities of network-based detection. Efficient dynamic analysis is thus required to collect more novel HTTP requests in shorter analysis time.

In Chapter 4, we propose a system for efficiently collecting HTTP requests through dynamic malware analysis [102, 103]. Specifically, our system analyzes malware over short periods, then determines whether analysis should be continued or suspended. This determination is made on the basis of network behavior observed in the short-period analyses. To make accurate determinations, we focus on the fact that malware communications resemble natural language from the viewpoint of data structure. We apply recursive neural networks [111], which have recently exhibited high classification performance in the field of natural language processing. In an evaluation of 42,856 malware samples, our proposed system collects 94% of novel HTTP requests and reduces analysis

time by 82% in comparison with a system that continues all analyses. Our system improves efficiency of dynamic malware analysis by suspending analyses of malware that do not disclose malicious behavior because of the analysis environment. We can improve network-based post-infection detection by using the collected HTTP requests to create blacklists, regular expression signatures, and machine learning classifiers.

1.4 Unknown Families

Malicious websites and Android applications are created with attack tools and used to efficiently accomplish attackers' objectives. For example, malware is created with toolkits [18] and malicious websites are created with exploit kits [34]. We define a set of malicious data created with the same attack tool as a *family*. When detecting new malicious data, a machine learning classifier is expected to achieve higher detection capabilities than the blacklists and signatures described in Section 1.1. Among machine learning algorithms, we focus on deep neural networks (DNNs), because they have outperformed traditional machine learning algorithms such as support vector machines (SVMs) [126] and random forest [17], and because researchers and security vendors have proposed sophisticated DNN-based detection systems [71, 90, 129].

However, attackers are continuously developing new attack tools to evade DNN-based systems [119]. Unknown families created with new attack tools exhibit unknown malicious behavior, obfuscation algorithms, and anti-analysis functions [47, 124]. The emergence of unknown families causes significant changes in the features of malicious data, a phenomenon referred to as *concept drift*. Concept drift is known to degrade classification performance of classifiers that assume classification targets are drawn from the same distribution as the training data, that is, independent and identically distributed (i.i.d.) random variables [49]. Machine learning techniques for detecting malicious data implicitly assume i.i.d. variables, because they are designed on the basis of machine learning for image recognition and natural language processing. Consequently, DNN-based systems trained using known families have difficulty detecting unknown families [84].

To prevent degraded classification performance under changes in data distributions, a method for detecting concept drift has been proposed [49]. This method statistically compares training data

with test data to detect concept drift. If a change is detected, the classifier is retrained using a new data distribution (known and unknown families in our setting). However, this method requires retraining of classifiers. Another method has been proposed for extracting invariant features [11]. This method designs invariant features on the basis of changes in malicious data. However, this method cannot improve existing DNN-based systems in terms of detection of unknown families.

In Chapter 5, we describe how to improve existing DNN-based systems in terms of detecting unknown families [98, 99]. Unknown families can be quite difficult to detect when their features completely differ from those of known families. Even so, some features are inherent across families because malicious data include similar code or produce similar behaviors to exploit vulnerabilities or to cost-effectively achieve a successful attack. For example, in drive-by download attacks, applications exploiting browsers or their plugins are limited to Flash, PDF, and Java, and website redirections are always abused to lure victims to exploit websites [19]. When features inherent across known families (family-invariant features) are prioritized in classification, unknown families become easier to detect. Therefore, we aim at building a classifier that prioritizes family-invariant features. We evaluate whether our methods robustly improve DNN-based systems in terms of detecting unknown families in three case studies. We select diverse and common targets of detection systems for cybersecurity and prepared datasets for malicious websites, Android applications, and portable executable (PE) files. Evaluation results show that our method robustly improves DNN-based systems without depending on datasets. Specifically, the method outperforms a conventional optimization method that optimizes a classifier only so that it accurately classifies malicious and benign data by at most 19%, 19%, and 7% in terms of TPRs for malicious websites, Android applications, and PE files, respectively. This method can improve network-based detection and compensate for degradation caused by anti-analysis techniques.

Chapter 2

Evasive Malicious Website Detection by Leveraging Redirection Subgraph Similarities

2.1 Introduction

Attackers have distributed malware through the Web by drive-by download attacks. When a client accesses a landing URL that is a starting point of attacks, the client is redirected to an exploit URL via multiple redirection URLs. At the exploit URL, vulnerabilities in browsers and/or their plugins are exploited, and the client is finally infected with malware [86]. This infected client suffers from damage, such as sensitive data leakage and illegal money transfer, and/or is integrated into distributed denial-of-service attacks. To expose more users to threats of drive-by downloads, attackers compromise benign websites and inject redirection code to malicious websites such as redirection and exploit URLs. Attackers compromise benign websites and create malicious websites automatically. Since websites built using the old version of CMSs are vulnerable, attackers automatically discover them with search engines by using specific search queries, typically called "search engine dorking" [60, 112], and compromise them to turn them into landing URLs. Malicious websites are automatically created with exploit kits [34].

To prevent drive-by download attacks, security researchers and vendors analyze malicious data, e.g., malicious URLs, web content, and redirections. They can create signatures of anti-virus software and build classifiers on the basis of malicious web content [19, 25, 113], redirection chains [72, 78, 135], and exploit kits [122]. All the above systems require malicious data to be collected by accessing malicious websites with a honeyclient, which is a decoy browser [24, 120]. Unfortunately, collecting all malicious data from malicious websites is not easy because attackers conceal the data with evasion techniques. To increase the exploitation success rate, attackers check clients by browser fingerprinting and change the destination URL depending on the fingerprint, e.g., IP address and client environments (the family/version of a browser on a real operating system (OS)/virtual machine/emulator) [51, 52]. In addition, if a client environment differs from the environment of attackers' targets, the attackers thwart the client's accesses by changing the server responses of malicious websites, which is called "cloaking" [55]. In other words, collecting all malicious data requires *correct* access by the clients of attackers' targets. Although multiple accesses to malicious websites by various clients improves the coverage of collected malicious data, preparing or emulating all the clients (i.e., OSes, browsers, and plugins) is not a realistic solution due to the requirement of a large amount of computational resources [52].

In this Chapter, we propose a system for detecting malicious websites without collecting all malicious data. Even if we cannot observe parts of malicious data, e.g., exploit code and malware, we can always observe compromised websites, into which attackers inject redirection code to malicious data. Since vulnerable websites are automatically discovered with search engines by attackers, compromised websites have similar traits. Therefore, we built a classifier by leveraging not only malicious but also compromised websites. More precisely, we convert all websites observed at the time of access into a redirection graph, whose vertices are URLs and edges are redirections between two URLs, and classify it with a graph mining approach. To perform classification, we integrate similarities between the redirection graph's subgraphs and redirection subgraphs shared across malicious, benign, and compromised websites. As a result of evaluating our system with crawling data of 455,860 websites, we find that it achieves a 91.7% TPR for malicious websites containing exploit URLs at a low FPR of 0.1%. Moreover, it detects 143 more malicious websites that use evasion techniques than conventional systems. These detected evasive websites are, for

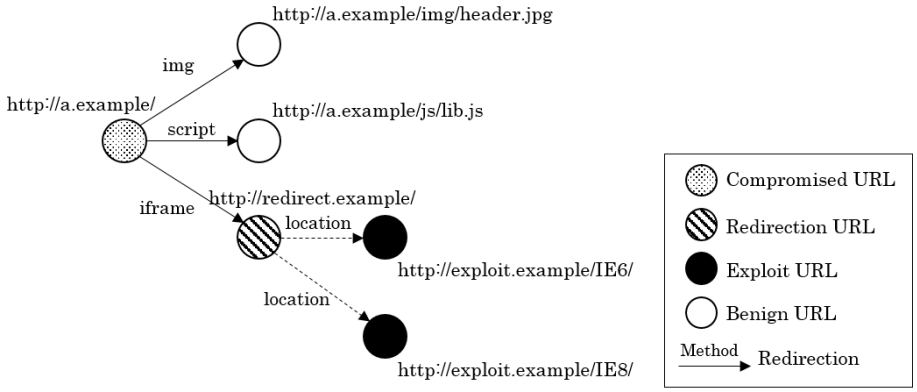


Figure 2.1: Redirection graph of a motivating example.

example, built by compromising a vulnerable CMS. These results show that our system successfully captures redirection subgraphs of not only malicious but also compromised websites.

Our contributions are summarized as follows.

- We propose a system that detects malicious websites by leveraging all websites observed at the time of access even if all malicious data cannot be collected due to evasion techniques.
- We show that leveraging the redirection subgraphs of benign, compromised, and malicious websites enhances the classification performance; the benign subgraphs contribute to reducing false positives such as subgraphs of web advertisements and the compromised and malicious subgraphs contribute to improving true positives such as subgraphs of compromised CMSs and exploit kits.

2.2 Motivating Example

We use simplified websites to demonstrate the effectiveness of our approach. Figure 2.1 shows a redirection graph. When a client accesses the URL of a compromised website, i.e., `http://a.example/`, the server responds with web content such as Fig. 2.2, and the client additionally requests the web content of the URLs specified in HTML tags. The `iframe` tag at line 13 in Fig. 2.2 is injected by an attacker, and the client is redirected to the next URL, `http://redirect.example/`, without being aware of it because this `iframe` tag is written in an invisible state and is outside the display.

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8" />
 <title>Sample Website</title>
 <script src="js/lib.js" type="text/javascript"></script>
 </head>
 <body>
 <img src="img/header.jpg" width="800px" alt="header" />
 ... [snipped] ...
 </body>
</html>
<iframe src="http://redirect.example/" style="position:absolute;width:0px;left:-99px;
    display:none;"></iframe>
```

Figure 2.2: Iframe injection at `http://a.example/`.

