IEICE TRANS. COMMUN., VOL.E99–B, NO.5 MAY 2016

PAPER *Special Section on Internet Architectures and Management Methods that Enable Flexible and Secure Deployment of Network Services*

BotProfiler: Detecting Malware-Infected Hosts by Profiling Variability of Malicious Infrastructure*

Daiki CHIBA[†,††a)], **Takeshi YAGI**[†], **Mitsuaki AKIYAMA**[†], **Kazufumi AOKI**[†], **Takeo HARIU**[†], ***and*** **Shigeki GOTO**[††], ***Members***

SUMMARY Ever-evolving malware makes it difficult to prevent it from infecting hosts. Botnets in particular are one of the most serious threats to cyber security, since they consist of a lot of malware-infected hosts. Many countermeasures against malware infection, such as generating network-based signatures or templates, have been investigated. Such templates are designed to introduce regular expressions to detect polymorphic attacks conducted by attackers. A potential problem with such templates, however, is that they sometimes falsely regard benign communications as malicious, resulting in false positives, due to an inherent aspect of regular expressions. Since the cost of responding to malware infection is quite high, the number of false positives should be kept to a minimum. Therefore, we propose a system to generate templates that cause fewer false positives than a conventional system in order to achieve more