```
var ua = "";
try{
 new ActiveXObject("dummy");
}catch( e ){
 ua = window.navigator.userAgent.toLowerCase();
}
if( ua.indexOf("msie 6") != -1 ){
 location.href = "http://exploit.example/IE6/";
}
else if( ua.indexOf("msie 8") != -1 ){
 location.href = "http://exploit.example/IE8/";
}
```

Figure 2.3: Evasion and browser fingerprinting at `http://redirect.example/`.

When the client accesses the URL specified by the `iframe` tag, it loads web content that contains the JavaScript code shown in Fig. 2.3. Lines 2–6 in Fig. 2.3 are evasion code that checks whether the client is a browser emulator or an actual browser. A browser emulator is usually designed to never raise exceptions regarding ActiveXObject [24]. However, since the code in Fig. 2.3 intentionally throws an ActiveXObject error, only browsers with correct exception handlers can execute browser fingerprinting code at line 5. The code at line 5 stores the UserAgent strings of the client in a variable `ua`. The variable `ua`, i.e., `navigator.userAgent` strings, is used for the following conditional branches at lines 7 and 10, and the redirection code at line 8 or 11 is executed if the variable contains "msie 6" or "msie 8" strings, respectively. In other words, Internet Explorer (IE) 6 is redirected to `http://exploit.example/IE6/`, and IE8 is redirected to `http://exploit.example/IE8/`. However, when clients other than IE6 and IE8 are used, no redirection occurs. Browser emulators also cannot execute browser fingerprinting code due to exception handling, so no redirection occurs.

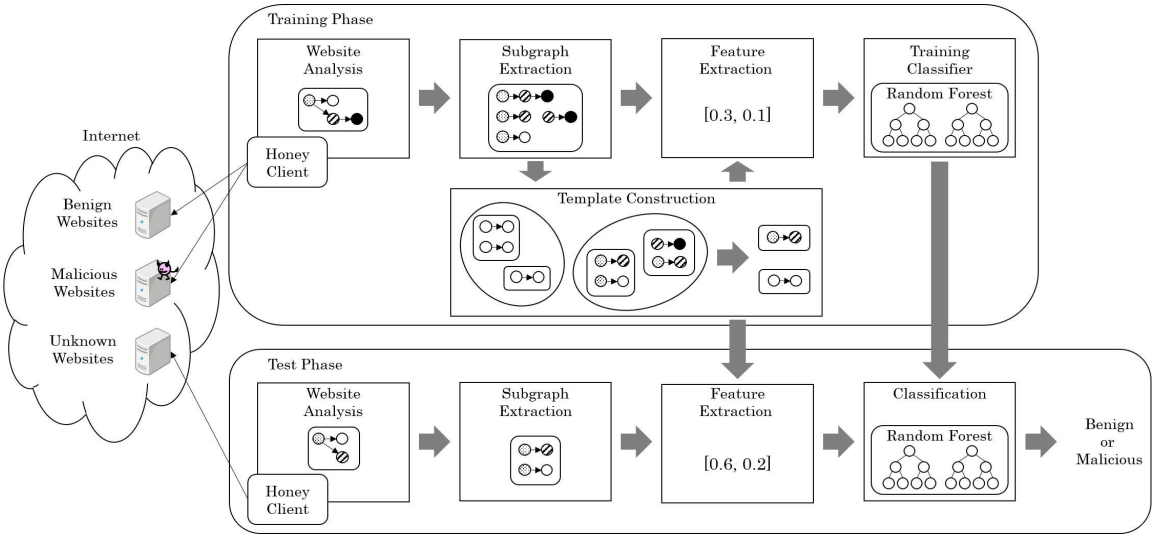


Figure 2.4: System framework.

Therefore, this example illustrates a website where only IE6 and IE8 can access exploit URLs and collect exploit code.

Conventional systems cannot detect this example for several reasons. A high-interaction honeyclient that uses an actual browser fails to collect exploit code and malware when the browser is not IE6 or IE8 due to browser fingerprinting. Similarly, a low-interaction honeyclient, i.e., a browser emulator, also fails to execute redirection code due to evasion code even if it emulates IE6 or IE8. Consequently, systems detecting malicious websites on the basis of URLs, redirections, and web content does not work effectively when these honeyclients cannot collect malicious data.

Our system can detect malicious websites that use evasion techniques by utilizing the redirection graphs of all websites observed at the time of access without being limited to those of malicious websites created with exploit kits. In the above example, we can certainly observe the redirection of the invisible `iframe` tag and redirections to benign URLs to which the compromised website originally refers, i.e., `http://a.example/js/lib.js` specified by the `script` tag and `http://a.example/img/header.jpg` specified by the `img` tag in Fig. 2.1. In other words, we can detect malicious websites by building our classifier with features representing redirection subgraphs of easily compromised websites even if we fail to observe parts of malicious redirections and web content due to evasion techniques. As shown in Section 2.5, our system detected 143 more evasive

malicious websites than the conventional content-based system.

2.3 Proposed System

We describe the design and implementation of our proposed system, which detects malicious websites on the basis of the redirection graphs of all websites.

2.3.1 System Design

Websites consist of multiple URLs and redirections between them. Their structure is represented as redirection graphs whose vertices are URLs and edges are redirections. To take advantage of their structure, we utilize a graph mining approach. One common approach is to perform classification by leveraging similarities of graphs. More precisely, subgraphs are extracted from each graph, and the similarities of many pairs of graphs are calculated on the basis of the number of subgraphs shared by the graphs. This approach leads to high classification accuracy but also has a drawback: a high computational cost. To achieve both high classification accuracy and low computational cost, we reduce the computational cost of subgraph extraction and the number of similarity calculations. In Section 2.3.2, we describe how to reduce the computational cost of subgraph extraction. Here, we discuss the number of similarity calculations. The classification is performed on the basis of similarities between test and training data. A large number of training data improves classification accuracy but results in a large number of similarity calculations, i.e., $O(NM)$, where N and M represent the number of training and test data, respectively. Our system constructs a comparatively small number of templates, which are subgraphs shared across training data, and performs classification on the basis of similarities between test data and templates. The number of similarity calculations is reduced to $O(M)$. Note that a graph mining approach also has another drawback: a large memory requirement. Because a large memory has become easier to obtain, we focus on only computational cost in this Chapter.

Figure 2.4 illustrates the framework of our system. In the training phase, we collect labeled redirection graphs (malicious or benign) with the honeyclient [120]. The redirection graphs are decomposed into their subgraphs. Then, templates are constructed from redirection subgraphs

shared across them. Their feature vectors are extracted on the basis of the similarities between their subgraphs and templates. The classifier of random forest is trained with their feature vectors. In the test phase, we collect unlabeled redirection graphs with the honeyclient. Their feature vectors are extracted in the same manner as in the training phase and classified using the trained classifier. We explain details of our system in the next subsection.

2.3.2 Implementation

Subgraph Extraction. We collect web content at each URL and the methods used for redirections by analyzing websites with a honeyclient. The websites are represented as redirection graphs, i.e., directed graphs whose vertices are URLs and edges are redirections. The most important structure of redirection graphs for detecting malicious websites is the path from landing to exploit URLs. To reduce the computational cost, we extract only subgraphs that have an important structure, i.e., path-shaped subgraphs. Excluding subgraphs that have a branch structure reduces the computational cost. Let $G = (V, E)$ denote a redirection graph, where V is a set of vertices, and $E \subseteq (V \times V)$ is a set of edges. Edge (v_i, v_j) represents the redirections from vertex v_i to vertex v_j. A set of paths, P, is defined as $P = \{p_{i,j} | v_j \in c(v_i), v_i \in V\}$, where $p_{i,j}$ is a path from v_i to v_j, and $c(v_i)$ is a set of vertices reachable from v_i through edges. We use the information of vertices and edges as a feature of a subgraph. The feature of a subgraph sg is represented as (m, r), where m is a vector containing the information of vertices and r is a vector containing the information of redirections. A redirection graph is decomposed into a set of features of subgraphs extracted from all path-shaped subgraphs.

For the information of vertices, we calculate the maliciousness of web content because exact matching of URLs and their web content can be easily evaded by changing the characters of URLs and small pieces of their web content. The maliciousness is calculated with machine learning using the 22 widely used features in Table 2.1. These features are extracted with the honeyclient [120]. The features are divided into three types: redirection, HTML, and JavaScript. We extract five redirection features: HTTP 3xx redirections to different domains (No. 1), redirections to files used for exploit (Nos. 2–4), and redirection without referer (No. 5). We extract three HTML features: the suspicious Document Object Model (DOM) position (No. 6), invisible content (No. 7), and exploitable classid

Table 2.1: Features for calculating maliciousness of web content.

No.	Type	Feature
1	Redirection	HTTP 3xx redirections to different domain
2		Redirection to Flash file
3		Redirection to PDF file
4		Redirection to Java Applet File
5		Redirection without referer
6	HTML	Suspicious DOM position
7		Invisible content
8		Exploitable classid
9	JavaScript	# of Element object functions
10		# of String object functions
11		# of Node object functions
12		# of ActiveXObject functions
13		# of Document object functions
14		# of Navigator object functions
15		# of object-encoding functions
16		# of Time object functions
17		# of eval functions
18		# of fingerprinting functions
19		# of obfuscated content
20		# of content containing shellcode
21		# of long parameters
22		Entropy

Table 2.2: Methods of redirections.

Type	Examples of methods		
HTTP 3xx	HTTP 301	HTTP 302	
HTML tag	`iframe`	`script`	`link`
JavaScript	`document.write`	`innerHTML`	

(No. 8). We extract 13 JavaScript features: the number of functions (Nos. 9–18), the number of suspicious content (Nos. 19–20), the number of long parameters (No. 21), and entropy (No. 22).

Fingerprinting functions are identified by arguments including versions of plugins. Obfuscated content is identified by Latin-1 code or a comma delimited string whose length is 128 or more. Content containing shellcode is identified by a string including 128 or more Unicode/non-printable ASCII characters. We define long parameters as JavaScript functions' arguments whose lengths are 350 or more. To calculate maliciousness, we apply random forest [17] as a classifier. We

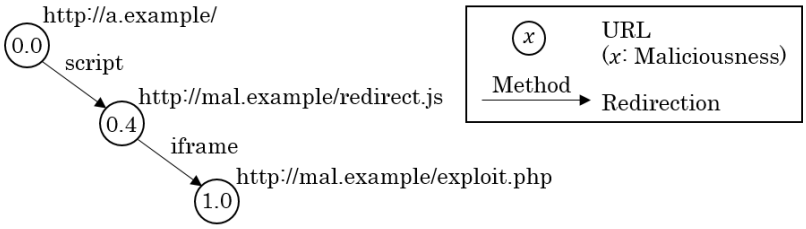


Figure 2.5: Example of a subgraph.

train the classifier by using the training data in Section 2.4. For the information of edges, we use redirection methods and destination domains (external or internal). Table 2.2 shows the three types of redirection methods: HTTP 3xx, HTML tag, and JavaScript. HTML tag redirections are triggered by tags for acquiring external sources such as `iframe`, `script`, and `link`. JavaScript redirections are triggered by DOM modification functions such as `document.write` and `innerHTML`.

Figure 2.5 shows an example of a subgraph consisting of three vertices and two edges. The maliciousness of the first, second, and third vertices is 0.0, 0.4, and 1.0, respectively. The redirection methods of the first and second edges are `script` and `iframe`. The destination domains of the first and second edge are *external* and *internal*. The feature of a subgraph of this example sg is represented as (m, r), where $m = [0.0, 0.4, 1.0]$, $r = [\texttt{script}-\text{external}, \texttt{iframe}-\text{internal}]$. The information of edges is represented by the hyphenation of the redirection method and the destination domain. Hereafter, we attach new indexes to features of subgraphs and represent a subgraph feature set extracted from a redirection graph as $SG = \{sg_i\}$ for convenience.

Template Construction. We split redirection graphs into clusters composed of similar redirection graphs and construct templates from the clusters. The similarity utilized for clustering is defined similarly to the Dice index. The Dice index, D, between set A and set B is defined as $D = 2 \times |A \cap B|/(|A| + |B|)$. If the intersection of two subgraph feature sets is defined on the basis of both the redirection information and the maliciousness, the amount of difference in the maliciousness does not properly affect the similarity because the maliciousness is a continuous value. For this reason, we define the intersection of two subgraph feature sets on the basis of only the redirection information and use the maliciousness as weighting coefficients.

The similarity function $S(SG_i, SG_j)$, given subgraph feature sets SG_i and SG_j, is defined as

$$S(SG_i, SG_j) = \frac{2 \times \sum_{(m_k, m_l) \in \Lambda} s(m_k, m_l)}{|SG_i| + |SG_j|}, \tag{2.1}$$

$$s(m_k, m_l) = \frac{1}{1 + \alpha \times |m_k - m_l|^2}, \tag{2.2}$$

$$\Lambda = \{(m_k, m_l) | (m_k, r_k) \in SG_i, (m_l, r_k) \in SG_j\}, \tag{2.3}$$

where α is a weight coefficient. Here, $s(m_k, m_l)$ denotes the similarity function given the maliciousness m_k and m_l. If there are multiple subgraph-feature-pair possibilities, we adopt the pair that has higher similarity than the others. The optimal pair can be quickly found by using the Hungarian algorithm [75].

This similarity is utilized for clustering. If the maximum similarity between redirection graphs belonging to one cluster and redirection graphs belonging to another is higher than threshold β, the two clusters are merged. This process is conducted from when each cluster is composed of one redirection graph to when no cluster can be merged.

Clustering is computationally expensive because similarities between all pairs of redirection graphs need to be calculated, i.e., $O(n^2)$, where n denotes the number of redirection graphs. We leverage locality sensitive hashing (LSH) [28] to avoid calculating the similarities of redirection graphs that have low similarities. Reducing the computational cost of clustering enables us to reduce computational resources or increase the number of candidate hyperparameters used for optimization. We encode a redirection graph into a vector by using bag-of-words representation to apply LSH. The vector contains the number of redirection methods or JavaScript functions/objects. The redirection methods include HTTP 302, `iframe` tag, and `script` tag. JavaScript functions/objects include `documentw.write, innerHTML, setInterval`, and `ActiveXObject`. The hash function is formulated as $h(x) = \lfloor \frac{a^T x + b}{c} \rfloor$, where a is a vector and b is a real number. Here, $\lfloor x \rfloor$ denotes the largest integer, which is equal to or less than x. Each element of a is chosen from the normal distribution whose mean is 0 and variance is σ^2. The real number, b, is chosen from the uniform distribution whose range is $[0, c]$. The parameters $\sigma^2 = 10, c = 1$ are selected so that the number of

websites that have the same hash values is not too small.

We construct templates from the clusters composed of γ or more redirection graphs. The template, T, is a set of features of subgraphs whose redirection information is shared across all redirection graphs in the cluster $C = \{SG_i\}$. Since maliciousness is a continuous value, we use the average maliciousness as the maliciousness of the template. Template T is formulated as

$$T = \{(m_i, r_i) | \forall SG \in C, \exists (m_i, r_i) \in SG\}, \tag{2.4}$$

$$m_i = \frac{1}{|C|} \sum_{m_j \in M_i} m_j, \tag{2.5}$$

$$M_i = \{m_j | (m_j, r_i) \in SG_j, SG_j \in C\}. \tag{2.6}$$

Feature Extraction. A high similarity between the features of subgraphs of a redirection graph and a template indicates that the redirection graph contains the template as its subgraph. In other words, we can encode the redirection graphs of websites into numerical values by calculating similarities between features of subgraphs of redirection graphs and templates. We extract a feature vector x containing similarities between a subgraph feature set SG and templates T_i: $x = [S(SG, T_1), ..., S(SG, T_N)]$, where N is the number of templates.

Classification. These feature vectors are classified by using supervised machine learning. We use random forest [17], which can classify a large amount of data accurately and quickly. We use a `randomForest` package in R [87]. Note that the classification algorithm is not limited to random forest; other algorithms of supervised machine learning can be applied.

2.4 Experimental Setup

Our proposed system is designed to detect not only malicious redirection graphs containing exploit URLs but also evasive malicious redirection graphs, which do *NOT* contain all malicious data. We evaluate the detection performance of our system using these redirection graphs. Here, we describe the experimental setup for evaluation.

Table 2.3: Features of the redirection-based system.

No.	Feature
1	# of different domains
2	Path length
3	# of HTTP 3xx redirections
4	# of different domain HTTP 3xx redirections
5	# of consecutive HTTP 3xx redirections
6	# of consecutive different domain HTTP 3xx redirections
7	# of consecutive short redirections
8	Median of redirection duration
9	Average of redirection duration
10	Minimum of redirection duration
11	Maximum of redirection duration

2.4.1 Conventional Systems for Comparison

We evaluate the effectiveness of our system by comparing it with conventional systems. Some conventional systems detect malicious websites by matching redirection or exploit URLs [122, 135]. These systems suffer from false negatives when targeted URLs are concealed by evasion techniques. Other conventional systems using statistical features [19, 25, 72] are assumed to be more robust against evasion techniques because their targets are not limited to specific URLs. For this reason, we compare our system with conventional systems that use statistical features. Note that we cannot compare our system with conventional systems that leverage large-scale user traffic [46, 114] because our system is supposed to use web content and redirections collected with a honeyclient.

We compare our system with the content-based system, the redirection-based system, and the combination of these systems. The content-based system extracts widely used features listed in Table 2.1 such as the conventional system [19] and classifies them by using random forest. If one or more pieces of web content in a redirection graph are detected, the redirection graph is classified as malicious. If no piece of web content is detected, the redirection graph is classified as benign.

The redirection-based system extracts features from paths between landing URLs and final destinations of redirections and classifies them by using random forest. Table 2.3 shows a list of features that have been proposed for the conventional system [72]. More precisely, the features are the number of different domains (No. 1), path length (No. 2), HTTP 3xx redirections (Nos. 3–6),

and redirection duration (Nos. 7–11). Short redirections are defined as redirections that occur in no more than one second. If one or more paths in a redirection graph are detected, the redirection graph is classified as malicious. If no path is detected, the redirection graph is classified as benign.

The combination of the content-based and redirection-based systems (combination system for short) classifies a redirection graph as malicious if it is detected by the content-based or redirection-based system. A redirection graph is classified as benign if it is detected by neither system.

2.4.2 Ground Truth

We collect the redirection graphs used for the evaluation by accessing websites listed on public blacklists [67, 128] or a list of popular websites [4] by using the low-interaction honeyclient [120]. Some websites listed on public blacklists no longer contain malicious web content, and websites listed on the popular-website list can be compromised and forced to engage in attacks. Since we need ground truth of redirection graphs, we access websites by using low-interaction and high-interaction honeyclients [3] almost at the same time. The high-interaction honeyclient detect exploit URLs on the basis of system behavior such as unintended process creation and file/registry accesses. We label redirection graphs detected by the high-interaction honeyclient as malicious and redirection graphs listed on the popular-website list and not detected as benign. We do not use the redirection graphs listed on public blacklists but not detected because they might not be detected due to the discrepancy between the targeted environment of exploit and the environment of the high-interaction honeyclient. Since redirection graphs are all websites observed at the time of access, malicious redirection graphs contained compromised and malicious websites. Note that the malicious redirection graphs do not necessarily contain exploit URLs due to evasion techniques even if the high-interaction honeyclient access exploit URLs. We use malicious redirection graphs that do not contain exploit URLs as evasive malicious redirection graphs.

Conventional systems require labeled web content or paths. For the content-based system, we label web content as malicious if it is collected from destinations of malicious redirections and domains of corresponding URLs are different from those of landing URLs. We label the web content of benign redirection graphs as benign. For the redirection-based system, we label paths

Table 2.4: Dataset.

	Label	Number	Period
Training	Malicious	2,170	Jan.–Apr. 2016
	Benign	199,982	Aug. 2016
1st Test	Malicious	365	May–Sep. 2016
	Benign	249,958	Aug. 2016
2nd Test	Evasive	3,385	Jan.–Sep. 2016

including exploit URLs as malicious and paths of benign websites as benign.

2.4.3 Dataset

We use one training dataset and two test datasets for our evaluation as shown in Table 2.4. The training dataset consists of 2,170 malicious redirection graphs collected from Jan.–Apr. 2016 and 199,982 benign redirection graphs collected in Aug. 2016. We confirm that each malicious redirection graph in the training dataset contains at least one malicious redirection. Note that we can collect malicious redirection graphs because their targeted environment and the environment of the low-interaction honeyclient are identical or evasion techniques are not used. The benign training dataset should be collected during the same period as the malicious dataset. However, we have not collected benign redirection graphs during that period; hence, we use benign redirection graphs collected in Aug. 2016 as a training dataset. Note that the collection period is not assumed to affect the evaluation results because the redirection graphs of benign websites are not subject to change.

We use the first test dataset to evaluate the detection performance with malicious redirection graphs containing exploit URLs from a large number of benign redirection graphs. It consists of 365 malicious redirection graphs collected from May–Sep. 2016 and 249,958 benign redirection graphs collected in Aug. 2016. We use the second test dataset to evaluate the detection performance with evasive malicious redirection graphs. It consists of 3,385 evasive malicious redirection graphs collected from Jan.–Sep. 2016. This dataset does not overlap with the training data collected in the period Jan.–Apr. 2016.

2.4.4 Hyperparameter Optimization

The hyperparameters are a weight coefficient, α, of similarity, threshold β for clustering, and threshold γ for template construction. In addition to them, we optimize the training dataset. The prevalence of new exploit kits or updates to exploit kits changes the redirection subgraphs of malicious websites. Therefore, malicious redirection graphs detected in the past do not always contribute to improving the classification performance. We optimize the percentage θ of malicious redirection graphs that we use for training. The number of malicious redirection graphs is limited, but we can collect a large number of benign redirection graphs. We optimize the ratio η of malicious to benign redirection graphs.

We further split the training dataset into a prior-training dataset and a validation dataset. The prior-training dataset includes 90% of the training dataset selected from the oldest data and is used for training a classifier The validation dataset includes 10% of the training dataset selected from the newest data and is used for evaluating the classification performance. We select the hyperparameters that had the highest classification performance. To evaluate the performance, we use the f-measure defined as:

$$\begin{aligned} \text{f-measure} &= \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \\ \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \end{aligned}$$

where TP, FN, TN, and FP denote the number of true positives, false negatives, true negatives, and false positives, respectively. The best hyperparameters are $\alpha = 1$, $\beta = 0.4$, $\gamma = 2$, $\theta = 50$, and $\eta = 1 : 10$. The hyperparameters of random forest, i.e., the number of decision trees and the number of features for each decision tree, are optimized by using the `tuneRF` function of the `randomForest` package in R [87] when a classifier is trained.

If a small difference in the hyperparameters of our system results in a large difference in its classification performance, it makes our system difficult to deploy because the hyperparameters need to be carefully optimized. To determine whether they are difficult or not to optimize, we

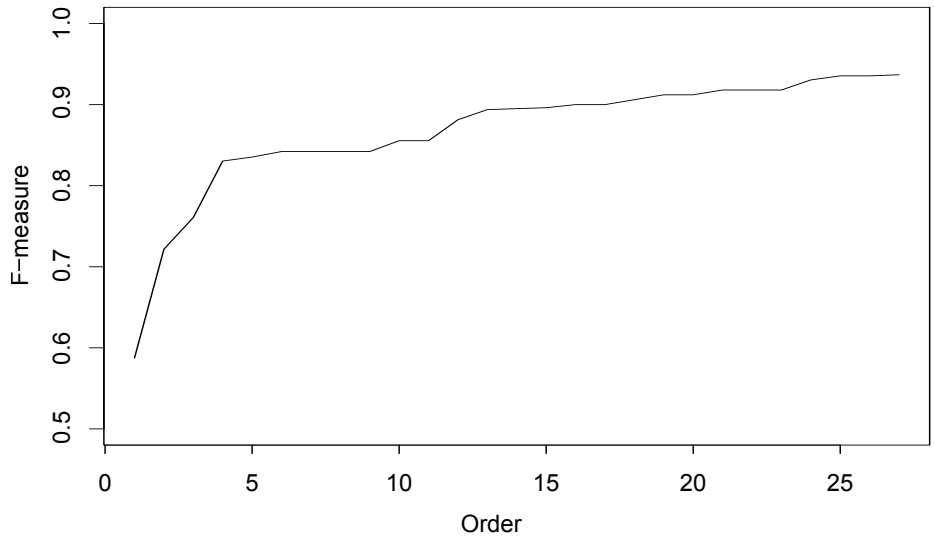


Figure 2.6: F-measure of different hyperparameters.

Table 2.5: Hyperparametes and their orders.

Order	α	β	γ	Order	α	β	γ
1	1	0.4	10	15	1	0.8	2
2	10	0.8	10	16	10	0.6	2
3	100	0.8	10	17	10	0.6	5
4	1	0.8	10	18	1	0.6	5
5	10	0.8	5	19	1	0.4	5
6	100	0.4	5	20	10	0.4	2
7	100	0.6	5	21	10	0.4	5
8	100	0.6	10	22	10	0.8	2
9	100	0.8	5	23	100	0.8	2
10	10	0.6	10	24	1	0.6	2
11	100	0.4	10	25	100	0.4	2
12	10	0.4	10	26	100	0.6	2
13	1	0.8	5	27	1	0.4	2
14	1	0.6	10				

investigate classification performance with different system-specific hyperparameters, i.e., α, β, and γ. We select α from 1, 10, and 100, β from 0.4, 0.6, and 0.8, and γ from 2, 5, and 10. The classifier is trained by using the prior-training dataset and calculate the f-measure on the validation dataset. Figure 2.6 shows the f-measure of every hyperparameter. The hyperparameters are arranged in the ascending order of f-measures. The hyperparameters and their orders are listed in Table 2.5.

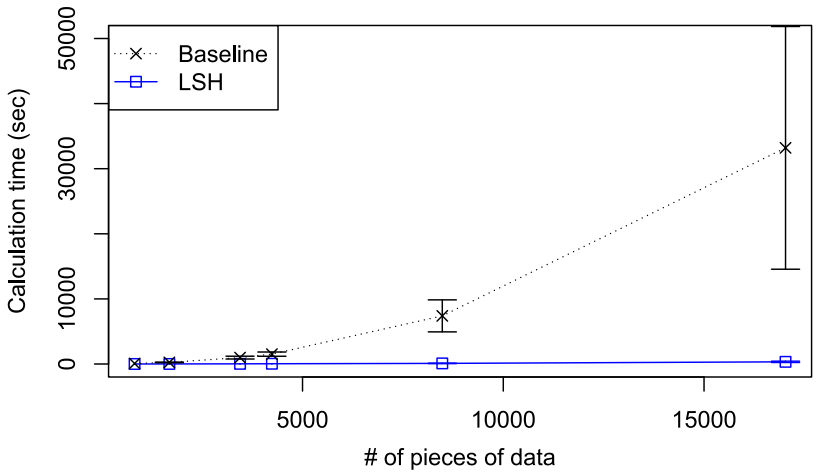


Figure 2.7: Calculation time of clustering. Error bars represent standard deviation.

Since our system achieves high f-measures with several hyperparameters, we can find with simple optimization methods such as a grid search.

Our system leverages LSH to reduce the computational cost of clustering. We evaluate the effectiveness of LSH by comparing the calculation times with clustering without LSH (baseline). Figure 2.7 shows the average and standard deviation of calculation time. If the number of data is the same, the calculation time differs depending on the hyperparameters. This result shows that LSH drastically speeds up clustering. Note that we select hyperparameters of LSH so that similar redirection graphs have the same hash value with high probability. Such hyperparameters cannot achieve optimal clustering speed but suppress degradation of classification performance caused by inaccurate clustering results.

2.5 Experimental Results

We now report the experimental results. In Section 2.5.1, we show the results of the first test dataset (see Section 2.4.3 for more detail) including malicious redirection graphs containing exploit URLs. We compare our system with conventional systems in terms of classification performance, degradation in classification performance over time, and calculation time. We further investigate constructed templates, false positives, and false negatives to obtain a better understanding of our

Table 2.6: Classification performance with malicious redirection graphs containing exploit URLs.

System	Proposed	Content	Redirect	Comb.
TPR (recall)	**0.9057**	0.8465	0.2564	0.8876
FPR	**0.0007**	0.0022	0.0028	0.0041
Precision	**0.6631**	0.0385	0.1388	0.2401
F-measure	**0.7657**	0.5300	0.1801	0.3780
AUC	**0.9938**	0.9664	0.6408	0.9774
TPR (FPR=0.1%)	**0.9171**	0.7726	0.2256	0.7726

system. Lastly, we present a case study of server-side cloaking to show the effectiveness of our system. In Section 2.5.2, we show the results of the second dataset including evasive malicious redirection graphs. To avoid duplicate evaluation, we focus on the detection capability of evasive malicious redirection graphs in this subsection. Specifically, we show classification performance, false negatives, and a case study of an evasive malicious website. The prototype version of our system is installed on an Ubuntu server with four 12-core CPUs and 128-GB RAM.

2.5.1 Detecting Malicious Websites with Exploit URLs

We report the evaluation results of the first test dataset including malicious redirection graphs containing exploit URLs in terms of classification performance, degradation in classification performance over time, and calculation time. We also show analysis results of constructed templates, false positives, false negatives, and a redirection graph of server-side cloaking.

Classification Performance. We evaluate our system by using widely used metrics: TPR (also known as recall), FPR, precision, f-measure, area under the receiver operating characteristic curve (AUC), and TPR at a fixed FPR of 0.1%. As shown in Table 2.6, our system outperforms all conventional systems for all metrics. These results show that leveraging the redirection graphs of all websites contributes to improving the classification performance.

Classification Performance Degradation Over Time. The prevalence of new exploit kits and updates to exploit kits degrades the classification performance. Therefore, we evaluate the performance degradation over time. Figure 2.9 shows the TPR of malicious redirection graphs collected for the first, second, and third or following month of the test dataset. The TPRs of our system and

Table 2.7: Statistics of templates

		Malicious	Benign	Compromised
# of templates		167	246	54
Order of importance	Highest	1	10	7
	Lowest	454	467	458
	Average	111.7	293.1	343.0

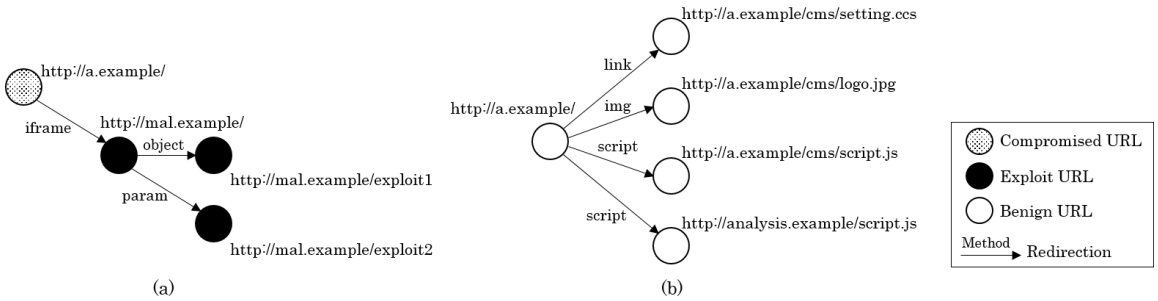


Figure 2.8: Redirection subgraphs of templates.

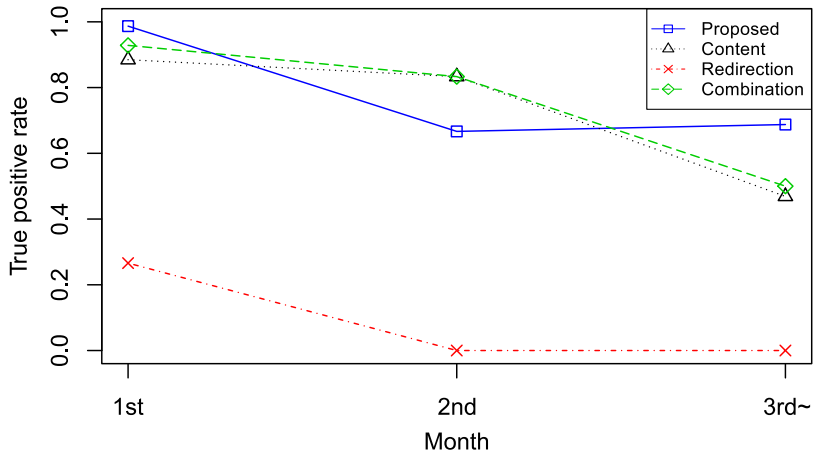


Figure 2.9: True positive rate degradation over time.

the redirection-based system become smaller for the second month of the test dataset than for the first month. The TPRs of the content-based and combination systems become smaller for the third or following months of the test dataset than for the first and second months. The degradation of the combination system is similar to that of the content-based system because the content-based system has a dominant role in the classification of the combination system. The degradation of our system is

Table 2.8: Calculation time for one website (sec).

System	Proposed	Content	Redirection	Comb.
Feature extraction	0.1251	**0.0033**	0.0073	0.0106
Classification	0.0005	**0.0003**	0.0002	0.0004

steeper than those of conventional systems because our system utilizes more types of websites. That is, our system is focused on malicious, benign, and compromised websites, and a change in structure of any one of them degrades classification performance. Specifically, classification performance of our system degrades because features of compromised websites change between the first and second months of the test dataset. The training data includes a large number of compromised websites constructed by using WordPress, which is a CMS. Similarly, the number of compromised websites constructed with WordPress in the test dataset is high in the first month but low in the second month. The number of such compromised websites in the third or following months of the test dataset is higher than that in the second month. This is why the proposed system has a higher TPR than the conventional systems in the third or following months of the test dataset.

Calculation Time. To detect malicious websites, we must analyze a large number of websites. To evaluate the capability of large-scale analysis, we evaluate the calculation time of feature extraction and classification. Table 2.8 shows that our system requires a 35 times longer calculation time than the content-based system. However, our system finishes the feature extraction and classification of one redirection graph in less than 0.13 seconds. The calculation time of feature extraction and classification is shorter than that of the download and execution of web content. Therefore, our system can classify a large number of redirection graphs collected by using a honeyclient.

Template Analysis. We analyze the templates of our system to elucidate the improvement in classification performance. Table 2.7 shows the number of templates and order of the *importance* calculated with the `randomForest` package for each label of template. The labels of templates are those of redirection graphs from which templates are constructed: malicious, benign, and malicious and benign (compromised for short). The *importance* represents the level of contribution of each template to classification. More than half the templates are constructed from only benign websites. This is because benign redirection graphs outnumbered malicious ones. However,

malicious templates tended to have a higher *importance* than benign and compromised templates. Some benign and compromised templates also have the highest *importance*.

One malicious template that has a high *importance* contains redirection subgraphs relevant to the Angler Exploit Kit as shown in Fig. 2.8(a). The `iframe` tag redirecting to the exploit URL of a different domain is injected at the landing URL. The malicious web content at `http://mal.example/` exploits a use-after-free vulnerability (CVE-2014-4130) and also contains redirection to other malicious web content such as Flash (CVE-2015-0313). Another malicious template that has a high *importance* contains redirection subgraphs relevant to an exploit kit and compromised website. Since it contains all websites related to drive-by download attacks, it is a large template composed of many redirection subgraphs.

One compromised template that has a high *importance* contains redirection subgraphs relevant to a CMS as shown in Fig. 2.8(b). Websites created with a CMS tend to be targeted and compromised by attackers. For this reason, redirection subgraphs relevant to the same CMS are included in malicious and benign redirection graphs. The landing URL includes some HTML tags redirecting to Cascading Style Sheets, image, and JavaScript code. These redirections are included in the template of the CMS. The landing URL also includes a `script` tag redirecting to an analysis service because many websites' administrators use it.

One benign template that has a high *importance* contains redirection subgraphs relevant to an advertisement. If websites use the same advertisement service, they have the same redirection subgraph for obtaining advertisement content. This is why a template relevant to the advertisement is constructed. We omit illustrating the template because it contains too many URLs to be depicted in a limited amount of space.

By using these templates, our system can classify redirection graphs on the basis of the structural similarities to exploit kits, CMSs, and advertisements. If the content-based system wrongly classified advertisement content as malicious, our system classified its redirection graphs as benign by referring to other web content and redirections relevant to the advertisement. In addition, if the content-based system could not detect malicious web content, our system detected its redirection graphs by taking the compromised CMSs into account.

As shown in the aforementioned examples, effective templates are essential to achieve high

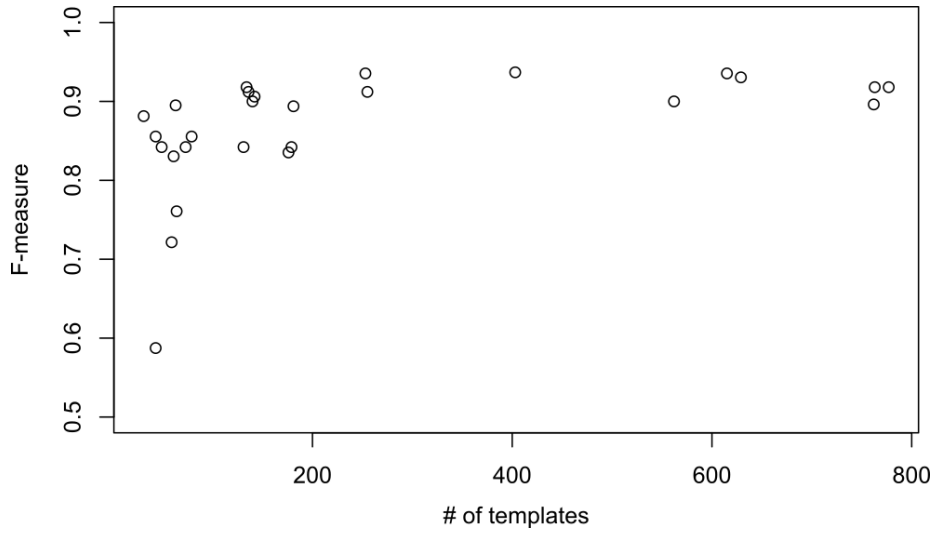


Figure 2.10: F-measure and the number of templates on different hyperparameters.

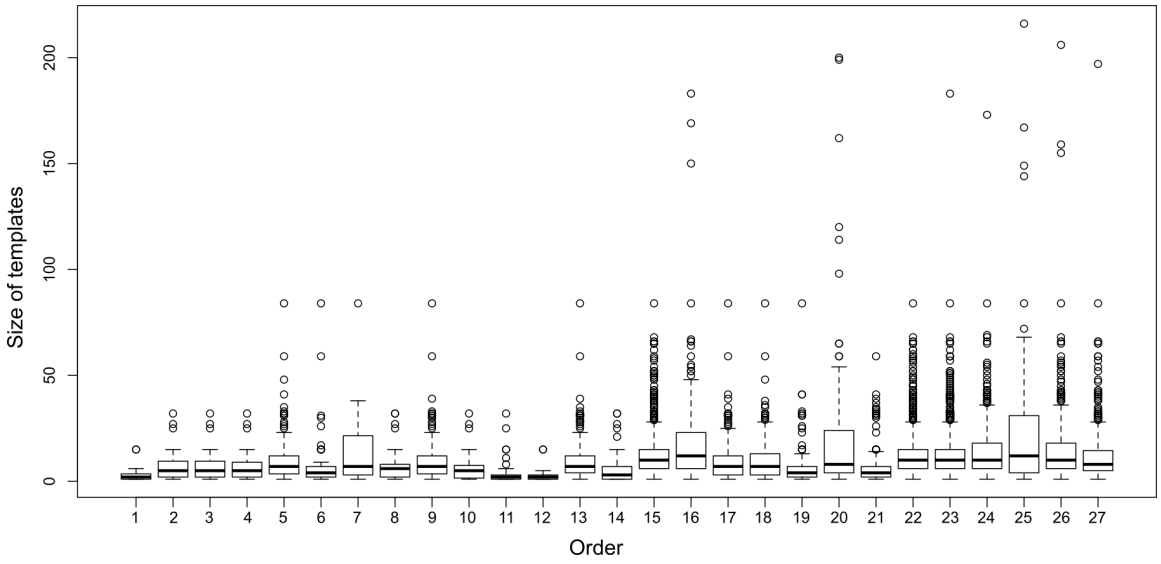


Figure 2.11: Distribution of size of templates on different hyperparameters.

classification performance. We investigate classification performance on the basis of different templates. Specifically, we investigate the number of templates, distribution of template sizes, and f-measure with different hyperparameters. We use the number of redirection subgraphs consisting of a template as the size of a template. When the number of templates is small, our system has a low f-measure as shown in Fig. 2.10. This is because coverage of redirection subgraphs is not

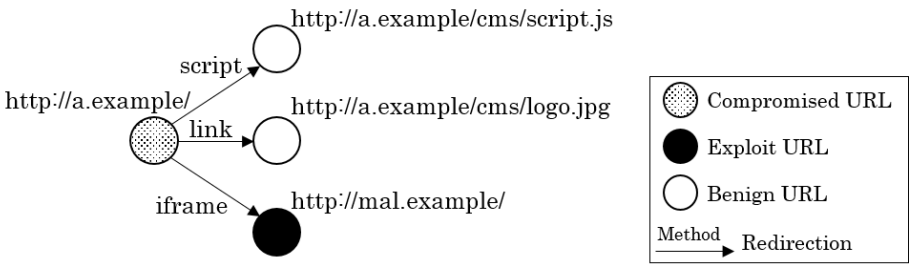


Figure 2.12: Redirection graph of a website launching server-side cloaking.

sufficient to classify redirection graphs accurately. Figure 2.11 shows the distribution of template sizes with different hyperparameters. The hyperparameters are arranged in the ascending order of f-measures. When the f-measure is high, large templates tend to be constructed and variance of the distribution tends to be high. When the f-measure is low, most templates are small. This is because small templates are similar to many redirection graphs and unable to extract discriminative features. To achieve a high classification performance, various differently sized templates need to be constructed.

False Positives and False Negatives. We identifies two main cases of false positives with manual inspection. The first is websites created with a CMS for electronic commerce sites. They contains multistage redirections to JavaScript code. Similarly, malicious redirection graphs have multistage redirections to malicious JavaScript code. This structural similarity causes the false positives. The second case is websites created with a CMS and slightly modified by their administrators. Some redirections for advertisement or analysis services are injected to landing URLs. The redirection graphs of these websites are similar to malicious redirection graphs created by compromising CMSs. Note that our system accurately classifies redirection graphs of plain or customized CMS websites as benign.

We identifies one main case of false negatives. On the websites of false negatives, benign JavaScript code that is the destination of redirections from landing URLs is compromised. However, in most malicious redirection graphs, the web content of landing URLs is compromised. The difference in redirection graphs causes false negatives.

Case Study of Server-side Cloaking. We describe a website for which redirection subgraphs of

Table 2.9: Number of TP and TPR at fixed FPR of 0.1%.

System	Proposed	Content	Redirection	Comb.
# of TP	**821**	678	71	669
TPR	**0.243**	0.200	0.021	0.206

compromised websites need to be captured for detection. The redirection from the landing URL to the exploit URL is triggered by an injected `iframe` tag as shown in Fig. 2.12. The status code of the HTTP response from the exploit URL is 200, but its body is empty. This website is speculated to be an attempt of server-side cloaking, which detects security vendors or researchers on web servers and conceals malicious web content from them. The exploit URL is created with Rig Exploit Kit, and most attempts to obtain malicious web content are unsuccessful. The content-based system could definitely not detect this website due to the lack of malicious web content.

Our system detects the website by utilizing the redirection graph of all of the websites. The compromised website is created with a CMS that is sometimes compromised by attackers. Moreover, it has a redirection to a different domain with a `iframe` tag. The same redirection is frequently used on malicious redirection graphs. Our system detects this website by capturing both traits of the compromised website and the malicious redirection.

2.5.2 Detecting Evasive Malicious Websites

We report the evaluation results of the second dataset including evasive malicious redirection graphs. To avoid duplicate evaluation, we focus on the detection capability and show classification performance, false negatives, and a case study of an evasive malicious website.

Classification Performance. Table 2.9 shows the number of TP and TPR at a fixed FPR of 0.1%. FPR is fixed using the test dataset of malicious redirection graphs containing exploit URLs. Our system detects 143 more evasive malicious redirection graphs than the content-based system. On the evasive malicious websites detected by our system, the evasion code prevents the low-interaction honeyclient from accessing exploit URLs at redirection URLs. The redirection graph and evasion code are more precisely illustrated as a case study. The content-based system cannot detect some malicious web content at redirection URLs. This is why our system detects more evasive malicious

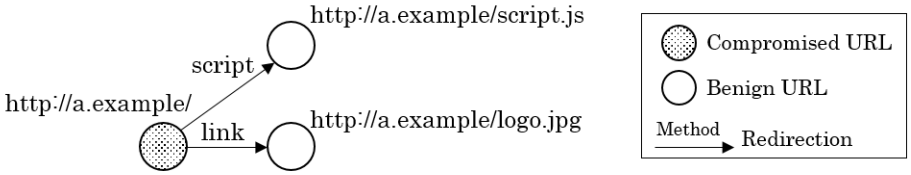


Figure 2.13: Redirection graph of a false negative.

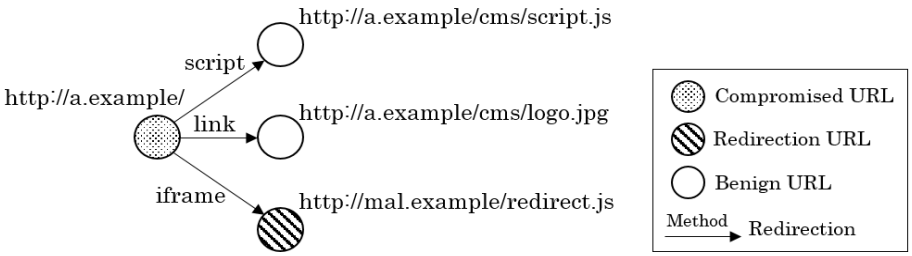


Figure 2.14: Redirection graph of an evasive malicious website.

redirection graphs.

False Negatives. We identifies one main case of false negatives as shown in Fig. 2.13. On these websites, the evasion code is used at landing URLs, i.e., `http://a.example/` in Fig. 2.13. Therefore, the low-interaction honeyclient is not redirected to malicious URLs but only to benign URLs, i.e., `http://a.example/script.js` and `http://a.example/logo.jpg` in Fig. 2.13. The redirection graphs of these websites are the same as those of benign websites. On benign websites, a client is typically redirected to some benign URLs to obtain scripts or images when it accesses a landing URL. As a result, benign redirection graphs are shallow and have many branches. For this reason, our system cannot detect websites where the evasion code is used at landing URLs. Note that we do not find any websites containing malicious redirections after manually inspecting 100 false negatives.

Case Study of Evasive Malicious Website. We describe the evasive malicious website shown in Fig. 2.14. This website (redirection URL) is pointed to by the `iframe` tag injected at the landing URL. Figure 2.15 shows the evasion code created with the Angler Exploit Kit. A different value is assigned to `flag` depending on the error when the ActiveXObject of "Anti-virus" is loaded at line 3. If the error occurs, `true` is assigned, and vice versa at lines 4 and 6. Subsequent malicious

```
var flag;
try {
 var a = new ActiveXObject("Anti-Virus");
 flag = true;
}catch( e ){
 flag = false;
}
```

Figure 2.15: Evasion code at `http://mal.example/redirect.js`.

code is executed only if it is `true`. Since the high-interaction honeyclient has no anti-virus software installed, it raises the exception. However, the low-interaction honeyclient does not raise any exception. By leveraging this difference, the evasive malicious website prevents the low-interaction honeyclient from accessing the exploit URL.

The compromised website is also created with a CMS sometimes compromised by attackers. Similar to the website launching server-side cloaking, our system can detect it by utilizing the redirection graph of all of the websites.

2.6 Limitations

Our system requires redirection subgraphs widely shared across malicious websites to distinguish malicious redirection graphs from benign ones. Therefore, our system does not detect malicious redirection graphs that has subgraphs different from those of typical malicious websites (as discussed in Section 2.5.1) or malicious redirection graphs that contain evasion code used at landing URLs (as discussed in Section 2.5.2). This is a general limitation from which all machine-learning-based systems suffer. To detect uncommon malicious redirection graphs, the number of malicious training data must be increased. To detect malicious redirection graphs that contain evasion code used at landing URLs, systems that detect injected code on compromised websites [60, 112] can be utilized complementarily. We can analyze a relatively small number of websites detected by these systems with various clients and detect malicious websites using collected malicious redirection graphs by using our system.

Another limitation is the degradation in classification performance over time. Conventional systems also have this limitation, but the degradation of our system is steeper than those of the

conventional systems as discussed in Section 2.5.1. Our system is based on redirection subgraphs of many websites such as benign, compromised, and malicious websites. Change in the structure of any one of them degrades the classification performance of our system. The advantage of our system is to achieve high classification performance if training and test data are similar. The evaluation results show our system achieves the highest classification performance on the first month of test data. To maintain high classification performance, the classifier should be retrained every month by using data labeled by high-interaction honeyclients. High-interaction honeyclients are not suitable for large-scale analysis due to their slow processing speed but are useful for labeling a limited number of websites.

2.7 Related Work

2.7.1 Malicious Website Detection

Many systems have been proposed from different perspectives for detecting malicious websites launching drive-by download attacks. Here, we describe conventional systems focuses on large-scale user traffic, system behavior, and web content and redirections.

Large-scale User Traffic. One approach for detecting malicious websites is aggregating large-scale user traffic [46, 114]. Attackers redirect clients to the same redirection URL from landing URLs and then redirect them to the exploit URLs targeting their environment. Geographical diversity and uniform client environments can be used as traits of malicious websites. However, these systems require logs provided by anti-virus vendors or large ISPs, and these logs are generally difficult to obtain. From the perspective of deployment, we design our system to use data collected with a honeyclient.

System Behavior. Decoy client systems using actual browsers have been proposed to detect exploit accurately by monitoring unintended process creation and file/registry accesses [3, 42, 65]. These systems are known as high-interaction honeyclients. They have the limitations of a slow processing speed due to the use of actual browsers and the limited coverage of collected malicious data due to browser fingerprinting as discussed in Sections 2.1 and 2.2. For this reason, they are not suitable for

large-scale analysis.

Web Content and Redirections. Systems in this category are designed for large-scale analysis. They collect web content or redirections by using browser emulators and classify them by using machine learning. Browser emulators developed for light-weight collection are known as low-interaction honeyclients. Malicious web content has distinctive features to exploit known CVE-ID (Common Vulnerabilities and Exposures identification) or trigger malicious redirections. Some systems are focused on JavaScript code and HTML tags [19, 25, 113]. Other systems are focused on multistage redirections such as the difference in domains and duration of redirections [72], URLs shared across malicious websites [135], sequences of URLs [61], and URLs/HTTP headers and redirections between them [122]. These systems are focused on malicious URLs, web content, or redirections, but our system is focused on the redirection graphs of all websites, i.e., malicious/benign web content and malicious/benign redirections.

2.7.2 Compromised Website Detection

Detecting compromised websites is another approach to preventing damage caused by drive-by download attacks. Soska and Christin [112] detect websites that will become malicious in the future by focusing on web content that is not generated by users, such as vulnerable CMSs. Li et al. [60] detect compromised JavaScript code that triggers malicious redirections by comparing it with its clean counterpart. These systems require a clean version of compromised websites or JavaScript code. Our system detects compromised websites by leveraging only already compromised websites and totally benign websites.

2.7.3 Classification of Graphs

Many researchers have studied different approaches for classifying graph-structured data such as protein or medicine.

Graph Kernels. One approach is to design a graph kernel that is a function to calculate similarities between graphs and classify graphs with machine learning algorithms based on kernel methods such as support vector machine [117]. The random walk kernel [33] is based on the co-occurrence of

sequences of labels on vertices or edges in random walks. The shortest-path kernel [15] is based on the co-occurrence of the length of shortest-paths between pairs of vertices. The graphlet kernel [97] is based on the co-occurrence of subgraphs that have k vertices. The Weishfeiler-Lehman graph kernel [96] is based on the co-occurrence of multilabels that are created by iteratively integrating a vertex's and its neighbors' labels. The deep graph kernel [131] is designed to extend the above graph kernels by leveraging latent representations of sub-structures. The deep graph kernel is defined by considering similarities between sub-structures as well as co-occurrence of sub-structures. These methods effectively calculate similarities between graphs on the basis of their sub-structures, but counting the occurrence of a large number of sub-structures is computationally expensive. Our system reduces computational cost by calculating similarities between a redirection graph and a small number of templates.

Convolution on Graphs. Since deep neural networks (DNNs) have achieved outstanding classification performance in image recognition and natural language processing, some studies have applied DNNs to classification of graphs. Duvenaud et al. [30] have proposed a convolutional neural network (CNN) that iteratively convolutes vectors representing a vertex and its neighbors and calculates the summation of convoluted vectors as a representation of a whole graph. Li et al. [59] conduct similar convolutions with the gated recurrent units. Dai et al. [26] designed an architecture of neural networks on the basis of graphical model inference procedures. Niepert et al. [82] apply conventional CNNs by arranging vertices in the order of certain criteria such as centrality. These methods have difficulty extracting features of differently sized redirection subgraphs such as exploit kits, compromised websites, and advertisements because the number of iterations of convolutions must be determined before training. Our system extracts features of differently sized redirection subgraphs by leveraging differently sized templates.

2.8 Summary

We propose a system for detecting malicious websites on the basis of the redirection graphs of all websites even when some malicious web content is concealed. We extract redirection subgraphs shared across malicious, benign, and compromised websites as templates and classified websites

using feature vectors containing similarities between their features of subgraphs and the templates. We find that templates contained redirection subgraphs of exploit kits, compromised websites, and advertiser websites. These templates enable our system to identify malicious websites by capturing redirection subgraphs of compromised websites as well as those of malicious websites. As a result of evaluating our system with crawling data of 455,860 websites, we find that it achieves a 91.7% true positive rate for malicious websites containing exploit URLs at a low false positive rate of 0.1%. Moreover, our system detects 143 more evasive malicious websites than the conventional content-based system.

Chapter 3

Event De-noising Convolutional Neural Network for Detecting Malicious URL Sequences from Proxy Logs

3.1 Introduction

Attackers have increased the number of infected hosts on enterprise networks by using drive-by download attacks despite the efforts of many security researchers and vendors. This type of attack infects a client with malware by exploiting the vulnerabilities of a browser and its plugins through the Web. Attackers lure unsuspecting users of compromised popular websites (landing URLs) and redirect them via redirection URLs to exploit URLs. At exploit URLs, exploit code is executed to force users to download and execute malware samples from malware distribution URLs. In this Chapter, we define a group of such URLs related to a drive-by download attack as a malicious website. Because some users ignore security warnings and software updates, attacks need to be detected and filtered at a network level [132].

To filter the accesses from users to malicious websites, methods for creating blacklists of URLs and domains have been proposed [6, 66]. However, it has become even more difficult to list malicious websites on blacklists before users access them. This is because malicious domains are

used only for a short period, and honeyclients, which are decoy client systems, can obtain exploit code only if their environment (a type of browser and its plugins) matches the targeted environment of attackers [61]. Therefore, detection of users who have already accessed malicious websites by analyzing communication logs on the network has started to be researched.

For communication log analysis, operators of enterprise networks record simplified logs, such as proxy logs [77] because they cannot feasibly record all web content. Simplified logs contain minimal information such as timestamps and URLs but not web content or HTTP headers. For this reason, we cannot focus on conventional features such as malicious content [19] and redirection chains [114] to detect malicious websites in typical enterprise networks. To make matters worse, simplified logs contain communications automatically sent by software other than browsers such as an OS or anti-virus software. These communications make classification difficult because their destination URLs are similar to malicious URLs with random strings and long query parameters. These negative effects on classification can be reduced by eliminating these communications from simplified logs, but some of their fully qualified domain names (FQDNs) are known as disposable domains that contain automatically generated random strings. To identify disposable domains, we must collect domains generated by the same software for six days [21]. In other words, disposable domains of software are difficult to identify right after it is installed. Therefore, we must detect accesses to malicious websites from simplified logs that contain communications sent by software other than browsers.

To tackle these problems, we propose a system for detecting malicious URL sequences from simplified logs by focusing on two points. The first point is that some artifacts of malicious redirections can be extracted from simplified logs by considering several nearby URLs. Specifically, simplified logs contain malicious URLs (i.e., landing URLs, redirection URLs, and exploit URLs), and their sequential order is preserved. In this Chapter, we refer to the sequences of destination URLs as *URL sequences* and the URL sequences including accesses to malicious websites as *malicious URL sequences*. We extract features of malicious redirections from URL sequences to detect accesses to malicious websites. The second point is that features for classifying communications sent by software other than browsers have not been well-designed. Since disposable domains are difficult to classify on the basis of domain-based features as mentioned above, we propose new

features by analyzing their URL paths or queries.

The drawback of machine learning is that it requires many training data. For example, accurately classifying URL sequences requires many training data that include accesses to multiple websites. Malicious URL sequences might include accesses to not only malicious websites but also benign websites accessed in a short period. Even if the malicious URL sequences include accesses to different benign websites, the classifier must detect these URL sequences. To train such a classifier, URL sequences including accesses to a large variety of combinations of websites are needed. By collecting a large number of URL sequences, the coverage of combinations is enhanced, but our system becomes difficult to deploy. To enhance the coverage of combinations with a moderate number of URL sequences, we also propose a data augmentation to classify URL sequences. Since effective data to enhance classification performance differ depending on the classification approach, we generate data suitable for the proposed classification approach.

We use a simple approach and our proposed approach for focusing on malicious redirection. For our approach, we also propose suitable data augmentation. We first describe classification approaches then explain data augmentation.

Convolutional Neural Network. The first approach extracts features of malicious redirection in a simple manner. This approach is based on the similarity between the relation of URLs in URL sequences and the relationships among words in sentences. Specifically, a URL sequence includes many pairs of URLs linked by redirections, i.e., source and destination of redirections, and a sentence includes many pairs of words that are syntactically dependent. Because convolutional neural networks (CNNs) achieve high classification accuracy in natural language processing [43], we applied a CNN to URL sequence classification.

Event De-noising CNN. The second approach extracts features of malicious redirections by using a method we developed for this purpose. This approach is based on the fact that malicious URL sequences include not only malicious URLs related to drive-by download attacks but also benign URLs redirected from landing URLs (e.g., image URLs, JavaScript library URLs, and advertisement URLs) and benign URLs accessed by software other than browsers. To reduce the negative effects of benign URLs, we developed an event de-noising CNN (EDCNN), which is an extension of the CNN

architecture. It reduces the effect of benign URLs that negatively affect classification performance such as *noise* in a URL sequence (generally, a sequence of discrete *events*). More precisely, we extended the architecture to extract features from *two URLs* in the vicinity rather than from *several nearby URLs* in the sequence. This structure enables the same features to be extracted from the same combination of two malicious URLs, even if some benign URLs are inserted between the malicious URLs, i.e., the EDCNN robustly extracts features of malicious redirections. As a result, the EDCNN is expected to more accurately classify URL sequences on the basis of malicious redirections.

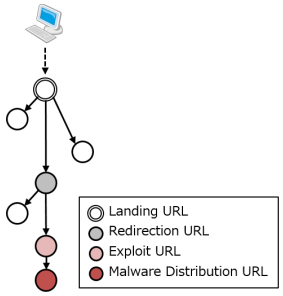
In addition, we propose a new data augmentation suitable for the EDCNN. The EDCNN is expected to propagate features of malicious URLs and ignore those of benign URLs. When training the classifier of the EDCNN, benign URLs in malicious URL sequences have little impact because they are ignored. On the other hand, malicious URLs largely affect training of the classifier. Specifically, positional relationships of malicious URLs need to be diverse for robustly extracting features of malicious redirections by the classifier. Therefore, we diversify them by generating new URL sequences including accesses to randomly selected websites. Including different benign websites in malicious URL sequences makes positional relationships of malicious URLs different. Any benign website can be included in malicious URL sequences because it is ignored by the EDCNN.

We empirically discuss the classification performance and capability of capturing malicious redirections with data collected over seven months. In summary, we make three contributions.

- To the best of our knowledge, we are the first to focus on URL sequence classification and propose a system for detecting malicious URL sequences.

- We identified 13 effective features for classifying URLs accessed by software other than browsers.

- We show that our EDCNN with the 13 identified features and data augmentation achieves practical classification performance by detecting malicious redirections even if some benign URLs exist between their URLs.



(a) Redirection Chain

Timestamp	Source IP Address	Method	URL
00:00:00	192.0.2.1	GET	a.example/index.html
00:00:01	192.0.2.1	GET	a.example/animate.js
00:00:02	192.0.2.1	GET	a.example/img.png
00:00:05	192.0.2.1	GET	b.example/redirect.php
00:00:06	192.0.2.1	GET	b.example/fingerprinting.js
00:00:10	192.0.2.1	GET	mal.example/exploit.html
00:00:11	192.0.2.1	GET	mal.example/malware.exe

(b) Simplified Log

a.example/index.html
a.example/animate.js
a.example/img.png
b.example/redirect.php
b.example/fingerprinting.js

(c) URL Sequence

Figure 3.1: Data formats.

3.2 Data Formats

To give a better understanding of our system, we explain the definitions of data relevant to our system: redirection chains, simplified logs, and URL sequences.

Redirection Chain. Attackers redirect a user's client from a landing URL to an exploit URL via redirection URLs. When a client accesses an exploit URL, code that exploits the vulnerabilities of a browser and its plugins is executed. The code execution forces the client to download and install malware from a malware distribution URL [114]. The relationships of these URLs are shown in Fig. 3.1(a), in which vertexes are URLs and edges are redirections. This tree-shaped architecture is called a redirection chain. The redirection chain of malicious websites has a distinct structural feature, i.e., multistage redirections. We define a malicious URL as a URL along a path from a landing URL to a malware distribution URL and a benign URL as a non-malicious URL.

Simplified Log. Operators of enterprise networks typically record simplified logs, which are lists of information regarding communications such as timestamps, source IP addresses, HTTP methods, and destination URLs as shown in Fig. 3.1(b). They are recorded in the order of communication occurrence. Simplified logs do not show the source and destination of a redirection but do contain malicious URLs (i.e., landing URLs, redirection URLs, and exploit URLs) and preserve their sequential order.

URL Sequence. Our system extracts URL sequences from simplified logs. A URL sequence is a list of URLs selected from a simplified log on the basis of certain criteria. Figure 3.1(c) shows a URL

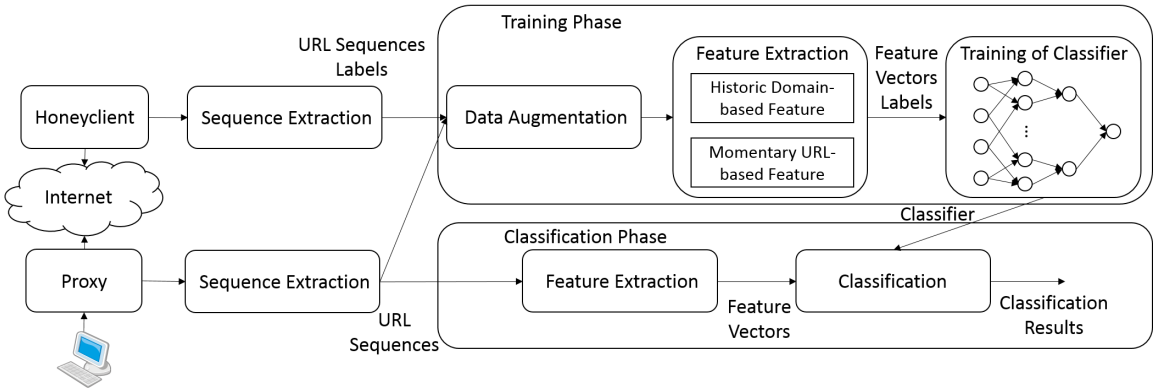


Figure 3.2: Overview of proposed system.

sequence extracted by selecting the first five URLs from the simplified log shown in Fig. 3.1(b). We define a malicious URL sequence as a URL sequence that contains at least one malicious URL and a benign URL sequence as a URL sequence that contains only benign URLs.

3.3 System Design

We now present our proposed system for detecting malicious URL sequences. We first describe our system design in this Section and give details of our system in Section 3.4. Figure 3.2 shows an overview of our system.

Our system receives simplified logs as input and detects malicious URL sequences. Since a large number of accesses to different websites are recorded in a simplified log, we extract parts of simplified logs as URL sequences and classify them. Specifically, we extract a URL sequence so that URLs related to a malicious redirection are included in the same URL sequence.

To classify URL sequences, one possible approach is treating URLs as text and applying methods that are based on natural language processing. However, attackers can easily evade this approach by manipulating characters in URLs. In contrast, methods for detecting malicious URLs are focused on IP addresses corresponding to the domain and the hierarchical structure of the domain [6, 19]. Attackers have trouble registering domains and IP addresses that are securely operated and so tend to register domains and IP addresses that are loosely operated. Therefore, some IP-address ranges

and top level domains are occasionally used for malicious purposes, whereas others are rarely used for such purposes. On the basis of this insight, heuristically designed features are extracted to detect malicious URLs and make it difficult for attackers to evade detection. For this reason, our system extracts these features from URLs and classifies them.

In the training phase, large numbers of malicious and benign URL sequences are required as training data to accurately classify sequences. However, it is generally difficult to collect simplified logs that include accesses to many malicious websites. Instead, we collect malicious URL sequences by recording accesses to known malicious websites with a honeyclient. Benign URL sequences are extracted from proxy logs. In addition, we generate URL sequences that contain malicious URLs that have diverse positional relationships for training the EDCNN. Then feature vectors are extracted from these URL sequences, and a classifier is trained. In the classification phase, URL sequences are extracted from proxy logs. Then feature vectors are extracted from them and are classified by the classifier. Note that HTTPS traffic can be analyzed with a man-in-the-middle proxy deployed in modern enterprise networks.

3.4 Proposed System

We describe components of our proposed system in detail.

3.4.1 URL Sequence Extraction

We extract URL sequences so that URLs related to a malicious redirection are included in the same URL sequence as shown in Fig. 3.3. First, simplified logs are divided on the basis of source IP addresses (①). Divided simplified logs contain accesses from each source IP address. After that, the logs are split between URLs that have intervals longer than a threshold (②). By setting a threshold longer than intervals of malicious redirections, we can obtain desired URL sequences.

3.4.2 Feature Extraction

We use two types of features designed from different viewpoints to achieve high classification performance. Note that our system is not limited to these two features. Any feature can be added or

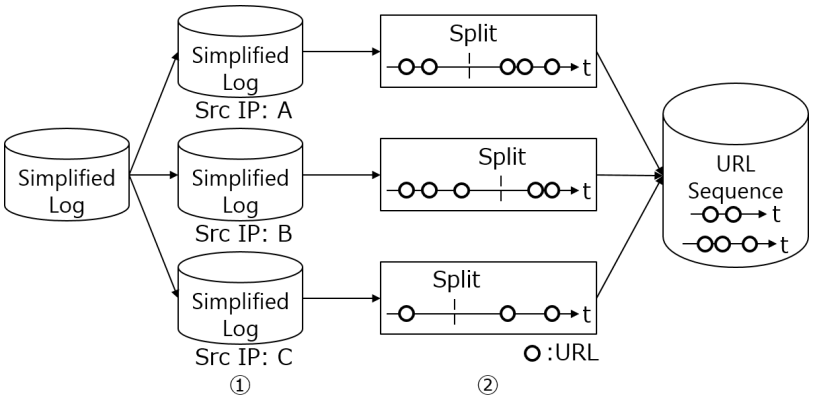


Figure 3.3: URL sequence extraction.

Table 3.1: Historic domain-based features.

No.	RHIP Features	No.	RHDN Features
1	# BGP Prefixes (FQDN)	19	# FQDNs
2	# BGP Prefixes (3LD)	20	Mean (Lengths)
3	# BGP Prefixes (2LD)	21	SD (Lengths)
4	# Countries (FQDN)	22	Mean (1-gram)
5	# Countries (3LD)	23	Median (1-gram)
6	# Countries (2LD)	24	SD (1-gram)
7	# IP addresses (3LD)	25	Mean (2-grams)
8	# IP addresses (2LD)	26	Median (2-grams)
9	# Organizations (FQDN)	27	SD (2-grams)
10	# ASN (FQDN)	28	Mean (3-grams)
11	# ASN (3LD)	29	Median (3-grams)
12	# ASN (2LD)	30	SD (3-grams)
13	# Registries (FQDN)	31	# TLDs
14	# Registries (3LD)	32	Ratio of .com
15	# Registries (2LD)	33	Mean (TLD)
16	# Dates (FQDN)	34	Median (TLD)
17	# Dates (3LD)	35	SD (TLD)
18	# Dates (2LD)		

removed easily.

Historic Domain-Based Features. Historic domain-based features are categorized into two approaches. One is a correspondence-based approach [6], which requires correspondence between domains and IP addresses, between IP addresses and autonomous systems (ASs), and so on. The

other is a behavior-based approach [7, 14], which requires large-cache DNS logs or authoritative DNS logs. However, these DNS logs are generally difficult to obtain. Therefore, from the perspective of ease of deployment, we take the correspondence-based approach. Specifically, we first extract the domain part from each input URL and then extract our historic domain-based features. As with Notos [6], our historic domain-based features come in two types: related historic IP addresses (RHIP) and related historic domain names (RHDN). A RHIP is a union of all resolved IP addresses for each fully qualified domain name (FQDN), its third-level domain (3LD) part, and its second-level domain (2LD) part in the past. We extract 18 RHIP features, Nos. 1–18 in Tab. 3.1: the numbers of the RHIP's Border Gateway Protocol (BGP) prefixes, countries, organizations, AS numbers (ASNs), and registries; allocated dates for the FQDN, 3LD, and 2LD; and the numbers of RHIPs for the 3LD and 2LD. On the other hand, a RHDN is a union of domains resolved to IP addresses in the same ASN of the past IP addresses of each FQDN. We extract 17 RHDN features, Nos. 19–35 in Tab. 3.1: the number, mean length, and standard deviation (SD) of the length of the RHDN's FQDNs. We also consider the occurrence frequency of n-grams of the RHDN's FQDNs: the means, medians, and SDs of 1-gram, 2-grams, and 3-grams. Moreover, we focus on the top-level domain (TLD) of the RHDN's FQDNs: the number of TLDs; ratio of the .com TLDs; and mean, median, and SD of the occurrence frequency of the TLDs in an RHDN.

Momentary URL-based Features. To detect malicious URLs of drive-by download attacks, many momentary URL-based features have been proposed [19]. We selected two feature types in Tab. 3.2 that are not covered by historic domain-based features. The first is the length of a part of a URL (Nos. 1–3): the length of the file name, URL, and path. The second is the presence of a malicious or benign trace in a URL (Nos. 4–8): the presence of a known malicious domain, a known malicious pattern in a URL or file name, a subdomain, and an IP address.

We further add new features for classifying URLs automatically accessed by software other than browsers such as an OS and anti-virus software. These URLs are difficult to identify on the basis of domain-based features [21]. We designed features by focusing on URL paths and queries. Manual analysis revealed distinct features of URLs accessed by software other than browsers: their filenames do not have file extensions, their TLDs are popular, hierarchies of their paths are very

Table 3.2: Momentary URL-based features.

No.	Features	Origin
1	Length of file name	[19]
2	Length of URL	
3	Length of path	
4	Presence of known malicious domain	
5	Presence of known malicious pattern in file name	
6	Presence of known malicious pattern in URL	
7	Presence of subdomain	
8	Presence of IP address	
9	Presence of `.html` or `.php` extensions	Proposed
10	Presence of `.js` extension	
11	Presence of `.pdf` or `.swf` extensions	
12	Alexa rank of TLD	
13	Depth of URL hierarchy	
14	# of queries	
15	# of "-" in queries	
16	# of "_" in queries	
17	# of symbols except "-" and "_" in queries	
18	Ratio of upper-case letters in queries	
19	Ratio of lower-case letters in queries	
20	Ratio of numerical digits in queries	
21	Ratio of symbols in queries	

deep, and characters used in their queries differ depending on software. In terms of file extension, we use the presence of content types frequently used by drive-by download attacks because this feature has more information than the presence of file extension (Nos. 9–11). Specifically, `.html` or `.php` extensions are used in compromised or redirection URLs, a `.js` extension is used in redirection or exploit URLs, and `.pdf` or `.swf` extensions are used in exploit URLs. To rank TLDs in terms of popularity, we use the Alexa rank of TLDs: the highest Alexa [4] rank among domains with a certain TLD (No. 12). From URL paths, we extract the depth of URL paths (No. 13). Regarding queries, we extract the number of queries, the number of hyphens "-", the number of underscores "_", the number of other symbols, ratio of upper-case letters, ratio of lower-case letters, ratio of numerical digits, and ratio of symbols (Nos. 14–21).

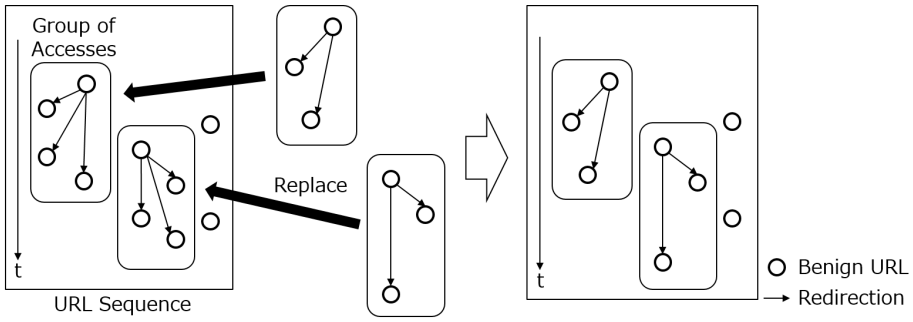


Figure 3.4: Data augmentation for benign URL sequence.

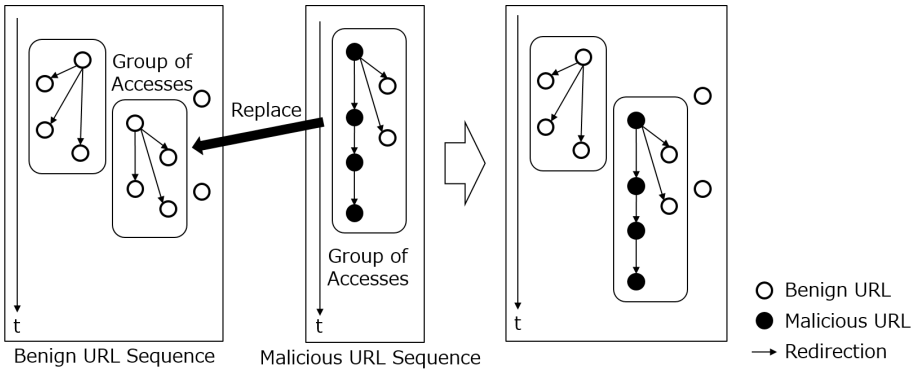


Figure 3.5: Data augmentation for malicious URL sequence.

3.4.3 Data Augmentation

We accelerate training of the EDCNN by generating new URL sequences that include accesses to different websites. A large variety of positional relationships of malicious URLs improves the detection capability of the EDCNN classifier. We also generate new benign URL sequences to shorten the period of collecting proxy logs for training data. However, generated URL sequences might degrade classification performance if the positional relationships of malicious URLs in them are totally different from those in proxy logs. To improve classification performance with data augmentation, we generate URL sequences similar to those extracted from proxy logs in terms of the number of included websites and intervals of accesses.

We make groups of accesses automatically redirected from the same URL by using referers to identify accesses to a website. Referers related to search engines are not used because they are not

automatic redirections. A new benign URL sequence is generated by replacing groups of accesses in a URL sequence containing multiple groups with those in other URL sequences as shown in Fig. 3.4. The intervals between the first accesses of every group and intervals between accesses of the same group are preserved. A new malicious URL sequence is generated by replacing one group in a generated benign URL sequence with one from a malicious URL sequence as shown in Fig. 3.5. A generated malicious URL sequence contains one group of accesses related to malicious websites because a host rarely accesses multiple compromised websites within a short period. Note that a referer is recorded while collecting training data but does not need to be recorded in the classification phase.

3.4.4 Classification

We classify URL sequences using a CNN or our EDCNN and compare them to determine which is more effective.

Notations. We describe the notations used in this Chapter. We define a URL sequence as $x = \{x_i\}_{i=1}^{I}$, where $x_i \in \mathbb{R}^K$ denotes the feature vector of the i-th URL, K denotes the number of features, and I denotes the number of URLs. The classification result of the URL sequence x is defined as $y \in \{0, 1\}$. A classification result of 0 represents benign and 1 represents malicious.

A neural network is composed of multiple layers that gradually extract higher level representations. We define the input and output of the l-th layer as h^{l-1} and h^l, respectively. In the case that h^l is three-dimensional data containing I_l rows, J_l columns, and K_l channels, we use $K_l@I_l \times J_l$ to represent its size, and $h^l_{i,j,k}$ to represent its value at the i-th row, j-th column, and k-th channel. For example, x_i is also represented as three-dimensional data with a size of $K@1 \times 1$ by stacking features along the channel axis. A URL sequence x is represented as three-dimensional data h^1 with a size of $K@I \times 1$ by stacking x_i along the row axis. The value of the k-th feature in the i-th URL is represented as $h^1_{i,1,k}$. Note that the number of columns of h^1 is 1 because no variable is stacked along the column axis. These representations are used to input a URL sequence into a neural network. In the case that h^l is the vector of size I_l, we define the i-th value in the vector as h^l_i. We also define the filter weight of the convolution layer for calculating the c-th channel of h^l as w^{l_c}. The size of the

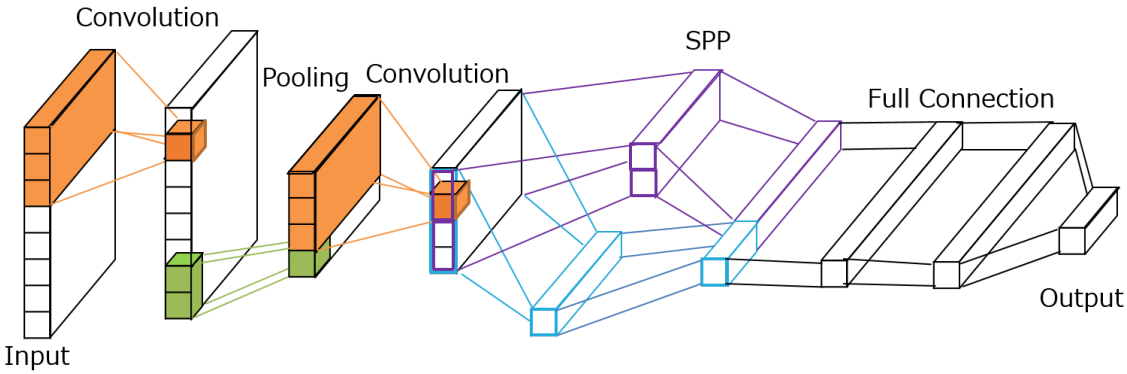


Figure 3.6: Architecture of CNN.

filter weight containing N_l rows and M_l columns is represented as $N_l \times M_l$ and the value at the n-th row and m-th column as $w^{l_c}_{n,m}$. Similarly, we use $N_l \times M_l$ to represent the window size of pooling to calculate h^l. The weight matrix of the fully connected layer for calculating h^l is represented as w^l, and the weight between h^{l-1}_i and h^l_j is represented as w^l_{ij}.

Convolutional Neural Network. Figure 3.6 shows the architecture of the CNN applied to our system. Convolution and pooling are repeated two times to convolute four malicious URLs that are expected to be included in malicious URL sequences. This CNN has eight layers: one input layer, two convolution layers, one max pooling layer, one spatial pyramid pooling (SPP) layer, two fully connected layers, and one output layer. The convolution, max pooling and SPP layers take three-dimensional data as input and output different three-dimensional data. The fully connected and output layers take vectors as input and output other vectors.

Input layer: The input layer arranges our features extracted from URLs to input them into the CNN. As described above, the output of this layer h^1 is represented as the three-dimensional data of $K@I \times 1$. Each feature is normalized so that its average becomes 0 and its variance becomes 1.

Convolution layers: The convolution layers calculate the value of the outputs from those in the receptive field: a small part of input. The output is calculated as $h^l_{i,1,c} = ReLU\left(\sum_{n=0}^{N_l}\sum_{k=1}^{K_{l-1}} w^{l_c}_{n,1,k} h^{l-1}_{i+n,1,k}\right)$. If $h^{l-1}_{i+n,1,k}$ is outside the previous layer, we set $h^{l-1}_{i+n,1,k} = 0$. We expect the redirections between malicious URLs to be extracted in the convolution layers.

Max pooling layer: The max pooling layer outputs the maximum value in the receptive field.

The output is calculated as $h^l_{i,1,k} = \max_{0 \le n \le N_l}(h^{l-1}_{i+n,1,k})$. We expect the max pooling layer to make classification robust against the position of malicious URLs.

Spatial pyramid pooling layer: The SPP layer outputs the fixed length of a vector, receiving an arbitrary size of inputs [39]. In this layer, inputs are divided into 1, 4, ... square areas. Then, the maximum value in each area is selected with respect to each channel. If the number of columns of the input is 1, the input is divided only in the row direction. When the input is divided into 1 and 4 areas, the SPP layer outputs $5K_{l-1}@1 \times 1$ from $K_{l-1}@I_{l-1} \times J_{l-1}, (J_{l-1} \ge 2)$, and outputs $3K_{l-1}@1 \times 1$ from $K_{l-1}@I_{l-1} \times 1$. Note that $K@1 \times 1$ can be calculated as the vector of length K. We expect the SPP layer to propagate information related to malicious redirection and its rough position.

Fully connected layers: The fully connected layers output integrated information considering all values of the previous layer. The output is calculated as $h^l_j = ReLU\left(\sum_i w_{ij} h^{l-1}_i\right)$. We expect the fully connected layers to integrate the information of malicious redirections scattered in the previous layer.

Output layer: The output layer outputs the probability of each class. The dimension of the output vector is the number of classes. The output is calculated as $h^l_j = \frac{e^{z_j}}{\sum_i e^{z_i}}$, where $z_j = \sum_i w_{ij} h^{l-1}_i$. All outputs are non-negative, and the summation of outputs is 1. Therefore, the output represents the probability of its corresponding class. The classification result is determined as $y = \text{argmax}_i h^l_i$.

The weights of every layer are optimized by backpropagation to minimize classification error. The algorithm we use for this optimization is AdaDelta [134], which automatically adapts the learning rate to reduce training time.

Event De-noising Convolutional Neural Network. We develop the EDCNN, which reduces the negative effect of benign URLs in a malicious URL sequence. Specifically, two URLs that are close in sequential order are convoluted at the convolution layers. To implement such a convolution in one layer, we must carry out complicated matrix multiplication. This results in a decrease in the training and classification speed. To tackle this problem, we introduce an *allocation layer*, which arranges values of the input for the intended convolution. Specifically, it arranges values to be convoluted side-by-side, and a conventional convolution layer is applied to its output. In this

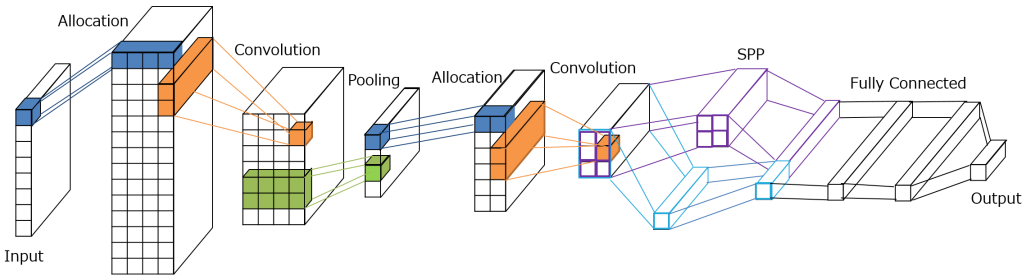


Figure 3.7: Architecture of EDCNN.

way, the intended convolution is implemented using only simple matrix multiplication and carried out without sacrificing training and classification speed. Figure 3.7 shows the architecture of the EDCNN. It has ten layers: one input layer, two allocation layers, two convolution layers, one max pooling layer, one SPP layer, two fully connected layers, and one output layer. We explain the *allocation layer* and layers that we modified: convolution and max pooling layers.

Allocation layers: To convolute two URLs that are close in sequential order in the convolution layer, we arrange values of the input so that the i-th and $i+1$-th values, i-th and $i+2$-th values, ..., and i and $i+N_l$-th values are adjacent. The output is calculated as

$$h^l_{2i,j,k} = \begin{cases} h^{l-1}_{i,1,k} & (i+j \leq I_{l-1}) \\ 0 & (otherwise) \end{cases} \tag{3.1}$$

$$h^l_{2i+1,j,k} = \begin{cases} h^{l-1}_{i+j,1,k} & (i+j \leq I_{l-1}) \\ 0 & (otherwise), \end{cases} \tag{3.2}$$

where $i \in \{a \in \mathbb{Z} | 1 \leq a \leq I_{l-1}\}$ and $j \in \{a \in \mathbb{Z} | 1 \leq a \leq N_l\}$.

Convolution layers: The convolution layers convolute the adjacent values arranged in the allocation layers. The output is calculated as $h^l_{i,j,c} = ReLU\left(\sum_{h=0}^{1}\sum_{k=1}^{K_{l-1}} w^c_{h,1,k} h^{l-1}_{2i+h,j,k}\right)$. The receptive field is moved by two in the row direction.

Max pooling layer: The values of the inputs in the same column are the convolution of candidates of malicious redirections. One of them may be the convolution of actual malicious redirection. Thus, it is reasonable to select one value from them. The output is calculated as $h^l_{i,1,k} = \max_{1 \leq j \leq J_{l-1}}(h^{l-1}_{i,j,k})$.

The weights are optimized by backpropagation to minimize classification error. Errors in the allocation layers are propagated backward to the previous layer through connections. Propagated errors are summed in the previous layer.

3.5 Evaluation

We evaluate the effectiveness of our URL sequence extraction and classification. We first explain the data collection and then report the evaluation results.

3.5.1 Data Collection

We collect two types of logs for our evaluation: proxy logs and honeyclient logs.

Proxy Log. For a typical simplified log, we use proxy logs for evaluation. We collect proxy logs from 26th Oct. to 25th Dec. 2016 from a university's network with the users ' consent. Source IP addresses are anonymized while recording the proxy log, but the same anonymized numbers are assigned to the same source IP address. Accesses to 824,950 URLs are collected in the proxy log.

Honeyclient Log. We use honeyclient logs to collect accesses to malicious websites. We collect honeyclient logs by accessing public blacklists [68, 128] with the honeyclient [3] from 26th May to 25th Dec. 2016. We use them if attacks are detected by the honeyclient. Accesses to 105,528 URLs are collected in the honeyclient logs.

3.5.2 Evaluation of URL Sequence Extraction

We evaluate whether our system can extract URL sequences adequate for classification.

Threshold Setting. Before extracting URL sequences, the threshold for splitting proxy logs must be decided. To make sure that URLs related to a malicious redirection are included in the same URL sequence, we set the threshold larger than intervals of accesses included in malicious URL sequences. Figure 3.8 shows the cumulative distribution function (CDF) of intervals of accesses in malicious URL sequences. Most intervals are shorter than 20 seconds, but we use 60 seconds as the threshold since it is sufficiently longer than most intervals.

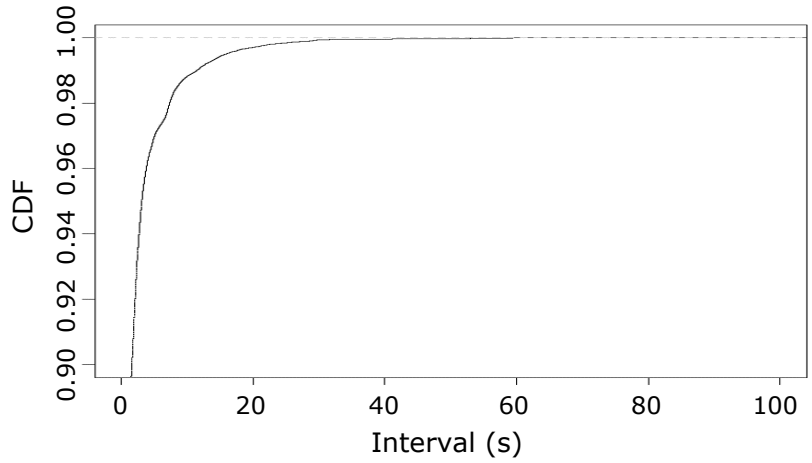


Figure 3.8: CDF of intervals of accesses.

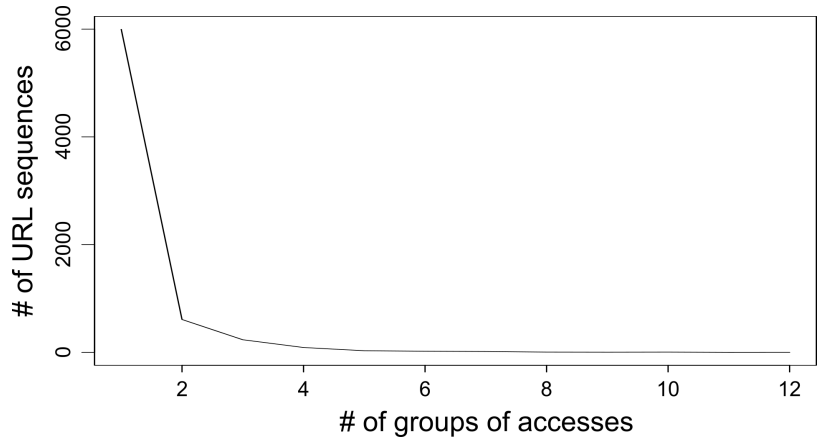


Figure 3.9: Distribution of the number of groups included in a URL sequence.

Analysis of URL Sequence. We analyze the extracted URL sequences to evaluate whether they are adequate for classification. URL sequences are difficult to classify accurately if they contain accesses related to multiple websites. We investigate the number of websites included in a URL sequence. We make groups of accesses automatically redirected from a URL in the same manner as data augmentation. The numbers of groups in the extracted URL sequences are shown in Fig. 3.9. Because most URL sequences include only one group, the extracted URL sequences are speculated to be adequate for classification.

Table 3.3: Dataset. DA represents data generated with the data augmentation.

	Label	Number	Number (DA)	Period
Training	Malicious	3,224	-	26th May 2016 – 25th Nov. 2016
	Benign	5,061	6,641	26th Oct. 2016 – 25th Nov. 2016
	Replaced	2,153	11,154	-
Test	Malicious	462	-	26th Nov. 2016 – 25th Dec. 2016
	Benign	7,044	-	26th Nov. 2016 – 25th Dec. 2016
	Replaced	3,050	-	-

3.5.3 Evaluation of Classification

We evaluate the classification performance. We describe the dataset and hyperparameter optimization and then show the evaluation results for classification performance and calculation time.

Dataset for Classification. We use URL sequences extracted from the proxy log to evaluate classification performance for sequences containing accesses to only benign websites. This is because anti-virus software is installed onto all hosts in the network where the proxy log is collected, but infection is not detected while collecting the proxy log. We use URL sequences extracted from the honeyclient logs to evaluate classification performance for those containing accesses to only malicious websites. To evaluate classification performance for URL sequences containing accesses to benign and malicious websites, we generate URL sequences by using the proxy log and honeyclient logs. The validity of evaluation depends on the similarity between URL sequences generated for evaluation and those when users access malicious websites. We generate URL sequences by replacing one group of accesses in a benign URL sequence with one from a randomly selected malicious URL sequence as shown in Fig. 3.5. The generated URL sequences are not expected to greatly differ from those of users' accesses to malicious websites for two reasons: 1) we can randomly select malicious groups of accesses because benign and malicious URLs have little relation, and 2) only one group of accesses in a URL sequence is replaced.

We use data collected before 25th Nov. 2016 as training data and data collected after as test data. No URL sequences between training and test data overlaps. In total, 28,621 URL sequences are used for training data and 10,556 for test data, as shown in Tab. 3.3. We define URL sequences extracted from the proxy log as benign URL sequences, URL sequences from honeyclient logs as

Table 3.4: Selected hyperparameters.

Hyperparameter	CNN	EDCNN
# of channels in 1st convolution layer	100	100
# of channels in 2nd convolution layer	100	100
# of rows in receptive field for convolution	3	3
# of neurons in fully connected layers	1000	1000

malicious URL sequences (M), and URL sequences generated using the proxy log and honeyclient logs as replaced URL sequences (R). Since replaced URL sequences include accesses to malicious websites, they should be detected by our system.

Conventional Approach. Malicious websites have been typically detected with a network IDS by individually classifying destination URLs. If one or more URLs are detected, the corresponding website is considered malicious [19]. We compare our approach with such a conventional individual-based approach. In this individual approach, URLs in a sequence are individually classified, and the URL sequence is classified as malicious if at least one URL is classified as such. Let x_i denote the feature vector of the i-th URL and $f(x_i) \in \{1, 0\}$ denote the classification results of that URL. A classification result of 1 represents malicious, and 0 represents benign. The classification result y of the URL sequence $x = \{x_1, ..., x_I\}$ is defined as $y = \max_{1 \leq i \leq I}\{f(x_i)\}$. We use random forest as the classifier. Random forest builds multiple decision tree classifiers that use parts of features. Its classification result is determined by a majority vote of classification results of decision tree classifiers.

Hyperparameter Optimization. Before training the classifier, we have to select its hyperparameters. The hyperparameters for the individual-based approach are the ratio of features used for each decision tree and the number of decision trees. They are optimized by using the `tuneRF` function of the `randomForest` package in R [44] when a classifier is trained. The hyperparameters for the CNN and EDCNN are the numbers of channels in convolution layers, rows in the receptive field for convolution, and neurons in fully connected layers. The hyperparameters are selected on the basis of a preliminary cross-validation as shown in Tab. 3.4.

Classification Performance. We now describe the evaluation results for classification performance

Table 3.5: Classification performance.

Classifier	TPR (M)	TPR (R)	FPR
Individual (HD)	47.5%	57.1%	9.0%
Individual (MUC)	88.3%	90.7%	6.9%
Individual (MUP)	53.4%	68.8%	13.2%
Individual (HD+MUC)	38.4%	43.1%	**2.2%**
Individual (HD+MUP)	27.5%	31.6%	2.9%
Individual (MUC+MUP)	56.6%	69.2%	11.1%
Individual (HD+MUC+MUP)	91.1%	91.8%	3.9%
CNN	98.5%	90.0%	7.3%
EDCNN (without DA)	97.0%	87.6%	3.1%
EDCNN (with DA)	**99.1%**	**95.0%**	3.4%

of unknown malicious URL sequences. We evaluate classification performance with two widely used indexes: TPR and FPR. These indexes are defined by using the numbers of the following URL sequences: TPs, which are correctly classified malicious URL sequences by the classifier; FNs, which are wrongly classified malicious URL sequences; TNs, which are correctly classified benign URL sequences; and FPs, which are wrongly classified benign URL sequences. Each index is defined as follows: $TPR = TPs/(TPs + FNs), FPR = FPs/(FPs + TNs)$.

We compare classification performances of three approaches: individual, CNN, and EDCNN. We also evaluate the effectiveness of features: historic domain-based features (HD), momentary URL-based features proposed in the conventional system (MUC), and momentary URL-based features proposed in our system (MUP). For this purpose, we train the individual approach by using all combinations of HD, MUC, and MUP. Note that the CNN and the EDCNN are trained by using all features. We further evaluate the effectiveness of data augmentation for the EDCNN by comparing the classification performances of the EDCNN with and without data augmentation.

Table 3.5 shows TPR and FPR of the classifiers. Results for the individual approach show that each type of features differently affects classification performance. HD is assumed to lower FPR because FPRs are low when HD, HD+MUC, or HD+MUP are used. MUC is assumed to increase TPR because TPRs are high when MUC, HD+MUC, or MUC+MUP are used. A clear tendency is not observed regarding MUP, but MUP is assumed to compliment HD and MUC because the classification performance with HD+MUC+MUP is higher than that with HD+MUC.

Table 3.6: Top 10 features of high *importance*.

No.	Type	Origin	Features
1	Domain	[6]	# Countries (3LD)
2	URL	Proposed	Alexa rank of TLD
3	Domain	[6]	# IP addresses (3LD)
4	Domain	[6]	SD (Lengths)
5	URL	[19]	Presence of subdomain
6	Domain	[6]	# Dates (2LD)
7	URL	Proposed	Ratio of capital English letters in queries
8	Domain	[6]	Ratio of .com
9	URL	[19]	Length of URL
10	Domain	[6]	# BGP Prefixes (2LD)

The classifiers based on relationships of URLs, i.e., CNN and EDCNN, achieve high TPR for URL sequences containing one group of accesses but comparatively low TPR for URL sequences containing multiple groups of accesses. This is because features of malicious redirections are difficult to extract from sequences that include accesses to multiple websites. Even under this condition, classification performance of the EDCNN is improved by using data augmentation. Out of the ten classifiers, the EDCNN with data augmentation achieves the most practical classification performance, i.e., high TPR and acceptable FPR.

To determine the contributions of the proposed features, we calculate the *importance* of the features which represents how much each feature contributes to classification. *Importance* is calculated with the `importance` function in the `randomForest` package of R [44]. The top 10 features of high *importance* are shown in Tab. 3.6, and all types of features are in the top 10. It also shows that the reason classification performance improves by adding our proposed features to conventional ones is that our features enable a classifier to execute classification on the basis of more distinctive features.

To better understand the classifiers, we compare the individual-based approach using all features, CNN, and EDCNN with data augmentation. Specifically, we analyze classification performance on different numbers of groups of accesses and URLs. TPRs and FPRs for different numbers of groups of accesses are shown in Fig. 3.10. When the number of groups is the same, the classifiers have similar FPRs but different TPRs. When the number of groups is small, the EDCNN and the CNN

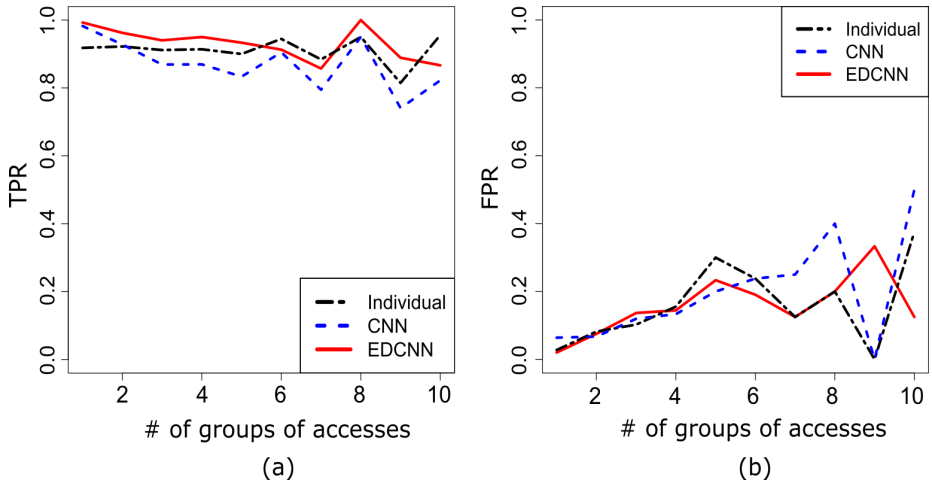


Figure 3.10: TPR and FPR for different numbers of groups of accesses.

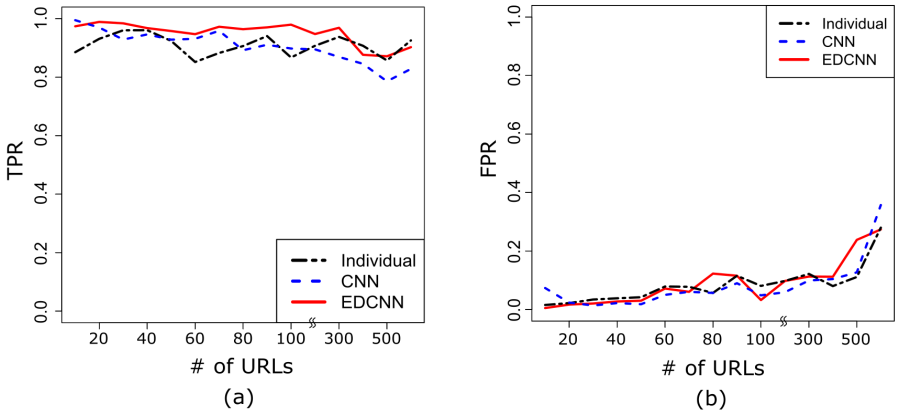


Figure 3.11: TPR and FPR for different numbers of URLs.

have high TPRs, but the individual-based approach has a low TRP. When the number of groups is large, the EDCNN and individual-based approach have high TPRs, but the CNN has a low TPR. Only the EDCNN can maintain high TPR regardless of the number of groups. TPRs and FPRs for different numbers of URLs are shown in Fig. 3.11. When the number of URLs is the same, the classifiers have similar FPRs but different TPRs. When the number of URLs is less than 300, the EDCNN has the highest TRP, and the CNN and individual-based approach have similar TRPs. When the number of URLs is 300 or more, the EDCNN and individual-based approach have similar TRPs that are higher than that for the CNN. These results show the EDCNN outperforms the CNN

Table 3.7: Calculation time.

Feature	Extraction (sec./seq.)	Classifier	Training (hours)	Classification (sec./seq.)
Domain	0.350	Individual	0.02	0.002
URL	0.007	CNN	47.19	0.086
Proposed	0.003	EDCNN	52.49	0.096

and the individual-based approach regardless of the number of groups and URLs in URL sequences.

Calculation Time. The maximum size of proxy logs to which our system can be applied depends on the calculation time of feature extraction and classification. Furthermore, a classifier needs to be periodically retrained to continuously maintain high classification performance even if features of benign and malicious sequences change over time. The calculation time of training needs to be shorter than a period when a classifier maintains high classification performance. Therefore, we evaluate the calculation time of feature extraction, training, and classification. Table 3.7 lists the calculation times of a prototype version of our system. Our prototype is implemented using two different machines. Momentary URL-based features were extracted and training/classification is conducted by using a Ubuntu server with four 12-core CPUs and 128-GB RAM, and historic domain-based features are extracted with a Hadoop cluster composed of 2 master servers with a 16-core CPU and 128-GB RAM and 16 slave servers with a 16-core CPU and 64-GB RAM. The individual-based approach is implemented by using the `randomForest` package of R, and the CNN and EDCNN are implemented by using Chainer [123]. The total calculation time of feature extraction and classification is about 0.5 seconds per sequence. The prototype system could inspect 189,473 URL sequences a day with the EDCNN. The calculation times of the CNN and EDCNN training are longer than that of the individual-based approach. In the evaluation, the CNN and EDCNN achieve high classification performance using data collected for one month. In other words, they maintained high classification performance for at last one month. The calculation times of training the CNN and EDCNN are sufficiently shorter than one month when their classifiers maintained high classification performance.

3.5.4 Threats to Validity

We further analyze evaluation results to confirm the validity of our evaluation.

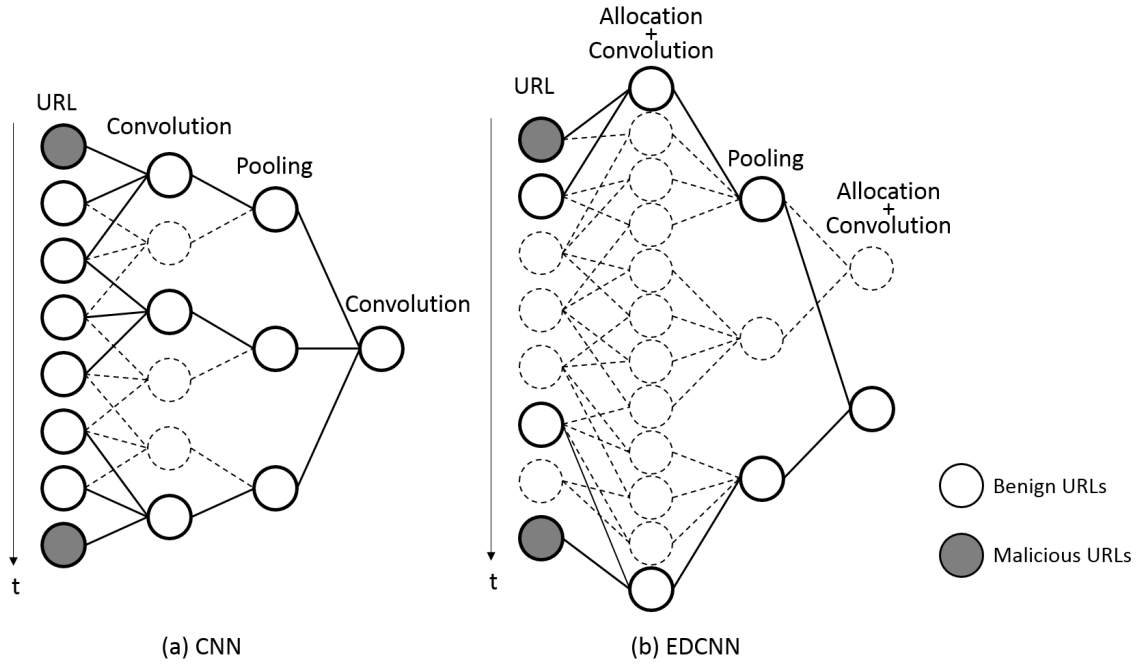


Figure 3.12: Convolution and pooling of (a) CNN and (b) EDCNN.

URL Sequences Detected only with EDCNN. We confirm that the structure of the EDCNN improves classification performance by analyzing URL sequences detected only with the EDCNN. In these sequences, URLs related to malicious websites are split into several parts and have about five benign URLs between them. The convolution and pooling of the EDCNN are assumed to enable the EDCNN to detect these URL sequences. Figure 3.12 shows an example of convolution and pooling with the CNN and the EDCNN. URLs/neurons illustrated with solid lines are those utilized for calculating the output of the neuron in the second convolution layer. Other URLs/neurons are illustrated with broken lines. Many benign URLs are utilized for calculation in the CNN because all neurons in the receptive field are convoluted. This means that the CNN must learn features of not only malicious URLs but also benign URLs between them. In the EDCNN, malicious URLs and two benign URLs are utilized for calculation. This means that the EDCNN can ignore some benign URLs between malicious ones. This is why the EDCNN could detect malicious URL sequences if about five benign URLs are located between malicious URLs.

Effect of Data Augmentation. We investigate the effect of data augmentation by evaluating classification performance of the CNN with data augmentation. The results are TPR(M) = 96.7%, TPR(R) = 88.5%, and FPR = 5.1%. The classification performance of the CNN is not improved by data augmentation. The difference in the effect of data augmentation is assumed to derive from the difference in the convolution and pooling. As discussed above, all neurons in the receptive field are convoluted in the CNN. Not only malicious URLs but also benign URLs affect classification results. When new URL sequences are generated, all groups of accesses are randomly selected. Consequently, some generated URL sequences are assumed to contain combinations of benign groups dissimilar to combinations in the original benign URL sequences. Although malicious and benign groups of accesses have little relation, benign groups of accesses are related to each other. For example, users are likely to access websites on similar topics in a short period. Therefore, some generated URL sequences contain strange sequences of benign URLs, and the CNN learns them. This is why the classification performance of the CNN is not improved by data augmentation. To improve classification performance of the CNN with data augmentation, we must generate new URL sequences by considering what type of benign groups are likely to be included in the same URL sequences. However, this is much more difficult because we need to simulate users' interests and browsing patterns. Note that our data augmentation does not affect the classification results of the individual-based approach because it does not change any feature vectors of individual URLs. This is why we do not evaluate the classification performance of the individual-based approach with the data augmentation.

False Positives and False Negatives. We investigate FPs and FNs of the EDCNN to confirm that they are not critical. FPs occur if URL sequences successively include some URLs slightly similar to compromised or malicious ones. These URLs are falsely detected as malicious redirections. In addition, FPs still occur if some apparently benign URLs are located between URLs similar to compromised or malicious ones because of the EDCNN's advantage that it ignores some URLs. FNs occur if 10 – 20 benign URLs are located between malicious URLs. The EDCNN can not convolute these malicious URLs because they are not in the same receptive field of convolution.

3.6 Discussion

In the evaluation of classification, we should ideally evaluate whether our proposed system can detect malicious URL sequences extracted from the proxy log because such detection is the main purpose of our system. However, infection is not detected while collecting the proxy log. Thus, we cannot evaluate whether our system can detect malicious URL sequences by using the proxy log. From the analysis in Section 3.5.2, we find that most URL sequences extracted from the proxy log contain one group of accesses. Similarly, a URL sequence extracted from a honeyclient log contains one group of accesses because URLs automatically redirected from a landing URL are recorded in a honeyclient log. On the basis of these facts, we can adequately evaluate classification performance in a serious situation by using URL sequences extracted from honeyclient logs. To evaluate classification performance for URL sequences containing accesses to benign and malicious websites, we carefully generate URL sequences by using the proxy log and honeyclient logs without degrading the validity of evaluation.

The FPR of our system might sound high. However, our system can be utilized as a lightweight filter due to its fast calculation time. It quickly reduces the number of URLs precisely analyzed with honeyclients [3, 65] or a web-content-based system [24]. The honeyclient [3] would take 37 days to analyze all URLs collected over 30 days in Section 3.5, so the honeyclient cannot analyze all URLs within the duration of collection. The honeyclient would take eight days to analyze URLs in URL sequences detected with our system. Since the duration of analysis is sufficiently shorter than the duration of collection, the FPR of our system is low enough to be practical.

Our system cannot detect malicious redirections if source and destination of malicious redirections are included in different URL sequences by setting their intervals to more than 60 seconds. However, users leave Web pages within 10 – 20 seconds while browsing the web [81]. If attackers increase intervals between the source and destination of malicious redirections, attacks have more difficulty succeeding because the possibility that users will leave the pages is increased. Therefore, our system is difficult for attackers to evade.

3.7 Related Work

3.7.1 Malicious Website Detection

Two types of honeyclients have been developed to crawl and detect malicious websites. High-interaction honeyclients use real vulnerable browsers and detect malicious websites by monitoring events in the kernel layer [3,65]. Low-interaction honeyclients use browser emulators and detect malicious websites by using the signature of web content and various heuristics [24]. Researchers also focused on web content and redirections. Prophiler [19] detects suspicious websites using HTML-, JavaScript-, and URL-based features as a lightweight filter for malicious websites. Stringhini et al. proposed a method of detecting malicious websites by aggregating different redirection chains of the same landing URLs obtained from a large number of users [114]. However, the above methods store and analyze an enormous amount of web content. Our system can use only URL sequences to detect malicious URLs that should be further examined.

Other studies have focused on the domain or URLs of malicious websites. Antonakakis et al. proposed Notos, which is focused on historic usage of domains and their corresponding IP addresses to detect unevaluated malicious domains [6]. Ma et al. used the lexical structure of phishing website URLs to detect malicious websites [66]. Our system is focused on a wider area than domains or single phishing URLs, i.e., we use URL sequences and the latest learning approaches on the basis of the inherent characteristics of the drive-by download attacks.

3.7.2 Deep Neural Network

Deep neural networks have been successfully applied in many areas, and CNNs have been applied to sequential data such as natural language matching [43]. In this Chapter, we applied a CNN to URL sequences. Benign URLs exist between malicious URLs in a URL sequence. Such benign URLs have a negative effect on URL sequence classification. Therefore, we develop the EDCNN to reduce the negative effect of benign URLs, making it the first CNN designed for security.

A recurrent neural network (RNN) is another class of neural network that has recurrent structures. In the field of computer security, Shin et al. used an RNN to recognize the functions in binaries [108].

We apply a CNN to malicious URL detection because its structure can be flexibly modified on the basis of the characteristics of URL sequences unlike that of a RNN.

3.8 Summary

We propose a system for detecting malicious URL sequences by using the information recorded in proxy logs. Our proposed system is based on three key ideas: focusing on sequences of URLs that include artifacts of malicious redirections, designing new features related to software other than browsers, and generating new training data with data augmentation. We compare three approaches for classification of URL sequences: an individual-based approach, convolutional neural network (CNN), and our event de-noising CNN (EDCNN). We develop the EDCNN to reduce the negative effects of benign URLs included in malicious URL sequences. Specifically, we extend the architecture of a CNN to extract features from *two URLs* in the vicinity. Only our EDCNN with our proposed features and data augmentation achieves a practical classification performance: a true positive rate of 99.1%, and a false positive rate of 3.4%. Further analysis shows that the EDCNN outperforms the individual-based approach and the CNN regardless of the number of groups of accesses and the number of URLs.

Chapter 4

Efficient Dynamic Malware Analysis for Collecting HTTP Requests using Deep Learning

4.1 Introduction

Malware authors or attackers always try to evade detection methods to increase the number of malware-infected hosts on the Internet. The detection methods are broadly divided into three types: static feature-, host-, and network-based. Static feature-based methods, such as general anti-virus engines, are easily evaded by changing malware samples' code structure with packing techniques [133]. Host-based methods, such as API call mining [93], are evaded by blending malicious API calls into legitimate system processes with API hooking or dynamic link library (DLL) injection [32].

This arms race regarding static feature-based and host-based methods increases the importance of network-based methods such as malicious communication detection [22, 77] and blacklist-based detection. These methods are difficult to evade because malicious network behavior is definitely observed. For example, attackers need to coordinate infected hosts to accomplish their mission by distributing configuration files or sending commands from C&C servers. Network-based methods

detect communications sent from infected hosts on the basis of characteristic patterns in HTTP or HTTPS requests collected with dynamic malware analysis. Note that we consider both HTTP and HTTPS requests but hereafter refer to both as "HTTP requests" for simplicity.

Since attackers modify a part of a program of malware samples or produce totally new ones, patterns of malicious HTTP requests change continuously over time. As a result, the detection rate of network-based methods gradually degrades. To maintain a high detection rate, novel HTTP requests, which have not been collected in past analyses, are collected by analyzing new malware samples typically for a *fixed short period* such as five minutes [31]. Ideally, all malware samples should be analyzed for a *long period* to collect more novel HTTP requests. If more characteristic patterns are identified by using collected requests, they increase the detection rate and make it even more difficult for attackers to evade network-based methods. However, more than 350 million new malware samples were detected in 2016 [118], and analyzing all new malware samples for a long period is obviously infeasible in a limited amount of time. Efficient dynamic analysis is thus required to collect more novel HTTP requests in a shorter analysis time.

The efficiency of dynamic analysis can be enhanced by prioritizing analyses of malware samples that send novel HTTP requests. Identifying such malware samples before analyzing them is difficult because malware samples are obfuscated with packing techniques [133]. Therefore, we propose a system that analyzes a malware sample for a short period and then determines whether the analysis should be continued or suspended. Our system leverages only network behavior, i.e., communications, for determination because host behavior might be concealed with API hooking or DLL injection [32].

As determination methods, there are two possible approaches referring to conventional signature-based detection leveraged by anti-virus engines and API call mining. One is based on the presence of a certain communication, and the other is based on network behavior modeling. For the first approach, the presence of novel HTTP requests is assumed to be an adequate criterion. Specifically, an analysis is continued if one or more novel HTTP requests are collected in the short-period analysis. This approach is based on the intuition that such malware samples are likely to continuously send other novel requests. However, some malware samples stop their malicious activities due to failed communications with C&C servers. After that point, they no longer communicate with any hosts.

Furthermore, the same malware sample might send different HTTP requests after a certain point of time if the secondary malware samples, which are downloaded by the original one, behave differently. In this case, an accurate determination cannot be made on the basis of communications sent in the short-period analysis. From these two points, the accurate determination is difficult to make on the basis of the first approach.

In the second approach based on network behavior modeling, we determine whether the analysis should be continued or suspended on the basis of network behavior in the short-period analysis. However, it is difficult to accurately predict whether novel HTTP requests will be collected by continuing the analysis. This is because this prediction requires information about attackers' attempts, such as attacks launched by malware samples, or infrastructure, such as the configuration of C&C servers. Inspired by the method optimizing the approximate loss function [89], we tackle this problem by relaxing the condition where the analysis is continued. Specifically, our system predicts the probability that a malware sample will send HTTP requests not collected in the short-period analysis and continues the analysis if the probability is high. Such prediction can be made by modeling continuous malicious activities such as secondary malware downloads and communications with C&C servers. Our system can collect many HTTP requests, which are expected to include not only ones previously collected in the past analyses but also many novel ones.

To make an accurate prediction, we focus on the fact that malware communications resemble natural language from the viewpoint of data structure. Natural language has a recursive structure; a phrase, e.g., noun phrase, consists of several words, and a sentence consists of several phrases. Similarly, malware communications have a recursive structure. A malicious activity, e.g., secondary malware downloads and communications with C&C servers, consists of several communications, e.g., DNS queries and HTTP requests. Functions of malware samples, e.g., information leakage and attack to other hosts, consist of several malicious activities. To capture such a recursive structure in malware communications, methods for natural language processing (NLP) are expected to be effective.

Many methods have been proposed for NLP: recursive neural network (RNN) [111], long short-term memory [116], and combination of word2vec and a convolutional neural network [53]. All methods can be applied to our proposed system, but we evaluated our system with a method

expected to achieve high classification performance. In natural language, words comprising a phrase are adjacent to each other, but in malware communications, communications consisting of a malicious activity are not necessarily adjacent. For example, a communication for testing Internet connection can be sent between communications with a C&C server when a malware sample sends communications for periodically testing Internet connection. This makes classification difficult because methods for NLP prioritize closer words. However, the RNN [111] is expected to be unaffected by this challenge. The RNN performs classification on the basis of inferred recursive structure: a tree-structured neural network. If we construct tree-structured neural networks considering the recursive structure of malware communications, the RNN can accurately classify malware communications. Therefore, we apply the RNN to our proposed system and empirically discuss its efficiency at collecting novel HTTP requests using 42,856 malware samples collected over six months.

The main contributions of this Chapter are as follows.

- We propose a system that identifies malware samples whose analyses should be continued on the basis of the network behavior in their short-period analyses. In our evaluation, we show that our system can efficiently collect novel HTTP requests in a limited amount of time by keeping the number of malware samples that are analyzed for long period to a minimum.

- To the best of our knowledge, we are the first to apply the RNN to malware communication analysis and to show that it can effectively capture the characteristics of malware communications.

4.2 Recursive Neural Network

The RNN is a tree-structured neural network. It is used for parsing natural language sentences [110] and sentiment analysis [111] in the field of NLP. Our proposed system uses the recursive neural tensor network (RNTN) [111], which improves on the performance of the RNN by using a tensor. The tensor enables the RNTN to calculate high-order composition of input features. The RNTN

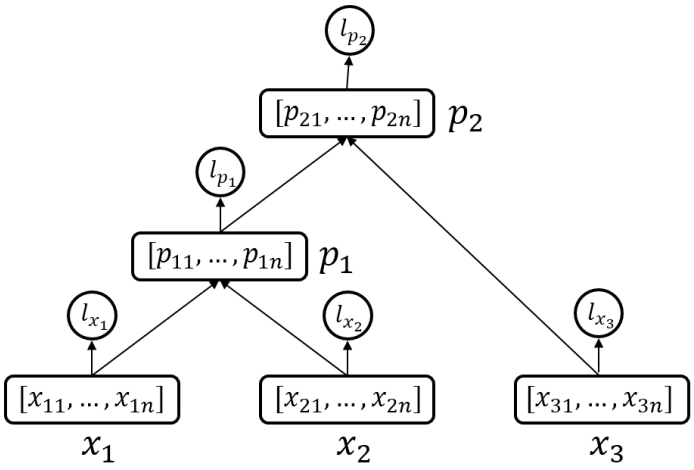


Figure 4.1: Recursive neural tensor network.

is a tree-structured network similar to the RNN, as shown in Fig. 4.1. When the input sequence $x_1, x_2, x_3, \ldots$ is given, these inputs are assigned to leaf nodes in sequence.

Each node has an n-dimensional feature vector and a label. The feature vectors of parent nodes are calculated using the feature vectors of their child nodes. For example, the feature vector of p_1 is calculated using the feature vectors x_1 and x_2:

$$p_1 = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T V^{[1:n]} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + W \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \tag{4.1}$$

where $V^{[1:n]} \in \mathbb{R}^{2n \times 2n \times n}$ and $W \in \mathbb{R}^{n \times 2n}$. The label l_{x_1}, which is the label of node x_1, is calculated as

$$l_{x_1} = \text{softmax}(W_s x_1), \tag{4.2}$$

where $W_s \in \mathbb{R}^{d \times n}$ and d is the number of labels. Tensor V and matrices W and W_s are commonly used in all nodes. The objective with this method is to estimate these variables. The extension of backpropagation on the basis of the prediction error of labels is applied to this process.

The feature vectors of parent nodes have two characteristics. The first is that they are calculated on the basis of the sequence of words. The second is that they represent the semantics of phrases.

This enables phrases to be found that have similar meanings by finding similar feature vectors. In other words, the RNN can infer the same semantics of phrases composed of different words.

4.3 Proposed System

In this section, we first discuss the design of our proposed system and then describe its implementation.

4.3.1 System Design

To improve the efficiency of dynamic analysis, systems based on static features [80] and host behavior [12] have been proposed. However, no system based on network behavior has been proposed, but similar systems have, e.g., malware detection or classification systems based on network behavior. In system design, even if no system has been proposed for the same purpose, referring to systems proposed for a similar purpose is beneficial. The similar systems are divided into network-signature-based [22, 77], correlation-based [36, 37] and statistics-based [74, 76] systems. Network-signature-based systems detect bots by signature matching. Correlation-based systems detect botnets on the basis of the correlation of network behavior such as communications with C&C servers and attacks by infected hosts. Nevertheless, it is assumed with these systems that network behavior is collected over a long period. In particular, with network-signature-based systems, the probability of signature matching decreases as the period of network-behavior collection shortens. Analogously, with correlation-based systems, the number of communications with C&C servers and attacks decreases. Therefore, it is difficult to determine whether dynamic analysis should be continued by using these systems. On the other hand, statistics-based systems extract statistical features from not only communications with C&C servers and attacks but also other communications and classify them with machine learning. For this reason, such systems are more suitable for classification based on a short period of network-behavior collection than other systems. Therefore, we apply machine learning to our proposed system.

The conventional statistics-based system [74, 76] extracts statistical features such as the number of communications for each application protocol and n-grams of network events. The effectiveness

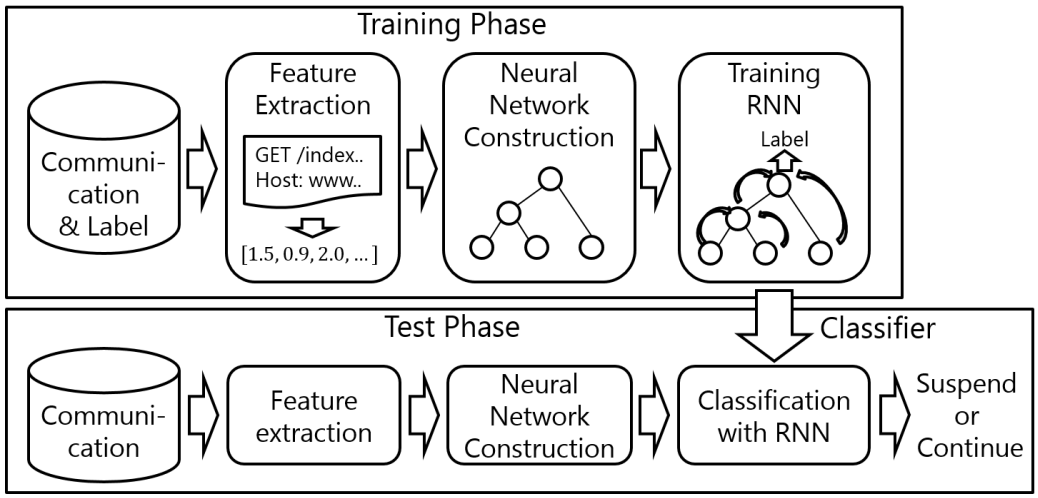


Figure 4.2: Overview of proposed system.

of these features also depends on the number of communications sent in the short-period analysis. If only a few communications are sent, the features will not be meaningful. To overcome this problem, we focus on the change in communication purposes. For example, a malware sample accesses a popular web page for testing the Internet connection, receives the command from the attacker, and then conducts a secondary attack. To this end, we use the sequence of communications to model malicious activities and apply the RNN to our proposed system.

Figure 4.2 shows the overview of our proposed system. In the training phase, our system receives pairs of malware communications and a label as input. Malware communications are collected by analyzing malware samples for a short period. The label indicates whether the analysis should be continued or suspended. In the end of the training phase, our system outputs the trained classifier of the RNN. In the test phase, our system receives new malware communications as input and outputs the determination of whether the analysis should be continued or suspended. The communications are collected by analyzing new malware samples for a short period. In both phases, a feature vector is extracted from each communication, and the neural network is constructed on the basis of communication purposes. The feature vectors and neural network are used for training and classification. We describe details of these components in the following subsections.

Table 4.1: List of features.

Type	No.	Feature	Reference
General	1	Protocol	[74, 76]
	2	Elapsed time	[58]
	3	Interval	[58]
	4	Existence of the identical communication	[58]
	5	Data size of request	[74]
	6	Data size of response	[74]
	7	Port number	[19, 74]
Hostname	8	TLD rank	[19]
	9	Presence of "ip" in FQDN	[19, 66]
	10	Presence of IP address	[19, 66]
	11	Presence of subdomain	[19]
HTTP	12	HTTP method	[77]
	13	Status code	[74]
	14	Presence of `.exe` in filename	[77]
	15	Presence of `.zip` in filename	[77]
	16	Depth of file path	[77]
	17	# of query parameters	[77]
	18	User agent	[58, 77]

4.3.2 Feature Extraction

The first step involves extracting features from communications collected with dynamic analysis. To achieve high classification performance, we must extract features representing a part of malicious activities, e.g., Internet connection tests, updates, and command reception. To this end, we extract 7 general features, 4 hostname-based features, and 7 HTTP-based features, totaling 18 features as shown in Table 4.1. We design these features by referring to not only the statistics-based systems [74, 76] but also the network-signature-based system [77], malicious URL detection systems [19, 66], and malware download detection system [58].

General features are the protocol, elapsed time, interval, existence of the identical communication, data size of request, data size of response, and port number (Nos. 1–7). The elapsed time is the difference between the analysis start time and the time when the target communication is sent. The interval is the difference between the time when the last and the target communications are sent. We set 1 for the existence of the identical communication if the identical communications have already

been collected in the analysis and 0 if they have not. Hostname-based features include the top level domain (TLD) rank, presence of "ip" in the fully qualified domain name (FQDN), presence of IP address, and presence of subdomain (Nos. 8–11). We use the highest Alexa[1] rank among domains with a certain TLD as the TLD rank. HTTP-based features include the HTTP method, status code, presence of `.exe` in filename, presence of `.zip` in filename, depth of file path, number of query parameters, and use agent (Nos. 12–18).

We vectorize categorical features by using one-hot encoding; we create a vector whose dimension is the number of categories and whose component corresponding to the vectorized category is 1 and other components are 0. Categorical features are protocols, HTTP methods, status codes, and user agent. Protocols are categorized into DNS, HTTP, HTTPS, other known protocols, and unknown ones. The HTTP methods are categorized into three types: GET, POST, and others. Status codes are divided into six groups on the basis of their 100 placement. For example, the first group includes 100, 101, and 102, and the second group includes 200, 201, 202, etc. The user agents are divided into three types: Mozzilla, unset, and others.

4.3.3 Neural Network Construction

To capture the malicious activities, we focus on the change in communication purposes. We construct a neural network in which communications related to the same purpose compose the same subtrees. We consider the relationship between the same communication purposes and the relationship between different communication purposes. The initial situation is that each node is composed of one communication. We select a set of nodes depending on the below three criteria and create their parent nodes. The selected nodes are removed from the selection candidates, and their top node is added to the candidates. We repeat this process to construct a tree-structured neural network. The first and second criteria are the relationships between the same communication purposes. The third criterion is the relationship between different communications purposes.

Hostname. Malware samples successively send communications related to a hostname, i.e., FQDN or IP address, to accomplish a purpose such as Internet connection tests, updates, and command

[1]https://www.alexa.com/topsites

reception. For example, a malware sample sends a DNS query to obtain the IP address of a FQDN and then sends a HTTP request to communicate with a C&C server. We combine all communications related to the same hostname. If the number of communications is more than two, we combine communications on the basis of time ordering because they are successively sent to accomplish the same purpose. Specifically, we combine the two earliest communications and make their parent node. Then we combine the parent node and the earliest communications next to the combined ones. This process is repeated until all communications related to a certain hostname are combined.

Identical URL Path and URL Query Parameters. If communications with C&C servers fail, malware samples frequently attempt to communicate with backup or alternative C&C servers. If the communication purpose with the backup servers is the same as that with the original C&C server, the path and query of these communications are assumed to be identical. Occasionally, malware samples change the URL query value. Therefore, if the path and query parameters of two communications are identical, they are supposed to be of the same purpose. However, communications that have different purposes can have the same general path such as `/index.php`. We thus take into account communications that have one or more query parameters. If the path and query parameters of the descendant nodes of A and that of the descendant nodes of B are identical, we combine nodes A and B.

Time Difference of Communications. After nodes are combined on the basis of the first and second criteria, candidate nodes have several descendant nodes. That is to say, communications are divided into several groups of communications. If the two communication groups have dependency, one group is sent subsequently to the other group. In other words, periods when communication groups are sent do not overlap. Hence, the communication groups, whose periods do not overlap, are supposed to have different purposes. Therefore, we combine two communication groups that have the closest periods. Let the set of times when communications of group A are sent be $T_A = \{t_{Ai}\}$ and let that of group B be $T_B = \{t_{Bj}\}$. The difference in periods of groups is calculated as $d = \sum_i \sum_j |t_{Ai} - t_{Bj}|$. This process is repeated until all communications become the descendant nodes of one node. In this manner, the tree-structured neural network is constructed.

4.3.4 Training and Classification

Since we obtain only the label of malware samples for training, we set the training labels to only the root nodes. After the training, the feature vectors of root nodes are calculated in accordance with (4.1). Finally, classification results are determined in accordance with (4.2).

4.4 Experimental Setup

We evaluate our proposed system in terms of efficiency for collecting novel HTTP requests. Here, we describe the experimental setup.

4.4.1 Dataset of Malware Samples

We conduct the evaluation with a dataset composed of malware samples collected from VirusTotal[2] in Jul.–Dec. 2017. We collect the malware samples that have different SHA1 hashes and that are detected by at least one anti-virus engine. We use AVClass [95] to refine our dataset by eliminating samples falsely detected by anti-virus engines. AVClass identifies the family name of a malware sample if several anti-virus engines label it with virtually the same family name. AVClass further identified potentially unwanted programs (PUPs) on the basis of certain keywords in labels output by anti-virus engines. We use malware samples that are identified by their family names and are not identified as PUPs by AVClass. Finally, we collected 42,856 unique malware samples.

We analyze all malware samples for 30 minutes with a safe dynamic analysis system called BotnetWatcher [8] to estimate when the determination should be made. Although we should ideally analyze them for a longer time, we selected 30 minutes as a sufficiently long time. Figure 4.3 shows the cumulative distribution function (CDF) of time when the last HTTP request was sent. On the basis of the data in Fig. 4.3, most malware samples stop sending HTTP requests within five minutes, and we can collect most HTTP requests in 30 minutes. In addition, Fig. 4.3 shows that it is appropriate that determination be made five minutes after analysis starts. Hence, we select five minutes as the determination time. We label a malware sample as *suspension* if all HTTP requests,

[2]https://www.virustotal.com/

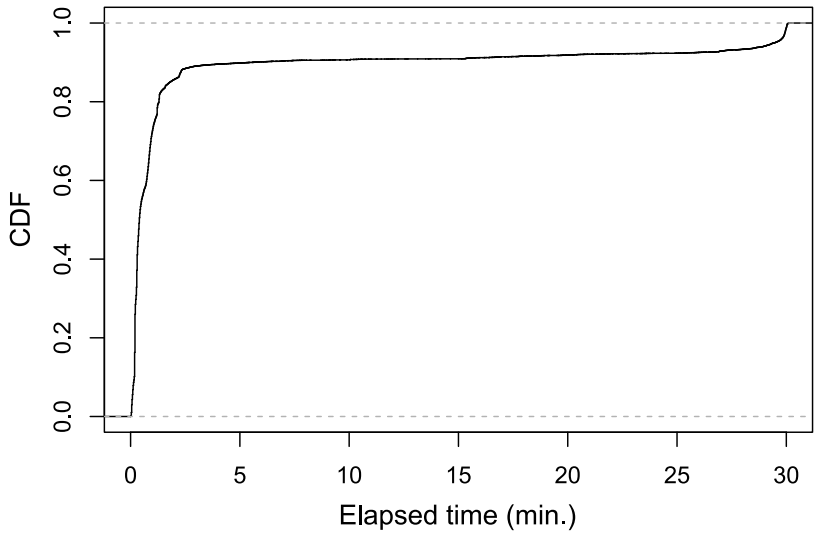


Figure 4.3: CDF of time when the last HTTP request was sent.

Table 4.2: Dataset.

	Suspension	Continuation	Period
Training	24,988	1,003	Jul.–Oct. 2017
Test	16,335	530	Nov.–Dec. 2017

which are collected in the last 25 minutes of its analysis, are collected in the first 5 minutes. On the other hand, we label a malware sample as *continuation* if all HTTP requests, which are collected in the last 25 minutes of its analysis, are *NOT* collected in the first 5 minutes.

In our evaluation, we use malware samples collected during Jul.–Oct. 2017 as training data and those collected during Nov.–Dec. 2017 as test data as shown in Table 4.2. As mentioned above, many more samples are labeled as *suspension* than *continuation* because many malware samples stop sending HTTP requests within five minutes. Note that we analyze all malware samples for 30 minutes for labeling, but we use communications sent in the first 5 minutes of analyses for training and classification. The average number of communications in the first 5 minutes was 98.7.

4.4.2 Conventional Systems for Comparison

We compare our proposed system with two naive systems, the unknown request-base system, and two behavior-based systems.

One naive system, which we call the continuation system, continues all analyses. The other, which we call the suspension system, suspends all analyses. The unknown request-based system continues the analysis if one or more HTTP requests not included in the reference set of the requests are collected in the short-period analysis. We use the set of HTTP requests collected in analyses of training data as the reference set.

The behavior-based systems make a determination on the basis of network behavior in the short-period analysis. Note that our proposed system also belongs to this type of system. Our system makes a determination on the basis of a sequence of feature vectors representing communications, but other types of features vectors can be leveraged for determination. One behavior-based system, which we call the overall system, leverages feature vectors representing all communications sent in the short-period analysis. The overall system extracts feature vectors used in the statistics-based method [76]: the number of application protocols, the number of communications related to a certain domains, etc. The other behavior-based system, which we call the individual system, leverages feature vectors representing individual communications. The individual system extracts feature vectors used in our proposed system, classifies them individually, and makes a determination by integrating individual classification results of all communications sent in the short-period analysis. Specifically, if one or more communications are classified as *continuation*, the analysis is continued. If no communication is classified as *continuation*, the analysis is suspended. Random forest [17] is utilized as the machine learning algorithm for these classifications because it performs nonlinear classification with high accuracy.

4.4.3 Hyperparameter Optimization

We split training data into prior-training and validation data to optimize hyperparameters of the RNN. We select the combination of parameters that have the highest F-measure (see Section 4.5 for definition). For optimization, we conduct grid search by changing the batch size, initialization interval of adagrad, and learning rate. A batch size is selected from 50, 100, and 500, an initialization interval from 1, 10, and 100, and a learning rate from 0.01, 0.001, and 0.0001. We set the number of iterations for training as 100. The best combination of parameters was 100 for the batchsize, 100

for the initialization interval, and 0.001 for the learning rate.

The hyperparameters of random forest for conventional systems, i.e., the number of decision trees and the number of features for each decision tree, are optimized by using the `tuneRF` function of the `randomForest` package in R [87] when a classifier is trained. For the individual system, the number of decision trees is 473 and that of features for each decision tree is 20. For the overall system, the number of decision trees is 91 and that of features for each decision tree is 2.

4.5 Experimental Results

We now report the experimental results. In Section 4.5.1, we show the results of the evaluation on classification. We compare our system with two other behavior-based systems in terms of classification performance. We further investigate the effectiveness and limitations of the RNN for classification by analyzing the classifier in detail. In Section 4.5.2, we show the results of the evaluation on HTTP request collection. We compare our system with all conventional systems in terms of collection efficiency and calculation time. In Section 4.5.3, we report the network behavior of malware samples from which novel HTTP requests are successfully collected on the basis of our system but not collected on the basis of the unknown request-based system. These case studies show the effectiveness of our system for HTTP request collection.

4.5.1 Evaluation on Classification

We evaluate the classification performance of our system by comparing it with those of the overall and individual systems. We further analyze the classifier of the RNN in terms of the contribution of features, important behavior for classification, and false positives/negatives. In this subsection, malware samples whose analysis should be continued and suspended are referred to as *positive* and *negative* samples, respectively.

Classification Performance. We evaluate classification performance by using widely used metrics: TPR, FPR, accuracy, area under the receiver operating characteristics (ROC) curve (AUC), precision, and F-measure. Note that TPR is also known as recall. A malware sample is classified as positive if

Table 4.3: Classification performance.

System	TPR (Recall)	FPR	Accuracy	AUC	Precision	F-measure
Overall	0.350	0.047	0.933	0.769	0.199	0.253
Individual	**0.794**	0.064	0.932	0.932	0.288	0.423
Proposed	0.755	**0.016**	**0.977**	**0.945**	**0.623**	**0.683**

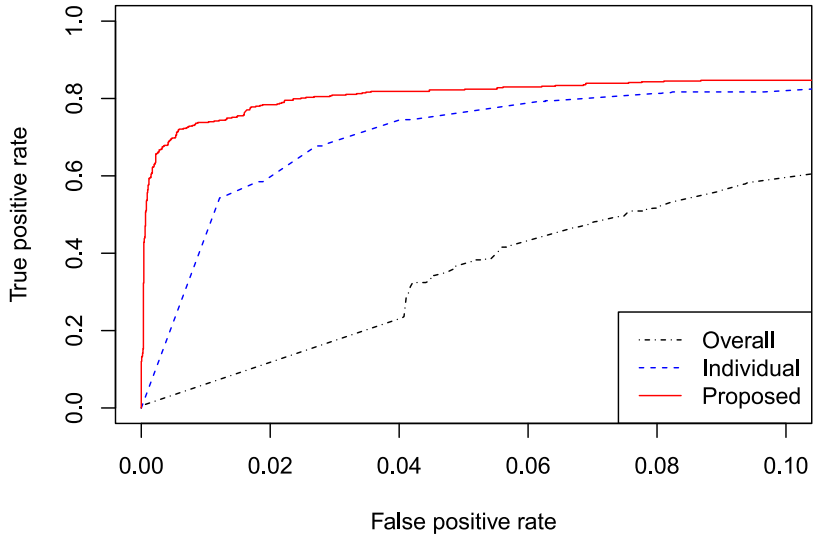


Figure 4.4: ROC curve.

the prediction probability is higher than a commonly used threshold, i.e., 0.5, in the training phase and evaluation of classification performance.

Table 4.3 shows the classification performance of each system. Our system outperforms conventional systems for most metrics. It is noteworthy that only our system achieves a high TPR and a low FPR. This results in our system having a high F-measure, which is defined by the number of true positives and false positives. Figure 4.4 shows the ROC curve from 0.0 to 0.1 FPR. Considering our dataset includes a small number of malware samples labeled *continuation*, i.e., *positive* samples, the classification performance at low FPRs is important. For example, to improve the collection rate (see Section 4.5.2 for the definition) at a TPR of 1.0, the FPR must be lower than 0.023. We select a sufficiently wide range of FPRs for Fig. 4.4. Our system stably achieved a higher TPR than conventional systems in this range of FPRs. Specifically, the TPRs of the overall, individual, and proposed system at an FPR of 0.01 are 0.06, 0.454, and 0.738, respectively. The TPRs at an FPR of

Table 4.4: Contribution of features.

Order	Type	Feature	Improvement
1	General	Elapsed time	0.068
2	General	Data size of request	0.010
3	General	Existence of the identical comm.	0.006
4	Hostname	Presence of "ip" in FQDN	0.006
5	Hostname	Presence of subdomain	0.005
6	HTTP	Depth of file path	0.004
7	General	Data size of response	0.003
8	General	Protocol	0.003
9	Hostname	TLD rank	0.003
10	HTTP	Status code	0.003

0.05 are 0.366, 0.745, and 0.817. The TPRs at an FPR of 0.10 of FPR are 0.582, 0.817, and 0.847. Not only Table 4.3 but also Fig. 4.4 shows that only our system achieves a high TPR and low FPR.

Contribution of Features. To understand how the classification is conducted, the contribution of each feature to classification is informative. We evaluate how much each feature improves accuracy of the RNN. Specifically, we evaluate the accuracy of the RNN using all features and using all but one of features. We then calculate their difference. Table 4.4 shows the ten most contributed features based on accuracy improvement. All types of features contributed to the classification, but general features contribute the most among the three feature types. These results show that application protocols other than HTTP or HTTPS are also important for classification.

Important Behavior for Classification. To more clearly identify the reason the RNN achieves the highest classification performance, we analyze important network behavior for classification. Since a small number of malware samples is labeled as *continuation* in our dataset, these samples need to be accurately detected to achieve high classification performance. For this reason, we analyze network behavior of malware samples classified as *continuation*. To this end, we take advantage of the fact that the RNN can output prediction probability of every node as well as the root node. Note that the RNN classifies malware samples as *continuation* if the prediction probability of the root node is larger than 0.5. We identify nodes satisfying the following three conditions and analyze communications that are descendants of the nodes.

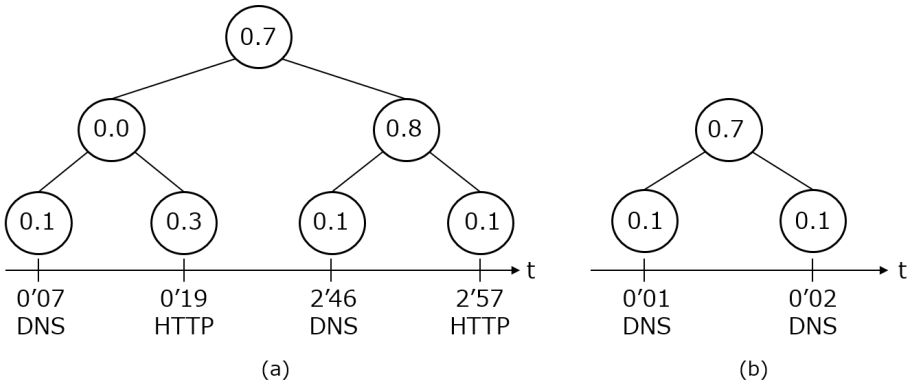


Figure 4.5: Network behavior and predicted probabilities. Predicted probability is written in each node.

1. The prediction probabilities of all descendants are lower than 0.5.

2. The prediction probabilities of all ancestors are larger than 0.5.

3. The prediction probability of the node is much larger than those of its children.

One important behavior is a pair of a DNS query and HTTP request in the latter half of the short-period analysis as shown in Fig. 4.5(a). These communications are related to a FQDN with which the malware sample has not communicated before. The malware samples from which novel HTTP requests are collected by continuing their analyses successively send communications related to several FQDNs to accomplish different purposes such as Internet connection tests, updates, and command reception. On the other hand, the malware samples that stop activities in the middle of their analyses send communications related to only a few FQDNs right after starting their analyses or repeatedly send communications related to a FQDN. The RNN is supposed to learn that the behavior of starting to send communications related to a new FQDN in the latter half of the analysis indicates continuation of malicious activities.

Another important behavior is a few DNS queries right after starting the analysis as shown in Fig. 4.5(b). These DNS queries are related to different FQDNs and are properly resolved. Since this malware sample sent a DNS query and HTTP request related to a different FQDN after five minutes, it is supposed to test the Internet connection with DNS and sleep a certain period of time to conceal malicious activities. On the other hand, the malware samples that stop activities in the middle of

their analyses send a DNS query and HTTP request right after starting their analyses. The RNN is supposed to learn that a few successfully resolved DNS queries indicate continuation of malicious activities and are likely to be followed by other malicious activities after a certain period of time.

False Positives and False Negatives. We analyze false positives and negatives to understand limitations of the RNN. The false positives, i.e., malware samples falsely classified as *continuation*, send many DNS queries and HTTP requests in the latter half of the short-period analysis. As mentioned above, this behavior is similar to that of malware samples from which novel HTTP requests are collected by continuing their analyses. This is why false positives occur. The false negatives, i.e., malware samples falsely classified as *suspension*, send a few HTTP requests right after starting the analyses and slept about five minutes. As mentioned above, this behavior is similar to that of malware samples that stop activities in the middle of their analyses. This is why false negatives occur. These examples show that the RNN cannot accurately classify all malware samples. However, the RNN achieves high classification performance by learning common network behavior.

4.5.2 Evaluation on HTTP Request Collection

We compare our system with all conventional systems in terms of collection efficiency and calculation time.

Collection Efficiency. We evaluate efficiency for collecting novel HTTP requests, which are not collected in the analyses of training data. Table 4.5 shows the number of novel HTTP requests, analysis time, and collection rate, i.e., the number of novel HTTP requests per minute. The analysis time is the summation of time for which malware samples are analyzed. A malware sample is analyzed for 5 minutes if the analysis is suspended. On the other hand, a malware sample is analyzed for 30 minutes if the analysis is continued. The analysis time does not include the time of activating the dynamic analysis system [8] and for feature extraction and classification. Since the continuation system continued all analyses, its analysis time is $30 \times 17,302 = 505,950$. The analysis time of the suspension system is $5 \times 17,302 = 84,325$ because it suspends all analyses. As a reference, we also show the collection efficiency of the classifier whose accuracy is 1.0 (oracle in Table 4.5). We further calculate the ratio of collected HTTP requests and time reduction compared

Table 4.5: HTTP request collection efficiency.

System	Number	Ratio	Time (min.)	Reduction	Collection rate
Suspension	14,467	77%	84,325	83%	0.172
Continuation	18,676	100%	505,950	0%	0.037
Unknown request	18,550	99%	332,650	34%	0.056
Overall	14,467	77%	84,325	83%	0.172
Individual	14,467	77%	84,325	83%	0.172
Proposed	17,534	94%	94,100	82%	0.186
Oracle	18,676	100%	99,150	80%	0.188

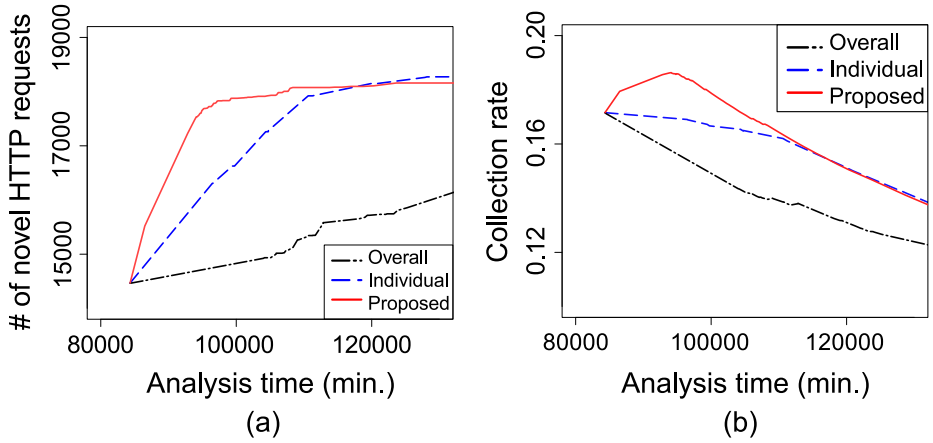


Figure 4.6: HTTP request collection efficiency with different thresholds.

with the continuation system. In this evaluation, we optimize the threshold of prediction probability above which malware samples are classified as *continuation* because the best threshold in terms of classification performance is different from that in terms of collection efficiency. Specifically, we select the best threshold in terms of collection rate.

The unknown request-based system collects most novel HTTP requests but does not sufficiently reduce analysis time. The overall and individual system can not improve the collection rate by determination due to the low classification performance. Consequently, their best collection rate is the same as that of the suspension system. Our system can improve the collection rate by determination and thus collects a large number of novel HTTP requests in a short analysis time. Our system has a higher collection rate than the conventional systems.

To precisely understand the collection efficiency of behavior-based systems, we investigate the

Table 4.6: Calculation time (sec./sample).

System	Feature extraction	Classification
Overall	0.0007	0.00001
Individual	0.0008	0.00006
Proposed	0.6730	0.00387

number of novel HTTP requests, analysis time, and collection rate with different thresholds as shown in Fig. 4.6. When a higher threshold is used, both the number of novel HTTP requests and analysis time increase because more malware samples are classified as *continuation*. To draw Fig. 4.6, we analyzes all malware samples for 30 minutes beforehand. If a malware sample is classified as *suspension*, we use the first 5 minutes of its analysis. On the other hand, if a malware sample is classified as *continuation*, we use 30 minutes of its analysis. We calculate the number of novel HTTP requests, analysis time, and collection rate with different thresholds and plot them in Fig. 4.6. Since our system achieves higher classification performance, it collects more novel HTTP requests in the same analysis time than other systems and successfully increases the collection rate by determination. The overall and individual systems can not sufficiently increase the number of novel HTTP requests even with longer analysis time. This results in the decrease in the collection rate.

Calculation Time. Considering deployment, calculation time must be much shorter than the analysis period. Hence, we investigate calculation time for feature extraction and classification as shown in Table 4.6. More precisely, the calculation time is the time of executing programs for feature extraction and classification per sample in the test phase. Our proposed system take more time than the conventional systems, but the calculation time is much shorter than that when a malware sample is analyzed. Specifically, a malware sample is analyzed for 5 minutes if a malware sample is classified as *suspension* and analyzed for 30 minutes if a malware sample is classified as *continuation*. Therefore, our proposed system can be deployed for efficient dynamic analysis.

4.5.3 Case Study

We confirm the effectiveness of our system by analyzing the network behavior of malware samples from which novel HTTP requests are successfully collected on the basis of our system but not collected on the basis of the unknown request-based system. These malware samples did not send novel HTTP requests for the first 5 minutes. This is why the unknown request-based system does not collect novel HTTP requests. The network behavior shown here is simplified due to space limitations.

Different Behavior of Secondary Malware. Our proposed system collects novel HTTP requests from a malware sample whose secondary malware sample behaves differently than in the past analysis, as shown in Table 4.7. This malware sent communications related to `example1.com` at first and downloaded an executable, which is supposed to be a secondary malware sample. Next, it sent communications related to `example2.com`. Since one of their URLs included `config`, the malware sample is supposed to obtain the configuration of its secondary malware sample. Communications for obtaining the configuration are typical for malware samples that continue to send communications, as with the behavior shown in Fig. 4.5(a). Since the RNN properly learns such behavior, this malware sample is classified as *continuation*. Then, the secondary malware sample successively sends a DNS query and HTTP request related to another FQDN. Since the training dataset included a malware sample that performed the same malicious activities as the analyzed one, the HTTP requests to `example1.com` and `example2.com` are already collected. However, subsequent HTTP requests are novel because their secondary malware samples or their configurations are periodically changed by attackers depending on their purpose.

DGA. The next example is a malware sample using the domain-name generation algorithm (DGA) as shown in Table 4.8. This malware sample sends many DNS queries related to randomly generated FQDNs using DGA and then sends HTTP requests to properly resolved FQDNs. The actual strings of `random`x in Table 4.8 are randomly generated by concatenating some words. Attackers use DGA to evade blocking based on blacklists of FQDNs by generating different FQDNs at different times and using a few of them. Malware samples leveraging DGA typically continue to send communications. Therefore, the RNN learns such behavior and classifies this malware sample as *continuation*.

Table 4.7: Network behavior of a malware sample whose secondary malware sample behaves differently.

Time	Protocol	FQDN or URL	Novel
0'05	DNS	example1.com	
0'06	HTTP	http://example1.com/malware.exe	
0'20	DNS	example2.com	
0'30	HTTP	http://example2.com/config	
5'20	DNS	example3.com	
5'40	HTTP	http://example3.com/?sid=xxx	✓

Table 4.8: Network behavior of a malware sample using DGA.

Time	Protocol	FQDN or URL	Novel
0'20	DNS	random1.com	
0'30	HTTP	http://random1.com/	
3'35	DNS	random2.com	
3'40	DNS	random3.com	
6'40	DNS	random4.com	
6'45	HTTP	http://random4.com/	✓

Properly resolved FQDNs differ depending on the time of the analysis, and `random4.com` is used at the time of the analysis for the first time. Therefore, our system collects novel HTTP requests from this malware sample.

Long Sleep. The last example is a malware sample that sleeps a long time after sending a few DNS queries, as shown in Table 4.9. This malware sample sends a DNS query and HTTP request related to another FQDN about 25 minutes after sending the DNS query related to `example2.com`. This malware sample is supposed to test the Internet connection right after starting the analysis and sleep a long time to conceal malicious activities. For this reason, the unknown request-based system cannot continue the analysis. However, as shown in Fig. 4.5(b), the behavior of this malware sample is common for malware samples that continue to send communications. Based on this behavior, the RNN classifies this malware sample as *continuation*, and our proposed system successfully collects a novel HTTP request.

Table 4.9: Network behavior of a malware sample that sleeps a long time.

Time	Protocol	FQDN or URL	Novel
0'02	DNS	example1.com	
0'03	DNS	example2.com	
25'02	DNS	example3.com	
25'03	HTTP	http://example3.com/e?xxx	✓

4.6 Discussion

We discuss our focus, the validity of the evaluation, and the limitation of our system in this section.

Our Focus. Our system would not be able to increase collection efficiency if malware samples belonging to the same family were input. Hence, the selection of input malware samples is also important to increase efficiency. However, suitable samples can be selected by simply selecting diverse malware families. For this reason, we consider how to increase collection efficiency after malware samples are selected.

Validity of Evaluation. We propose a system for efficiently collecting novel HTTP requests to enhance detection performance of the network-based methods such as malicious communication detection [22, 77] and blacklist-based detection. In Section 4.5, we show that our proposed system efficiently collects novel HTTP requests, but we do not investigate whether it enhances detection performance. Since the effect of our system differs depending on the methods, we cannot conduct a uniform evaluation. However, we show in case studies that our system collects novel HTTP requests useful for detection.

In terms of maintaining classification performance, the classifier of the RNN needs to be retrained periodically. Considering the computational cost of retraining, the RNN should keep classification performance high for a long period. In our evaluation, we use test data collected for two months, which is a reasonable interval for retraining, and show that our system achieves high classification performance for two months. Hence, our system is expected to maintain high classification performance by retraining at most every two months.

Limitation. If malware samples sleep a long time after dynamic analysis starts, our proposed system cannot be applied. In this case, we have two options. One is continuing analysis until

malware samples send communications. The other is suspending analysis. Both decrease efficiency or exhaustiveness. However, the percentage of these malware samples is reported to be only 5.39% [10]. In our evaluation, we can not apply our proposed system only to 0.4% of malware samples. Therefore, the efficiency of our system is not significantly decreased by these malware samples.

4.7 Related Work

4.7.1 Analysis, Detection, and Countermeasure

The methods for analysis, detection, and countermeasure have been extensively studied from the following viewpoints.

Static feature: One of the main methods used by modern anti-virus software is signature-based scanning. Griffin et al. proposed a string signature generation system countering variants of malware families to reduce signature database size [35]. However, almost all recent malware samples are obfuscated by the packer, and their characteristic strings are enfolded. Therefore, malware samples should be unpacked before they are applied to signature-based methods [69, 92, 125, 130].

Host behavior: On an infected host, the analysis of system call or API call events is useful to detect malware samples [56, 70, 115]. However, there are now malware samples that circumvent or interfere with monitoring such host-based events on the analysis environment, i.e., malware sandbox. To counter sandbox-aware malware, Kirat et al. proposed an efficient analysis system running on actual hardware [54].

Network behavior: The communications initiated by malware, e.g., C&C, are useful to build countermeasures such as blacklisting and network-based signature generation [77, 85].

4.7.2 Appropriate Sample Selection to Avoid Full Analysis

Bayer et al. proposed a technique that avoids analyzing the same polymorphic programs and reduces the amount of time required for analyzing malware samples [12]. To detect polymorphic malware samples, it analyzes them for a short time and finds the most behaviorally similar sample. It

can successfully avoid the full analysis of about 25.3% of malware samples. Neugschwandtner et al. proposed a system leveraging both the behavioral clustering of Bayer et al. [12] and static feature-based clustering that selects the malware sample, which is most likely to yield relevant information (e.g., C&C communication), without actually running it [80]. We infer that analyzing malware samples generated by the same toolkit, e.g., polymorphic malware samples, and controlled by different attackers are also important for collecting informative communication because those malware samples may access different malicious sites. In our study, however, the time reduction of analysis is superior, although we do not eliminate polymorphic malware samples, e.g., our proposed system avoids the full analysis of 98% of malware samples and reduces analysis time by 82%.

4.8 Summary

We propose a system for efficiently collecting novel HTTP requests with dynamic malware analysis. Specifically, we analyze a malware sample for a short period and then determine whether its analysis should be continued or suspended. Our system identifies malware samples whose analyses should be continued on the basis of the network behavior in their short-period analyses. To make an accurate prediction, we focus on the fact that malware communications resemble those of natural language from the viewpoint of data structure. For this reason, we apply the recursive neural network to our proposed system. In the evaluation with 42,856 malware samples, our proposed system collects 94% of novel HTTP requests and reduces analysis time by 82% in comparison with the system that continues all analyses. We further find that our system effectively collects novel HTTP requests from a malware sample whose secondary malware sample behaves differently, a malware sample that uses a domain generation algorithm, and a malware sample that sleeps for a long period.

Chapter 5

Detecting Unknown Families by Prioritizing Family-Invariant Features

5.1 Introduction

Malicious data such as malicious websites and malicious Android applications are created with attack tools and used to efficiently accomplish attackers' malicious objectives. For example, malware samples are created with toolkits [18] and malicious websites are created with exploit kits [34]. We define a set of malicious data created with the same attack tool as *a family* in this Chapter. To prevent damage caused by malicious data, researchers and security vendors have proposed sophisticated detection systems that classify malicious and benign data, i.e., binary classification, with DNNs [71, 90, 129]. These DNNs are well-designed to extract high-order semantic features (representations) from malicious data through their multiple layers. Since the extracted representations are highly effective for classification, DNN-based systems have outperformed conventional systems based on traditional machine learning (ML) such as SVM [126] and random forest [17].

At the same time, attackers continuously develop new attack tools to evade DNN-based systems [119]. Unknown families, which are malicious data created with new attack tools, have malicious behavior, obfuscation algorithms, and anti-analysis functions different from known ones [47, 124]. The emergence of unknown families causes significant changes in features of

malicious data, i.e., concept drift. Concept drift is known to degrade classification performance of classifiers that assume a classification target is drawn from the same distribution as the training data, i.e., i.i.d. random variables [49]. DNNs for detecting malicious data implicitly assume i.i.d. variables because they are designed on the basis of DNNs for image recognition and natural language processing, which assume i.i.d. variables. Consequently, DNN-based systems trained using known families have difficulty detecting unknown families [84].

In this Chapter, we study how to improve existing DNN-based systems in terms of detection of unknown families. If features of unknown families completely differ from those of known families, unknown families are quite difficult to detect. In fact, some features are inherent across different families because malicious data include similar code or produce similar behavior to exploit vulnerabilities or cost-effectively achieve a successful attack. For example, in drive-by download attacks, applications exploiting browsers or their plugins are limited to Flash, PDF, and Java, and redirections of websites are always abused to lure victims to exploit websites [19]. If features inherent across known families (family-invariant features) are prioritized in classification, unknown families become easier to detect. Therefore, we aim at building a classifier that prioritizes family-invariant features.

A naive approach is to apply feature selection to DNN-based systems. Feature selection is commonly used for traditional-ML-based systems that accept feature vectors as input. Elements of feature vectors are semantic features, e.g., features representing obfuscation and malicious communication. In the process of feature selection, elements effective for classification are selected and input to classifiers. Similarly, we can select elements representing family-invariant features and input them to DNNs. However, this approach is not applicable to some DNN-based systems because they accept raw data, e.g., opcode sequence and dex bytecode, as input [71, 129]. Since elements of raw data are not semantic features but code fragments included everywhere in both malicious and benign data, we cannot determine whether each element represents a family-invariant feature or not. For this reason, we cannot apply feature selection to those DNN-based systems. To propose a method applicable to all DNNs, we focus on representations extracted through multiple layers. If representations consist of family-invariant features, the classification results become based on family-invariant features. Therefore, we optimize representations so that they consist of

family-invariant features in addition to training for classification of malicious and benign data.

We propose two optimization methods that can be applied to all DNN-based systems' training and improve them in terms of detection of unknown families. Our methods are designed to solve a problem in cyber security by modifying an optimization method studied in domain adaptation [2].

One is FirOpt$_{\text{all}}$, which optimizes representations so that they consist of features inherent across *all* known families. FirOpt stands for **F**amily-**i**nvariant **r**epresentation **Opt**imization. For optimizing representations (so-called representation learning), we use an additional neural network (NN) that accepts representations as input and classifies known families. The NN can easily classify known families if representations include features specific to each known family. In other words, if representations consist of features inherent across all known families, the NN cannot accurately classify known families. Therefore, FirOpt$_{\text{all}}$ optimizes representations so that the NN cannot accurately classify known families on the basis of the representations. As a result, FirOpt$_{\text{all}}$ builds a classifier that prioritizes features always used in attacks without depending on families, e.g., malicious redirections used in drive-by download attacks.

The other method is FirOpt$_{\text{part}}$, which improves robustness of FirOpt$_{\text{all}}$. That is, FirOpt$_{\text{part}}$ is effective without depending on datasets and NN architectures. FirOpt$_{\text{all}}$ may be ineffective when the number of features inherent across all known families is small. In this case, the optimized representations consist of a small number of features, and thus a classifier based on the representations cannot accurately classify malicious and benign data. FirOpt$_{\text{part}}$ relaxes the constraint of FirOpt$_{\text{all}}$ on representation learning. Specifically, it includes features inherent across a *part* of known families in representations because such features may be effective for unknown family detection. For example, Flash is abused by a part of known families, but a feature regarding Flash is effective if an unknown family abuses Flash for exploitation. Since features inherent across *all* known families are expected to be more effective than those inherent across a *part* of known families, FirOpt$_{\text{part}}$ more highly prioritizes features inherent across a larger number of known families. Optimized representations have sufficient features to accurately classify malicious and benign data, and thus FirOpt$_{\text{part}}$ robustly improves DNN-based systems in terms of detection of unknown families.

We evaluate whether our methods robustly improve DNN-based systems in terms of detection of unknown families with three case studies. We select diverse and common targets of detection

Table 5.1: Example of communications to malicious websites.

Family	URL (host)	URL (path)	URL (query string)	Content-type
Rig	top.[snipped].org	/	?ct=kul[snipped]	Flash
Neutrino	neopyrali[snipped].com	/street/[snipped].swf	(n/a)	Flash
Magnitude	2fcdg7ef8.[snipped].gdn	/[snipped]8275b280	?win[snipped]	Flash

systems for cyber security and prepared three datasets: malicious websites, malicious Android applications, and malicious PE files. For NN architectures, we use fully connected NNs and a recurrent NN (RNN) to evaluate applicability for both non-structured and structured data. Our evaluation results show that $\text{FirOpt}_{\text{part}}$ robustly improves DNN-based systems without depending on datasets and NN architectures. Specifically, $\text{FirOpt}_{\text{part}}$ outperforms a conventional optimization method, which optimizes an NN only so that it accurately classifies malicious and benign data, by at most 19%, 19%, and 7% in terms of TPRs for malicious websites, Android applications, and PE files, respectively.

Our contributions are summarized as follows:

- We propose an optimization method $\text{FirOpt}_{\text{part}}$ that can improve existing DNN-based systems in terms of detection of unknown families without depending on datasets and NN architectures.
- Our evaluation using three datasets of cyber security and two NN architectures shows that not only features inherent across *all* known families but also those inherent across a *part* of known families are necessary to robustly detect unknown families.

5.2 Motivating Example

We use examples of communications to illustrate the effectiveness of our methods. Table 5.1 shows URLs and content-type of communications to malicious websites of three families. These malicious websites are constructed with exploit kits (e.g., Rig, Neutrino, and Magnitude) for drive-by download attacks. The structures of their URLs greatly differ. For example, the URLs of Rig and Magnitude have query strings, but the URL of Neutrino does not. The lengths of domains and depths of URL paths also differ. On the other hand, some features are inherent across all families. Communications of all families have the same identifier of the content-type.

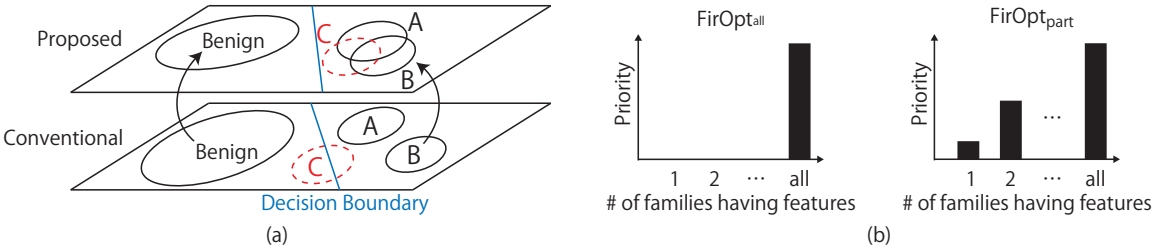


Figure 5.1: (a) Comparison between conventional and proposed methods. Families A and B are known, and family C is unknown. (b) Comparison between FirOpt$_{all}$ and FirOpt$_{part}$.

Figure 5.1(a) illustrates the difference between a conventional method and our proposed methods. We consider a situation in which families A and B are created with previously developed exploit kits, and family C is created with a newly developed one. In other words, families A and B are known, and family C is unknown. The conventional method optimizes the decision boundary so that it can distinguish between malicious and benign data. When features differ depending on families as shown in Table 5.1, distributions of families do not overlap. As a result, unknown family C is difficult to detect using known families (i.e., families A and B) and benign data as shown in Fig. 5.1(a).

The situation we are considering is called concept drift. More precisely, concept drift is the situation in which a distribution of data changes over time. In Fig. 5.1(a), the distribution of malicious data changes from A and B to A, B, and C. To prevent a classifier from degrading classification performance under concept drift, a method has been proposed for detecting change in a distribution [49]. If change is statistically detected, a classifier is retrained by using a new distribution of data (known and unknown families in our setting). Our methods are aimed at improving detection of unknown families without detecting concept drift and retraining classifiers.

Our methods optimize representations so that they consist of family-invariant features. In Fig. 5.1(a), our optimization changes distributions of families A, B, and C from bottom ones to upper ones. If representations are optimized to consist of features inherent across families A and B, the distributions of families A's and B's representations become similar. These features are likely to be included in unknown family C as features inherent across all families in Table 5.1. In this case, the distribution of family C's representations becomes similar to those of families A and B as shown

in Fig. 5.1(a). A classifier based on such representations is expected to accurately detect unknown family C even if it is trained by using known families A and B and benign data.

Next, we clarify the difference between our methods: $\text{FirOpt}_{\text{all}}$ and $\text{FirOpt}_{\text{part}}$. $\text{FirOpt}_{\text{all}}$ only prioritizes features inherent across *all* known families as shown in Fig. 5.1(b). If the identifier of the content-type is only the feature inherent across all families, $\text{FirOpt}_{\text{all}}$ builds a classifier that prioritizes only the identifier of the content-type. In this case, it cannot accurately classify malicious and benign websites because some benign websites also use Flash. Consequently, it is unable to detect both known and unknown families. Meanwhile, $\text{FirOpt}_{\text{part}}$ prioritizes features inherent across a *part* of known families as well as features inherent across *all* known families as shown in Fig. 5.1(b). $\text{FirOpt}_{\text{part}}$ builds a classifier that prioritizes not only the identifier of the content-type but also the presence of query strings and length of domains. The classifier is based on sufficient features to classify malicious and benign websites, and thus it can detect unknown families. Note that $\text{FirOpt}_{\text{part}}$ more highly prioritizes features inherent across a larger number of known families as shown in Fig. 5.1(b). This prevents a classifier from over-fitting features inherent across a small number of known families, and $\text{FirOpt}_{\text{part}}$ differs from the conventional method on this point.

5.3 Family-invariant Representation Optimization

5.3.1 Notations

We use a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)\}_{i=1}^{N}$ for building a classifier, where $\mathbf{x}_i$ is an input, $\mathbf{y}_i \in \mathbb{R}^2$ is a class label, and $\mathbf{z}_i \in \mathbb{R}^{N_z}$ is a family label. A benign class label is denoted as $\mathbf{y}_i = [1, 0]$, and a malicious one is denoted as $\mathbf{y}_i = [0, 1]$. The family labels are one-hot vectors and attached only to malicious data, and N_z denotes the number of known families. Note that training data includes only known families as malicious data.

A multilayer NN is denoted as a function F. When $\mathbf{x}$ is input to an NN F, the predicted class label is denoted as $\hat{\mathbf{y}} = F(\mathbf{x}; \boldsymbol{\theta}_F)$, where $\boldsymbol{\theta}_F$ is parameters of the NN. An NN can be considered by dividing it into two parts: lower layers F_0 and higher layers F_1. In this case, the predicted class label is denoted as $\hat{\mathbf{y}} = F_1(F_0(\mathbf{x}; \boldsymbol{\theta}_{F_0}); \boldsymbol{\theta}_{F_1})$. The activations at an intermediate layer are called a

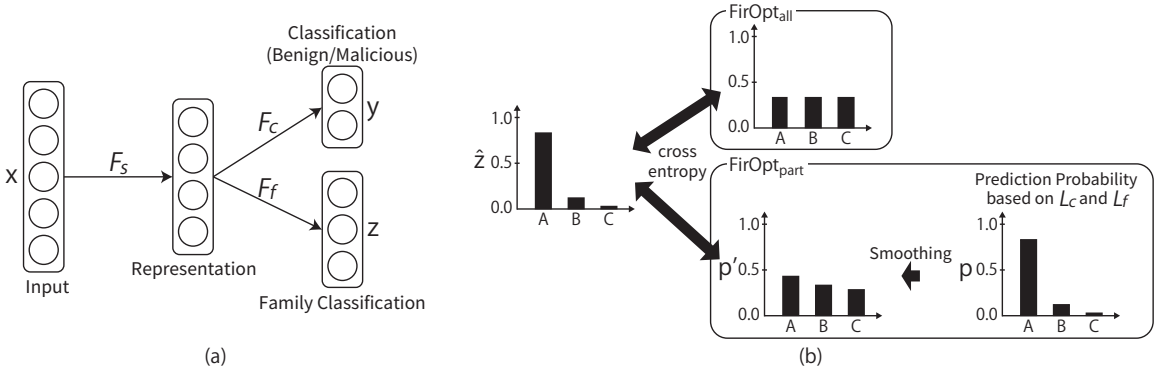


Figure 5.2: (a) Overview of neural network used in our proposed methods. (b) Family confusion loss.

representation and denoted as $\mathbf{h}$. For example, a representation $\mathbf{h} = F_0(\mathbf{x}; \boldsymbol{\theta}_{F_0})$ denotes the activations when $\mathbf{x}$ is propagated through F_0. For convenience, we use $F(\mathbf{x}; \boldsymbol{\theta}_F)$ and $F(\mathbf{x})$ interchangeably in this Chapter.

5.3.2 $\text{FirOpt}_{\text{all}}$

We use an NN that consists of a shared network and two heads, as shown in Fig. 5.2(a). This NN is designed by referring to that of domain adaptation [2]. The shared network outputs a representation, i.e., $\mathbf{h} = F_s(\mathbf{x})$, one head predicts a class label, i.e., $\hat{\mathbf{y}} = F_c(\mathbf{h})$, and the other head predicts a family label, i.e., $\hat{\mathbf{z}} = F_f(\mathbf{h})$. F_f is an additional NN for representation learning. In the training phase, the whole NN, i.e., F_s, F_c, and F_f, is optimized using benign data and malicious data (i.e., known families) in terms of classification of class labels and representation learning. In the test phase, F_s and F_c classify benign data and malicious data (i.e., unknown families) on the basis of $\hat{\mathbf{y}} = F_c(F_s(\mathbf{x}))$.

In the optimization for the training phase, we consider three losses: one for classification of class labels and two for representation learning. The loss for classification of class labels is called the classification loss. This loss is used to make F_s and F_c accurately classify malicious and benign

data and defined by using the cross entropy as follows:

$$\mathcal{L}_c = -\sum_{i=1}^{N}\sum_{j=1}^{2} y_{ij}\log\hat{y}_{ij}. \tag{5.1}$$

For representation learning, we adversarially optimize F_f and F_s using two losses. One loss is considered in the optimization of F_f, and the other is considered in the optimization of F_s. The former, which is considered in the optimization of F_f, is called the family classification loss. This loss is used to make F_f accurately classify known families and defined by using the cross entropy as follows:

$$\mathcal{L}_f = -\sum_{i=1}^{N}(\mathbb{1}[y_{i2}=1]\sum_{j=1}^{N_z} z_{ij}\log\hat{z}_{ij}). \tag{5.2}$$

Note that the family classification loss is calculated by using only malicious data because family labels are attached only to them. The latter, which is considered in the optimization of F_s, is called the family confusion loss. This loss is used to increase family-invariant features in representations. We define family confusion loss using the prediction probabilities of known families $\hat{\mathbf{z}}$ because we can identify whether representations consist of family-invariant features on the basis of $\hat{\mathbf{z}}$. When representations consist of features across *all* known families, distributions of families' representations completely overlap. In this case, the family classification is too difficult, and thus elements of $\hat{\mathbf{z}}$ become equal, i.e., a uniform distribution. Therefore, we use a uniform distribution as the objective of $\hat{\mathbf{z}}$. The family confusion loss is defined using the cross entropy between a uniform distribution and $\hat{\mathbf{z}}$ as shown in Fig. 5.2(b):

$$\mathcal{L}_{conf} = -\sum_{i=1}^{N}(\mathbb{1}[y_{i2}=1]\sum_{j=1}^{N_z}\frac{1}{N_z}\log\hat{z}_{ij}). \tag{5.3}$$

The family confusion loss is used to optimize an NN in the opposite way to the family classification loss. Hence, the family confusion loss is also called the adversarial loss.

To robustly optimize an NN on the basis of the opposite losses, we iteratively optimize the

parameters of NNs on the basis of the following two objectives:

$$\min_{\boldsymbol{\theta}_s,\boldsymbol{\theta}_c} \quad \mathcal{L}_c + \alpha\mathcal{L}_{conf} \tag{5.4}$$

$$\min_{\boldsymbol{\theta}_f} \quad \mathcal{L}_f, \tag{5.5}$$

where α is a hyperparameter that controls the effect of the family confusion loss.

5.3.3 $\text{FirOpt}_{\text{part}}$

We use the same NN shown in Fig. 5.2(a) for $\text{FirOpt}_{\text{part}}$ as $\text{FirOpt}_{\text{all}}$. The NN is optimized on the basis of (5.4) and (5.5) considering three losses: the classification, family classification, and family confusion losses. The only difference is the definition of the family confusion loss. The classification and family classification losses are defined by (5.1) and (5.2), respectively.

We define the family confusion loss of $\text{FirOpt}_{\text{part}}$ so that representations include more features inherent across a larger number of known families. If features specific to a small number of known families are decreased from the representations, the prediction probabilities of known families are smoothed and become similar to a uniform distribution because family classification becomes more difficult. Therefore, we use smoothed prediction probabilities to define the family confusion loss as described in the following two steps.

Step 1. We optimize an NN without a constraint of representation learning by minimizing two objectives: $\min_{\boldsymbol{\theta}_s,\boldsymbol{\theta}_c} \mathcal{L}_c$ and $\min_{\boldsymbol{\theta}_f} \mathcal{L}_f$. We use the prediction probabilities of known families $\mathbf{p} = F_f(F_s(\mathbf{x}))$ in Step 2.

Step 2. We define $\mathbf{p}'$ by smoothing $\mathbf{p}$ with a constant $a \in \mathbb{R}$:

$$p'_{ij} = \frac{p_{ij} + a}{\sum_{k=1}^{N_z}(p_{ik} + a)}. \tag{5.6}$$

By using $\mathbf{p}'$ as the objective of $\hat{\mathbf{z}}$, the family confusion loss is defined as the cross entropy between

Table 5.2: Prediction probabilities of families A, B, and C before and after smoothing, where $a = 1.0$.

Families across which features are inherent	Before smoothing			After smoothing			
	p_A	p_B	p_C	p'_A	p'_B	p'_C	$p_A - p'_A$
A	1.00	0.00	0.00	0.50	0.25	0.25	0.50
A and B	0.50	0.50	0.00	0.38	0.38	0.25	0.12
A, B, and C	0.33	0.33	0.33	0.33	0.33	0.33	0.00

$\mathbf{p}'$ and $\hat{\mathbf{z}}$:

$$\mathcal{L}_{conf} = -\sum_{i=1}^{N}(\mathbb{1}[y_{i2} = 1]\sum_{j=1}^{N_z} p'_{ij} \log \hat{z}_{ij}). \tag{5.7}$$

Note that a smoothing parameter a is optimized by cross-validation using training data.

We describe why we can optimize representations so that they include more features inherent across a larger number of known families on the basis of this family confusion loss. Table 5.2 shows examples of prediction probabilities before and after smoothing. The difference between the prediction probabilities before and after smoothing (i.e., $p_A - p'_A$) indicates the degree of constraints based on the family confusion loss. Table 5.2 shows that the constraints are larger when representations consist of features inherent across a smaller number of known families. As a result, features inherent across a small number of known families tend not to be included in representations.

5.4 Evaluation

We evaluate whether our methods robustly improve detection of unknown families with three case studies. Robustness is important for our methods because it determines their applicability. Specifically, if they are effective without depending on datasets and NN architectures, our methods can be applied to any DNN-based system and improve its detection of unknown families. We first describe the experimental setup and then report the experimental results of the three case studies. We conduct the evaluation using an Ubuntu server with 12 core CPU and 64GB RAM.

Table 5.3: Dataset of malicious website case study.

		Training		Test	
Label	Family	Period	#	Period	#
Benign		Oct. 10, 2017	10k	Jan. 16, 2018	100k
Malicious	Rig	May 7, 2015–Nov. 5, 2016	270	Nov. 7, 2016–Oct. 25, 2017	270
	Neutrino	Jun. 19, 2013–Jul. 12, 2016	97	Jul. 13, 2016–Sep. 26 2016	97
	Magnitude	Jan. 15, 2014–May 28, 2015	41	May 28, 2015–Aug. 5, 2017	42
	Sundown	Dec. 27, 2015–Dec. 29, 2016	19	Dec. 29, 2016–May 7, 2017	20

5.4.1 Experimental Setup

Case Studies. We select diverse and common targets of detection systems for cyber security and prepare three datasets: malicious websites, malicious Android applications, and malicious PE files. Among these, the malicious website dataset was collected by ourselves to precisely analyze trained classifiers. The others are public datasets to objectively conduct evaluations. NN architectures are fully connected NNs and an RNN. These are selected to show that our methods can be applied to NNs for both non-structured and structured data.

Conventional Method. We compare our proposed methods with a baseline method to measure how much they improve DNN-based systems in terms of detection of unknown families. The baseline optimizes NNs by minimizing only the classification loss $\mathcal{L}_c$.

Hyperparameter Optimization. We optimize the hyperparameters by cross-validation using the training data. For cross-validation, we prepare different datasets, each containing validation data of one of the known families and training data of the other known families. The benign data are randomly split into halves for training and validation. We select the best hyperparameters in terms of a partial area under a receiver operating characteristic (ROC) curve (pAUC) in a region of false positive rates (FPRs) from 0 to a threshold. Since detection systems for cyber security commonly keep their FPRs under 0.1 [19, 122], we select 0.1 as the FPR threshold when we calculate a pAUC.

5.4.2 Malicious Website Case Study

In this case study, we use proxy logs to detect sequences of communications to malicious websites, which are created with exploit kits for drive-by download attacks.

Datasets. Benign proxy logs are collected from company networks with users' consent. To protect privacy, no information identifying users or companies is recorded. Malicious proxy logs are prepared using pcaps collected from Malware Traffic Analysis[1] and Broad Analysis[2]. Table 5.3 shows the numbers and periods of benign and malicious data. We prepare four datasets, each containing test data of one of four families and training data of the other three families.

Features. We design features referring to conventional systems for detecting drive-by download attacks [19, 72, 122]. We extract feature vectors representing a communication and then integrate vectors related to a sequence. Finally, we obtain feature vectors representing sequences of communications.

Features representing a communication are selected if they have been shown to be effective in previous work and are easy to implement. We consider implementability so that others can easily conduct follow-up research. Features are divided into two types: general and URL. General features are the interval, existence of identical communications, and combination of HTTP method and content-type. URL features include the presence of an IP address in the hostname, presence of a subdomain, popularity of top level domain (TLD), file types, structural features of the URL, and characters used in the URL.

We integrate the above features with different procedures depending on the data formats. For numeric features, we calculate their average and standard deviation. For Boolean features, we calculate their summation and average. For categorical features, we use their 1-grams and 2-grams. An integrated feature vector has 793 dimensions. The feature vectors are normalized before they are input into the NN so that the average and standard deviation of each feature become 0 and 1, respectively.

Neural Network Architecture. For F_s, F_c, and F_f, we use multi-layer NNs, each consisting of input, output, and one intermediate layer. All layers are fully connected. The activation functions for the last layers of F_c and F_f are softmax, and those for all other layers are rectified linear units (ReLU). We apply dropout to F_c to prevent overfitting and select Adam as the optimizer.

As a result of cross-validation, we select 10 for the number of neurons in intermediate layers,

[1] https://www.malware-traffic-analysis.net/
[2] http://www.broadanalysis.com/

Table 5.4: Classification performances in malicious website case study.

	Rig	Neutrino	Magnitude	Sundown
Baseline	0.9089	**0.9894**	0.4655	0.4905
$\text{FirOpt}_{\text{all}}$	0.8890	0.9892	0.3757	0.4590
$\text{FirOpt}_{\text{part}}$	**0.9183**	0.9890	**0.5833**	**0.5580**

Table 5.5: Calculation times in malicious website case study.

	Training (sec.)	Test (ms/data)
Baseline	477	0.772
$\text{FirOpt}_{\text{all}}$	654	0.734
$\text{FirOpt}_{\text{part}}$	652	0.759

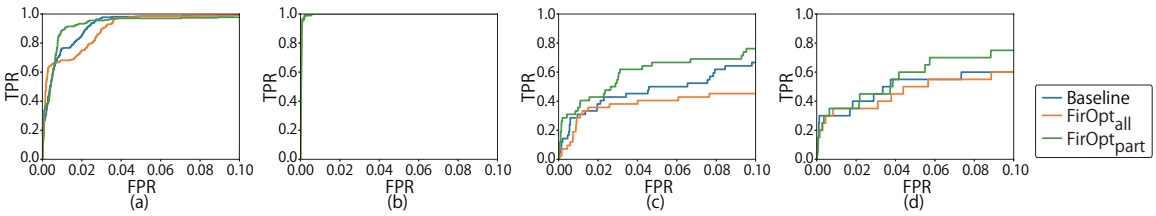


Figure 5.3: ROC curves in the malicious website case study. Families assumed to be unknown are (a) Rig, (b) Neutrino, (c) Magnitude, and (d) Sundown.

0.001 for the learning rate, 0.1 for the ratio of dropout, 100 for the batch size, 500 for the number of epochs, 0.01 for α, 1.5 for the parameter of smoothing a when Sundown is assumed as unknown, and 1.0 for a when the other families are assumed as unknown.

Classification Performance. Figure 5.3 shows ROC curves, and Table 5.4 shows pAUC in a region of FPRs from 0 to 0.1. We select 0.1 as the FPR threshold for a pAUC because detection systems for cyber security commonly keep their FPRs under 0.1 [19, 122]. $\text{FirOpt}_{\text{part}}$ achieves detection performance similar to or higher than the baseline if any family is assumed to be unknown. In other words, $\text{FirOpt}_{\text{part}}$ outperformed the baseline in terms of overall detection performance. In the best-case scenario, $\text{FirOpt}_{\text{part}}$ achieves a 19% higher TPR than the baseline at 3.2% FPR when Magnitude is assumed to be unknown. Unlike $\text{FirOpt}_{\text{part}}$, detection performance of $\text{FirOpt}_{\text{all}}$ is similar to or lower than the baseline in all cases.

Table 5.6: Features whose order of contributions largely increases (left: $\text{FirOpt}_{\text{all}}$, right: $\text{FirOpt}_{\text{part}}$).

Feature	Rise	Feature	Rise
Std of query length	705	**# of binary**	758
2-gram[others/image,GET/application]	46	**1-gram[GET/application]**	753
Std of number ratio	31	**Std of query length**	703
2-gram[POST/audio,POST/others]	30	**Std of upper case ratio**	696
2-gram[GET/binary,POST/video]	30	**Mean of path length**	142
2-gram[GET/binary,POST/multipart]	27	**Presence of subdomain**	42
2-gram[others/multipart,GET/video]	27	2-gram[others/image,others/binary]	27
2-gram[others/font,POST/video]	21	2-gram[GET/binary,POST/multipart]	20
2-gram[GET/video,GET/font]	18	2-gram[POST/multipart,POST/text]	11
2-gram[GET/text,POST/text]	16	**Std of URL length**	11

We calculate contributions of features to classification results. Many methods have been proposed for calculating the contribution of features in the research field of object recognition [9]. These methods are based on a gradient of a prediction. If a gradient with respect to a pixel in an image is large, the prediction is sensitive to the value of the pixel. In other words, the contribution of the pixel is large. When $\mathbf{x}$ is input to F, the contribution s_i of the i-th feature x_i is calculated as follows: $s_i = \left(\frac{\partial F(\mathbf{x})}{\partial x_i}\right)^2$. We use the average of the above contributions over all data because we need to calculate contributions with respect to all data not specific data.

We identify features whose order of contributions is largely increased with our methods as shown in Table 5.6. We compare the order with our methods and that with the baseline to investigate the effect of representation training. Features that often appear in malicious communications are written in bold. $\text{FirOpt}_{\text{part}}$ makes a classifier be based on many features appearing in malicious communications, but $\text{FirOpt}_{\text{all}}$ does not. The classifier trained with $\text{FirOpt}_{\text{all}}$ does not use features inherent across a *part* of families in classification, and thus those occasionally appearing in benign communications needed to be used. The classifier trained with $\text{FirOpt}_{\text{part}}$ uses features inherent across a *part* of families in classification, and thus those occasionally appearing in benign communications does not need to be used. As a result, the classifier trained with $\text{FirOpt}_{\text{part}}$ is superior to that with $\text{FirOpt}_{\text{all}}$ in terms of generalizability.

Calculation Time. Table 5.5 shows calculation times for training and test. Our methods need more time for training than the baseline because they calculate additional losses, i.e., family classification

Table 5.7: Android application dataset.

Training								Test	
Family	#	Family	#	Family	#	Family	#	Family	#
Benign	59,485	Droidkungfu	115	Fakenotify	35	Lotoor	20	Benign	25,495
Fakeinst	965	Basebridge	94	Batterydoctor	34	Kmin	16	Mobiletx	55
Opfake	468	Boxer	94	Pjapps	26	Placms	13	Iconosys	22
Ginmaster	223	Geinimi	74	Fakelogo	26	Dougalek	12	Spyset	13
Jifake	138	Bgserv	56	Steek	21	Rufraud	11	Smsspy	11

loss and family confusion loss, and update parameters of NNs on the basis of these losses. Even though our methods take longer to train, they are deployable to detection systems because attack tools are not frequently updated, e.g., less than once in three months [113]. Since the training time of our methods, i.e., 10 minutes, is much shorter than the interval of updates, detection systems can finish training with our methods much earlier before their detection ability degrades. All methods have similar test times because they conduct the same calculation: forward propagation through F_s and F_c. Therefore, the test phase presents no problem for deploying our methods.

5.4.3 Malicious Android Application Case Study

We use the Marvin dataset [63] for this case study. We select this dataset because it is large scale and available from the authors who developed Marvin. It includes features extracted from benign and malicious Android applications and their metadata such as hash values.

Dataset. Benign applications of the Marvin dataset are collected from Google Play Store[3] and confirmed to be benign with VirusTotal[4]. Malicious applications are randomly collected from VirusTotal. Since this dataset does not include family labels of malicious applications, we identify their family labels with AVClass [95]. Leveraging a large number of families and malicious samples in this dataset, we evaluate detection of families that are actually unknown at a certain point in time. Among applications of known families, we use those uploaded before Aug. 1st, 2012 as malicious training data. Malicious test data are applications uploaded after Aug. 1st, 2012 among applications of unknown families. Family labels and the number of samples in each family are

[3]https://play.google.com/
[4]https://www.virustotal.com/

Table 5.8: Classification performances in Android application case study.

	Mobiletx	Iconosys	Spyset	Smsspy
Baseline	0.0205	0.2259	0.0000	0.9927
$\text{FirOpt}_{\text{all}}$	0.0696	**0.2764**	0.0000	0.9927
$\text{FirOpt}_{\text{part}}$	**0.1275**	0.2427	0.0000	0.9900

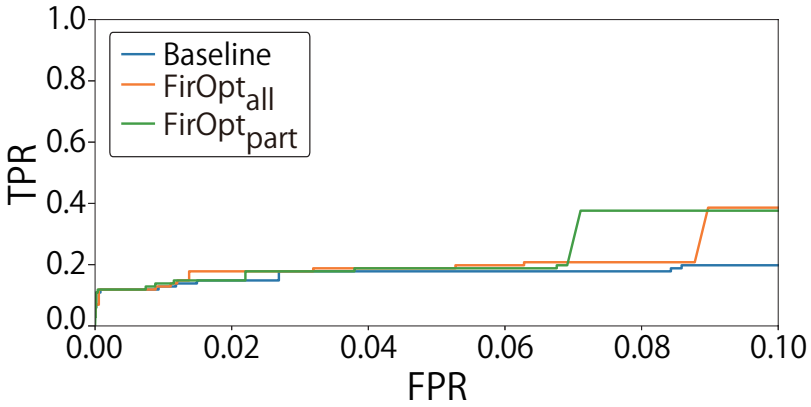


Figure 5.4: ROC curves in Android application case study.

shown in Table 5.7.

Features. The Marvin dataset includes 496,943 features that are designed to exhaustively extract malicious activities. Features are divided into static and dynamic analysis features. The static analysis features are extracted from a manifest file and Android Application Package (APK) file. Dynamic analysis features are extracted with a sandbox Andrubis [64]. Among these features, 8,539 effective features are selected as inputs to a classifier on the basis of F-score [20].

Neural Network Architecture. Features are extracted as a vector, not an image or sequence. Hence, we use a fully connected NN for this case study. For F_s, F_c, and F_f, we use multi-layer NNs, each consisting of input, output, and one intermediate layer. The activation functions for the last layers of F_c and F_f are softmax, and those for all other layers are ReLU. We apply dropout to F_c and select Adam as the optimizer.

As a result of cross-validation, we select 10 for the number of neurons in intermediate layers, 0.001 for the learning rate, 0.1 for the dropout rate, 500 for the batch size, 50 for the number of epochs, 0.01 for α, and 1.5 for the parameter of smoothing.

Table 5.9: Calculation times in malicious Android application case study.

	Training (sec.)	Test (ms/data)
Baseline	724	0.0273
$\text{FirOpt}_{\text{all}}$	1,490	0.0312
$\text{FirOpt}_{\text{part}}$	2,218	0.0295

Classification Performance. Table 5.8 shows pAUC of each method in a region of FPRs from 0 to 0.1. Both $\text{FirOpt}_{\text{part}}$ and $\text{FirOpt}_{\text{all}}$ achieve detection performance similar to or higher than the baseline without depending on unknown families. To compare overall detection performance, we also draw a ROC curve using all unknown families as shown in Fig. 5.4. $\text{FirOpt}_{\text{part}}$ outperforms $\text{FirOpt}_{\text{all}}$ and the baseline in terms of overall detection performance. Compared with the baseline, $\text{FirOpt}_{\text{part}}$ improves TPR by at most 19.8% at 7.0% FPR. In terms of detection of each family, the best method differs depending on unknown families. In terms of detection of Mobiletx, $\text{FirOpt}_{\text{part}}$ outperforms $\text{FirOpt}_{\text{all}}$ probably because the number of known families that produce similar behavior is small. In terms of detection of Iconosys, $\text{FirOpt}_{\text{all}}$ outperforms $\text{FirOpt}_{\text{part}}$ probably because the number of known families that produce similar behavior is relatively large. In terms of detection of Spyset, no method can detect it at all. This is because Spyset has an adware function, but no known family in this dataset has a similar function. In terms of detection of Smsspy, all methods accurately detected it. This is because Smsspy is a family that sends text messages to a remote site, and nine known families have this function, i.e., sending text messages.

Calculation Time. The evaluation results of calculation time are consistent with those of the malicious website case study. As shown in Table 5.9, training time does not cause any problem for deployment because it was much shorter than the update frequency of attack tools. Test times for this dataset are similar without depending on methods.

5.4.4 Malicious PE File Case Study

In the previous two case studies, we use fully connected NNs, but our proposed method can be applied to other NNs such as a convolutional NN (CNN) and RNN. To evaluate our method using other NNs, we use a dataset and NN of a previous study that proposes a system for detecting

Table 5.10: Dataset of malicious PE file case study.

Family	#	Family	#	Family	#
Benign (Training)	1,843	Artemis	130	Wisdomeyes	68
Benign (Test)	188	Eldorado	101	Scar	62
Dinwod	222	Zusy	73	Kazy	58

malicious PE files with an RNN [90].

Dataset. PE files dating from 2006 to 2017 are collected from four software sources: Softonic[5], PortableApps[6], SourceForge[7], and VirusTotal. These files are labeled with VirusTotal. The numbers of benign and malicious samples are shown in Table 5.10. This dataset does not include a large number of malicious samples, and thus we conduct an evaluation assuming that one family is unknown, the same as in the malicious website case study.

Features. To be robust to obfuscation and make an instant prediction, dynamic features are extracted by executing PE files in a short period. Dynamic features are 10 activities of a machine such as total processes, maximum process ID. All PE files are executed for 19 seconds with Cuckoo Sandbox[8], and the feature vectors are extracted every second. As a result, a sequence of feature vectors is obtained from a PE file.

Neural Network Architecture. The RNN consists of stacked multiple long-short-term memory (LSTM) cells and a fully connected layer. We use stacked LSTM cells as F_s and use fully connected NNs as F_c and F_f. Adam is selected as the optimizer.

For hyperparameters of F_s and F_c, we select the best ones shown in the paper of Rhode et al. [90]. The depth of LSTM cells is 2, the number of neurons in hidden layers is 195, the number of epochs is 39, the dropout rate is 0.1, the weight regularization is $l1$, the batch size is 64, the number of layers in F_c is 2, the activation function of the last layer in F_c is sigmoid, and that of intermediate layers is ReLU.

The other hyperparameters are selected on the basis of cross-validation. The number of layers

[5]https://en.softonic.com/
[6]https://portableapps.com/
[7]https://sourceforge.net/
[8]https://cuckoosandbox.org/

Table 5.11: Classification performances in malicious PE file case study.

	Dinwod	Artemis	Eldorado	Zusy	Wisdomeyes	Scar	Kazy
Baseline	**0.9900**	0.8185	0.8514	0.8460	0.8504	0.7555	0.8955
$\text{FirOpt}_{\text{all}}$	0.9891	0.8414	**0.8851**	**0.8541**	**0.8838**	0.7179	0.9069
$\text{FirOpt}_{\text{part}}$	0.9889	**0.8450**	0.8737	0.8499	0.8815	**0.7782**	**0.9307**

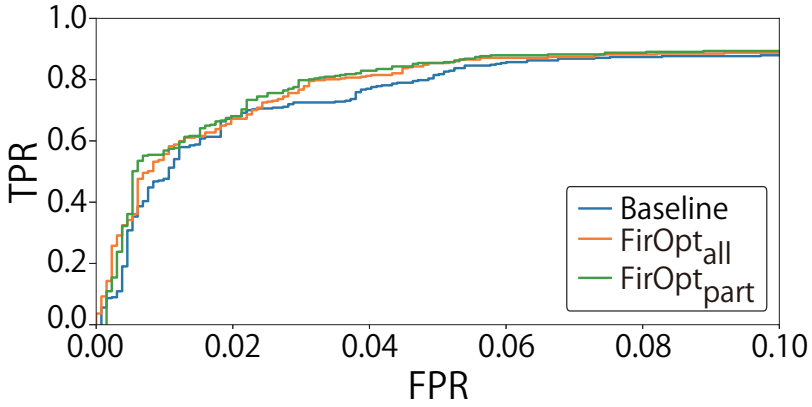


Figure 5.5: ROC curves in malicious PE file case study.

Table 5.12: Calculation times in malicious PE file case study.

	Training (sec.)	Test (ms/data)
Baseline	515	2.68
$\text{FirOpt}_{\text{all}}$	915	2.97
$\text{FirOpt}_{\text{part}}$	909	2.98

in F_f is 3, the activation function of the last layer in F_f is softmax, and that of an intermediate layer in F_f is ReLU. The parameters of smoothing are 1.0 for Dinwod, 5.0 for Artemis, 2.0 for Eldorado, 5.0 for Zusy, 2.0 for Wisdomeyes, 0.0 for Scar, and 0.0 for Kazy.

Classification Performance. Table 5.11 shows pAUC of each method in a region of FPRs from 0 to 0.1. $\text{FirOpt}_{\text{part}}$ achieves detection performance similar to or higher than the baseline without depending on unknown families. On the other hand, $\text{FirOpt}_{\text{all}}$ does not outperform the baseline when Scar is assumed to be unknown. This is probably because features inherent across a *part* of families are needed to detect Scar. To compare overall classification performance, we draw a ROC curve using all unknown families. Figure 5.5 shows that $\text{FirOpt}_{\text{part}}$ outperforms the other methods. Compared with the baseline, $\text{FirOpt}_{\text{part}}$ improves TPR by at most 7.4% at 0.7% FPR.

Calculation Time. Table 5.12 shows calculation times for training and test. We obtain results consistent with the other two case studies. Training times of our methods are longer than that of the baseline but much shorter than the update frequency of attack tools. Test times of all methods are similar.

5.5 Discussion

Applicability of Proposed Methods. Our methods can be applied to datasets and NN architectures other than three case studies in the evaluation. Our methods can be applied to any malicious data created with attack tools such as Internet of Things (IoT) malware [5] and malicious JavaScript [25]. Our methods can also be applied to any NN architecture such as a CNN [40] and graph CNN [29].

Although $\text{FirOpt}_{\text{part}}$ does not always significantly outperform the baseline in terms of detection of each unknown family, it achieved similar to or higher than the baseline without depending on unknown families. In other words, $\text{FirOpt}_{\text{part}}$ improves overall detection performance without degrading detection of each family. Therefore, these results do not lower its applicability.

Validity of Evaluation. In the two case studies, i.e., malicious website and PE file case studies, we use one family as test data assuming that it is unknown. Even in these case studies, we should ideally evaluate the detection of malicious data produced by newly developed attack tools. In our evaluation, we show that $\text{FirOpt}_{\text{part}}$ improved overall detection of actual unknown families in the malicious Android case study. Furthermore, evaluation results are consistent across three case studies without depending on datasets, and NN architectures. Therefore, $\text{FirOpt}_{\text{part}}$ is definitely expected to improve overall detection of malicious data created with newly developed attack tools.

We evaluated our methods in comparison to the baseline, which is an NN optimized in terms of the classification loss. Detection of unknown families also depends on extracted features and NN architectures. However, the purpose of our methods is to improve existing DNN-based systems in terms of detection of unknown families. In other words, design of features and NN architectures is out of the focus of this Chapter. For this reason, we compare our methods with an optimization method on the basis of the classification loss.

Family Label Collection. Our methods need family labels as well as class labels for training. Family labels can be identified on the basis of labels of intrusion detection system or anti-virus engines [83, 95]. These labels have been shown to sometimes be wrong at the time of the first detection and corrected a few weeks later [50]. This causes a problem if our methods need newly detected data of known families. In fact, the effectiveness of our methods is evaluated by using test data collected for at least three months in Section 5.4. Delay until obtaining reliable family labels is much shorter than the period for collecting test data. For this reason, our methods are expected to achieve sufficient detection performance if we use data whose reliable family labels can be obtained for training.

5.6 Related Work

Detection Systems Leveraging DNNs. Many researchers have proposed sophisticated detection systems leveraging DNNs. Examples include malicious Android application detection [71, 129] and malicious PE file detection [90]. However, these systems are not focused on detecting unknown families. A method has been proposed for extracting invariant features [11]. This method designed invariant features on the basis of a hypothesis of changes in malicious data. On the other hand, our purpose is to improve existing DNN-based systems in terms of detection of unknown families, and thus we use previously proposed features.

Classification Under Changing Data Distributions. To prevent a classifier from degrading classification performance under changes in data distributions, a method has been proposed for detecting concept drift [49]. It uses a statistical comparison of training data with test data to detect concept drift. If a change is detected, a classifier is retrained by using a new distribution of data (known and unknown families in our setting). However, our purpose is to improve detection of unknown families without detecting concept drift and retraining classifiers.

Domain Adaptation. To accurately classify an unlabeled dataset (target domain), a labeled dataset (source domain) are used in domain adaptation. Domain adaptation methods optimize representations so that they consist of features inherent across the source and target domains (domain-invariant

representations) [2]. In this Chapter, we use only known families for training and do not use a target of detection, i.e., unknown families. Furthermore, we relax the constraint of representation learning studied in domain adaptation and define the constraint of $\text{FirOpt}_{\text{part}}$. While domain adaptation methods optimize representations so that they consist of features inherent across all domains (i.e., the source and target domains), $\text{FirOpt}_{\text{part}}$ includes features inherent across a *part* of families in representations.

5.7 Summary

To improve existing DNN-based systems in terms of detection of unknown families, we propose two optimization methods for building a classifier that prioritizes family-invariant features. One is $\text{FirOpt}_{\text{all}}$: a method optimizing representations so that they consist of features inherent across *all* known families. The other is $\text{FirOpt}_{\text{part}}$: a method improving robustness of $\text{FirOpt}_{\text{all}}$ by including features inherent across a *part* of known families in representations as well as features inherent across *all* known families. We evaluate whether our methods robustly improve DNN-based systems in terms of detection of unknown families with three case studies using three datasets and two neural network architectures. Evaluation results are consistent across three case studies: $\text{FirOpt}_{\text{part}}$ outperforms $\text{FirOpt}_{\text{all}}$ and the baseline in terms of overall detection of unknown families. Although the best method differs depending on unknown families, $\text{FirOpt}_{\text{part}}$ achieves detection of unknown families similar to or higher than the baseline without depending on unknown families, datasets, and NN architectures. On the other hand, $\text{FirOpt}_{\text{all}}$ does not outperform the baseline in terms of detection of some unknown families in the malicious website and portable executable file case studies. These results show that the robustness of $\text{FirOpt}_{\text{part}}$ is induced by including features inherent across a *part* of known families in the representations.

Chapter 6

Conclusion

To improve the effectiveness of network-based detection, we proposed systems for minimizing negative effects of anti-analysis techniques on malicious communication collection and a method for detecting dynamically changing attacks even when some malicious communications cannot be collected by anti-analysis techniques. Specifically, we focused on unknown families, which are new types of malicious websites or malware samples.

In Chapter 2, we proposed a system for detecting malicious websites without collecting all malicious data. Even if we cannot observe some malicious data, such as exploit code or malware, we can always observe compromised websites into which attackers inject redirection code to malicious data. Since attackers use search engines to automatically discover vulnerable websites, compromised websites have similar traits. We therefore built a classifier by leveraging both malicious and compromised websites. Specifically, we converted all websites observed during an access into a redirection graph whose vertices are URLs and edges are redirections between two URLs, and classified it with graph mining. To perform this classification, we integrated similarities between the redirection graph's subgraphs and redirection subgraphs shared across malicious, benign, and compromised websites. This system enhanced the exhaustiveness of collected malicious websites and improved detection capabilities of network-based pre-infection detection.

In Chapter 3, we proposed a system for detecting communications to malicious websites from simple logs such as proxy logs. We focused on sequences of destination URLs, because some

artifacts of malicious redirections can be extracted from simple logs by considering several nearby URLs. Specifically, simple logs contain malicious landing, redirection, and exploit URLs with their sequential order preserved. We call these *URL sequences*, and URL sequences including accesses to malicious websites *malicious URL sequences*. To find an effective approach for classifying URL sequences, we compared three approaches: an individual-based approach, a convolutional neural network (CNN), and a novel event de-noising CNN (EDCNN). By using detected malicious communications to create blacklists, regular expression signatures, and machine learning classifiers, we improved detection capabilities of network-based pre-infection detection.

In Chapter 4, we proposed a system for efficiently collecting HTTP requests through dynamic malware analysis. Specifically, our system analyzes a malware sample over short periods, then determines whether the analysis should be continued or suspended. This determination is made on the basis of network behavior observed in the short-period analyses. To make accurate determinations, we focused on the fact that malware communications resemble natural language from the viewpoint of data structure. We applied recursive neural networks, which have recently exhibited high classification performance in the field of natural language processing. Our system improved efficiency of dynamic malware analysis by suspending analyses of malware that do not disclose malicious behavior because of the analysis environment. We improved network-based post-infection detection by using the collected HTTP requests for creating blacklists, regular expression signatures, and machine learning classifiers.

In Chapter 5, we investigated how to improve existing DNN-based systems in terms of detecting unknown families. Unknown families can be quite difficult to detect when their features completely differ from those of known families. Even so, some features are inherent across families because malicious data include similar code or produce similar behaviors to exploit vulnerabilities or to cost-effectively achieve a successful attack. When features inherent across known families (family-invariant features) are prioritized in classification, unknown families become easier to detect. We therefore aimed at building a classifier that prioritizes family-invariant features. Evaluation results showed that our method robustly improves DNN-based systems without depending on datasets. This method can improve network-based detection and compensate for degradation caused by anti-analysis techniques.

In the future, more and more detection systems will be proposed with the increasing diversity of attacks. This increase in the number of systems will produce new research problems related to computational costs and false positives. To maintain high detection capabilities with acceptable computational costs and false positive rates, we will further research how to optimize combinations of detection systems for multi-layered defense.

Bibliography

[1] Lawrence Abrams. New trickbot version focuses on microsoft's windows defender. https://www.bleepingcomputer.com/news/security/new-trickbot-version-focuses-on-microsofts-windows-defender/. Accessed: September 25th, 2019.

[2] Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, and Mario Marchand. Domain-adversarial neural networks. *arXiv preprint arXiv:1412.4446*, 2014.

[3] Mitsuaki Akiyama, Makoto Iwamura, and Yuhei Kawakoya. Design and implementation of high interaction client honeypot for drive-by-download attacks. *IEICE Transactions on Communications*, Vol. 93, No. 5, pp. 1131–1139, 2010.

[4] Alexa Internet, Inc. The top 500 sites on the web. https://www.alexa.com/topsites. Accessed: September 25th, 2019.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *Proceedings of the 26th USENIX Security Symposium*, pp. 1092–1110, 2017.

[6] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for dns. In *Proceedings of the 19th USENIX Security Symposium*, pp. 273–290, 2010.

[7] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou II, and David Dagon. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX Security Symposium*, pp. 1–16, 2011.

[8] Kazufumi Aoki, Takeshi Yagi, Makoto Iwamura, and Mitsutaka Itoh. Controlling malware http communications in dynamic analysis system using search engine. In *Proceedings of the 3rd International Workshop on Cyberspace Safety and Security*, pp. 1–6, 2011.

[9] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. *Journal of Machine Learning Research*, Vol. 11, No. Jun, pp. 1803–1831, 2010.

[10] Gabriel Negreira Barbosa and Rodrigo Rubira Branco. Prevalent characteristics in modern malware. In *BlackHat USA*, 2014.

[11] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *Proceedings of the 25th USENIX Security Symposium*, pp. 807–822, 2016.

[12] Ulrich Bayer, Engin Kirda, and Christopher Kruegel. Improving the efficiency of dynamic malware analysis. In *Proceedings of the 25th ACM Symposium on Applied Computing*, pp. 1871–1878, 2010.

[13] BBC News Services. Coincheck: World's biggest ever digital currency 'theft'. `https://www.bbc.com/news/world-asia-42845505`. Accessed: September 25th, 2019.

[14] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium*, 2011.

[15] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pp. 74–81, 2005.

[16] Riccardo Bortolameotti, Thijs van Ede, Marco Caselli, Maarten H Everts, Pieter Hartel, Rick Hofstede, Willem Jonker, and Andreas Peter. Decanter: Detection of anomalous outbound http traffic by passive application fingerprinting. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 373–386, 2017.

[17] Leo Breiman. Random forests. *Machine learning*, Vol. 45, No. 1, pp. 5–32, 2001.

[18] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: the commoditization of malware distribution. In *Proceedings of the 20th USENIX Security Symposium*, pp. 187–202, 2011.

[19] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web*, pp. 197–206, 2011.

[20] Yi-Wei Chen and Chih-Jen Lin. *Combining SVMs with various feature selection strategies*, pp. 315–324. 2006.

[21] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. Dns noise: Measuring the pervasiveness of disposable domains in modern dns traffic. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 598–609. IEEE, 2014.

[22] 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. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA*, pp. 758–765, 2015.

[23] Charles Cooper. Wannacry: Lessons learned 1 year later. `https://www.symantec.com/blogs/feature-stories/wannacry-lessons-learned-1-year-later`. Accessed: September 25th, 2019.

[24] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 281–290, 2010.

[25] Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *Proceedings of the 20th USENIX Security Symposium*, pp. 33–48, 2011.

[26] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 2702–2711, 2016.

[27] Hung Dang, Yue Huang, and Ee-Chien Chang. Evading classifiers by morphing in the dark. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, pp. 119–133, 2017.

[28] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pp. 253–262, 2004.

[29] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 29th Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

[30] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

[31] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys*, Vol. 44, No. 2, p. 6, 2012.

[32] Holy Father. Hooking windows api-technics of hooking api functions on windows. *The Assembly-Programming-Journal*, Vol. 2, No. 2, 2004.

[33] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. *Learning Theory and Kernel Machines*, Vol. 2777 of Lecture Notes in Computer Science, pp. 129–143, 2003.

[34] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pp. 821–832, 2012.

[35] Kent Griffin, Scott Schneider, Xin Hu, and Tzi-Cker Chiueh. Automatic generation of string signatures for malware detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion detection*, pp. 101–120, 2009.

[36] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, pp. 139–154, 2008.

[37] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium*, 2008.

[38] Shuang Hao, Alex Kantchelian, Brad Miller, Vern Paxson, and Nick Feamster. Predator: proactive recognition and elimination of domain abuse at time-of-registration. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*, pp. 1568–1579, 2016.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proceedings of the 13th European Conference on Computer Vision*, pp. 346–361, 2014.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the 29th IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[41] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the 2008 Network and Distributed System Security Symposium*, 2008.

[42] Honeynet Project. Capture-hpc. `https://www.honeynet.org/projects/old/capture-hpc/`. Accessed: September 25th, 2019.

[43] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Proceedings of the 27th Advances in Neural Information Processing Systems*, pp. 2042–2050, 2014.

[44] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, Vol. 5, No. 3, pp. 299–314, 1996.

[45] Information Security Office. Cleaning an infected computer of malware. `https://security.berkeley.edu/education-awareness/best-practices-how-tos/compromised-systems/cleaning-infected-computer-malware`. Accessed: September 25th, 2019.

[46] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. Nazca: Detecting malware distribution in large-scale networks. In *Proceedings of the 21st Network and Distributed System Security Symposium*, 2014.

[47] Kyriakos K. Ispoglou and Mathias Payer. malwash: Washing malware to evade dynamic analysis. In *Proceedings of the 10th USENIX Workshop on Offensive Technologies*, 2016.

[48] Ashish Jadhav, Deepti Vidyarthi, and M Hemavathy. Evolution of evasive malwares: A survey. In *Proceedings of the 2016 International Conference on Computational Techniques in Information and Communication Technologies*, pp. 641–646, 2016.

[49] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *Proceedings of the 26th USENIX Security Symposium*, pp. 625–642, 2017.

[50] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pp. 45–56, 2015.

[51] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *Proceedings of the 22nd USENIX Security Symposium*, pp. 637–652, 2013.

[52] Kyungtae Kim, I Luk Kim, Chung Hwan Kim, Yonghwi Kwon, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu. J-force: Forced execution on javascript. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 897–906, 2017.

[53] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[54] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: bare-metal analysis-based evasive malware detection. In *Proceedings of the 23rd USENIX Security Symposium*, pp. 287–301, 2014.

[55] Christoph Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: Decloaking internet malware. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, pp. 443–457, 2012.

[56] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th USENIX Security Symposium*, pp. 351–366, 2009.

[57] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring PUP prevalence and PUP distribution through pay-per-install services. In *Proceedings of the 25th USENIX Security Symposium*, pp. 739–756, 2016.

[58] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitras. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1118–1129, 2015.

[59] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[60] Zhou Li, Sumayah Alrwais, XiaoFeng Wang, and Eihal Alowaisheq. Hunting the red fox online: Understanding and detection of mass redirect-script injections. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*, pp. 3–18, 2014.

[61] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pp. 674–686, 2012.

[62] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *Proceedings of the 14th Recent Advances in Intrusion Detection*, pp. 338–357, 2011.

[63] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *Proceedings of the 39th IEEE Annual Computer Software and Applications Conference*, pp. 422–433, 2015.

[64] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van Der Veen, and Christian Platzer. Andrubis–1,000,000 apps later: A view on

current android malware behaviors. In *Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pp. 3–17, 2014.

[65] Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pp. 440–450, 2010.

[66] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1245–1254, 2009.

[67] Malwarebytes. hphosts. `http://www.hosts-file.net/`. Accessed: September 25th, 2019.

[68] MalwareBytes. hphosts. `https://hosts-file.net/?s=Download`. Accessed: September 25th, 2019.

[69] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, pp. 431–441, 2007.

[70] Lorenzo Martignoni, Elizabeth Stinson, Matt Fredrikson, Somesh Jha, and John C Mitchell. A layered architecture for detecting malicious behaviors. In *Proceedings of the 11th International Symposium Recent Advances in Intrusion Detection*, pp. 78–97, 2008.

[71] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail Joon Ahn. Deep android malware detection. In *Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy*, pp. 301–308, 2017.

[72] Hesham Mekky, Ruben Torres, Zhi-Li Zhang, Sabyasachi Saha, and Antonio Nucci. Detecting malicious http redirections using trees of user browsing activity. In *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications*, pp. 1159–1167, 2014.

[73] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*, pp. 1009–1024, 2017.

[74] Aziz Mohaisen, Andrew G West, Allison Mankin, and Omar Alrawi. Chatter: Classifying malware families using system event ordering. In *Proceedings of the 2014 IEEE Conference on Communications and Network Security*, pp. 283–291, 2014.

[75] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, Vol. 5, No. 1, pp. 32–38, 1957.

[76] Saeed Nari and Ali A Ghorbani. Automated malware classification based on network behavior. In *Proceedings of the 2013 International Conference on Computing, Networking and Communications*, pp. 642–647, 2013.

[77] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *Proceedings of the 22nd USENIX Security Symposium*, pp. 589–604, 2013.

[78] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Webwitness: investigating, categorizing, and mitigating malware download paths. In *Proceedings of the 24th USENIX Security Symposium*, pp. 1025–1040, 2015.

[79] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Towards measuring and mitigating social engineering software download attacks. In *Proceedings of the 25th USENIX Security Symposium*, pp. 773–789, 2016.

[80] Matthias Neugschwandtner, Paolo Milani Comparetti, Gregoire Jacob, and Christopher Kruegel. Forecast: skimming off the malware cream. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pp. 11–20, 2011.

[81] Nielsen Norman Group. How long do users stay on web pages? `https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/`. Accessed: September 25th, 2019.

[82] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 2014–2023, 2016.

[83] Open Information Security Foundation. Suricata. `https://suricata-ids.org/`. Accessed: September 25th, 2019.

[84] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. *arXiv preprint arXiv:1807.07838*, 2018.

[85] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, pp. 391–404, 2010.

[86] Niels Provos, Panayiotis Mavrommatis, M. A. Rajab, and Fabian Monrose. All your iframes point to us. In *Proceedings of the 17th USENIX Security Symposium*, pp. 1–16, 2008.

[87] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[88] Moheeb Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. Trends in circumventing web-malware detection. *Google Technical Report*, 2011.

[89] Jason DM Rennie and Nathan Srebro. Loss functions for preference levels: Regression with discrete ordered labels. In *Proceedings of the IJCAI Multidisciplinary Workshop on Advances in Preference Handling*, pp. 180–186, 2005.

[90] Matilda Rhode, Pete Burnap, and Kevin Jones. Early-stage malware prediction using recurrent neural networks. *Computers & Security*, Vol. 77, pp. 578–594, 2018.

[91] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pp. 31–39, 2010.

[92] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pp. 289–300, 2006.

[93] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. Malware detection based on mining api calls. In *Proceedings of the 25th ACM Symposium on Applied Computing*, pp. 1020–1025, 2010.

[94] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *Proceedings of the 11 the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 192–211, 2014.

[95] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 230–253, 2016.

[96] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, Vol. 12, pp. 2539–2561, 2011.

[97] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pp. 488–495, 2009.

[98] Toshiki Shibahara, Daiki Chiba, Mitsuaki Akiyama, Kunio Hato, Daniel Dalek, and Masayuki Murata. Unknown family detection based on family-invariant representation. In *NeurIPS 2018 Workshop on Security in Machine Learning*, 2018.

[99] Toshiki Shibahara, Daiki Chiba, Mitsuaki Akiyama, Kunio Hato, and Masayuki Murata. Unknown family detection using adversarial training. *Computer Security Symposium*, Vol. 2018, No. 2, pp. 348–355, 2018 (in Japanese).

[100] Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, Kunio Hato, and Masayuki Murata. Evasive malicious website detection by leveraging redirection subgraph similarities. *IEICE Transactions on Information and Systems*, Vol. 102, No. 3, pp. 430–443, 2019.

[101] Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, and Takeshi Yada. Detecting malicious websites by integrating malicious, benign, and compromised redirection subgraph similarities. In *Proceedings of the 41st IEEE International Conference on Computers, Software and Applications*, Vol. 1, pp. 655–664, 2017.

[102] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Kunio Hato. Efficient dynamic malware analysis for collecting http requests using deep learning. *IEICE Transactions on Information and Systems*, Vol. 102, No. 4, pp. 725–736, 2019.

[103] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Daiki Chiba, and Takeshi Yada. Efficient dynamic malware analysis based on network behavior using deep learning. In *Proceedings of the 59th Annual IEEE Global Communications Conference*, pp. 1–7, 2016.

[104] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Yuta Takata, and Takeshi Yada. Poster: Detecting malicious web pages based on structural similarity of redirection chains. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, pp. 1671–1673, 2015.

[105] Toshiki Shibahara, Takeshi Yagi, Mitsuaki Akiyama, Yuta Takata, and Takeshi Yada. Detecting malicious web pages based on structural similarity of redirection chains. *Computer Security Symposium*, Vol. 2015, No. 3, pp. 496–503, 2015 (in Japanese).

[106] 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. In *Proceedings of the 2017 IEEE International Conference on Communications*, pp. 1–7, 2017.

[107] Toshiki Shibahara, Kohei Yamanishi, Yuta Takata, Daiki Chiba, Taiga Hokaguchi, Mitsuaki Akiyama, Takeshi Yagi, Yuichi Ohsita, and Masayuki Murata. Event de-noising convolutional neural network for detecting malicious url sequences from proxy logs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 101, No. 12, pp. 2149–2161, 2018.

[108] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *Proceedings of the 24th USENIX Security Symposium*, pp. 611–626, 2015.

[109] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software*. No Starch Press, 2012.

[110] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 129–136, 2011.

[111] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, p. 1642, 2013.

[112] Kyle Soska and Nicolas Christin. Automatically detecting vulnerable websites before they turn malicious. In *Proceedings of the 23rd USENIX Security Symposium*, pp. 625–640, 2014.

[113] Ben Stock, Benjamin Livshits, and Benjamin Zorn. Kizzle: A signature compiler for detecting exploit kits. In *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 455–466, 2016.

[114] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, pp. 133–144, 2013.

[115] Andrew H Sung, Jianyun Xu, Patrick Chavez, and Srinivas Mukkamala. Static analyzer of vicious executables (save). In *Proceedings of the 20th Annual Computer Security Applications Conference*, pp. 326–334, 2004.

[116] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

[117] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, Vol. 9, No. 3, pp. 293–300, 1999.

[118] Symantec. Internet security threat report. `https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf`. Accessed: September 25th, 2019.

[119] Symantec. Internet security threat report. `https://www.symantec.com/security-center/threat-report`. Accessed: September 25th, 2019.

[120] Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, and Shigeki Goto. Website forensic investigation to identify evidence and impact of compromise. In *Proceedings of the 12th International Conference on Security and Privacy in Communication Networks*, 2016.

[121] Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, Takeo Hariu, and Shigeki Goto. Minespider: extracting hidden urls behind evasive drive-by download attacks. *IEICE Transactions on Information and Systems*, Vol. 99, No. 4, pp. 860–872, 2016.

[122] Teryl Taylor, Xin Hu, Ting Wang, Jiyong Jang, Marc Ph Stoecklin, Fabian Monrose, and Reiner Sailer. Detecting malicious exploit kits using tree-based similarity searches. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*, pp. 255–266, 2016.

[123] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems in the 29th Annual Conference on Neural Information Processing Systems*, 2015.

[124] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, and P. G. Bringas. Sok: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, pp. 659–673, 2015.

[125] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G Bringas. Sok: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 659–673, 2015.

[126] Vladimir Vapnik, Steven E Golowich, and Alex J Smola. Support vector method for function approximation, regression estimation and signal processing. In *Proceedings of the 10th Advances in Neural Information Processing Systems*, pp. 281–287, 1997.

[127] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *Proceedings of the 28th IEEE Symposium on Security and Privacy*, Vol. 5, No. 2, pp. 32–39, 2007.

[128] www.malwaredomainlist.com. Malware domain list. `http://www.malwaredomainlist.com/mdl.php`. Accessed: September 25th, 2019.

[129] Ke Xu, Yingjiu Li, Robert H Deng, and Kai Chen. Deeprefiner: Multi-layer android malware detection system applying deep neural networks. In *Proceedings of the 3rd IEEE European Symposium on Security and Privacy*, pp. 473–487, 2018.

[130] Babak Yadegari, Brian Johannesmeyer, Ben Whitely, and Saumya Debray. A generic approach to automatic deobfuscation of executable code. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, pp. 674–691, 2015.

[131] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.

[132] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pp. 199–208, 2013.

[133] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pp. 297–300, 2010.

[134] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[135] Junjie Zhang, Christian Seifert, Jack W Stokes, and Wenke Lee. Arrow: Generating signatures to detect drive-by downloads. In *Proceedings of the 20th International Conference on World Wide Web*, pp. 187–196, 2011.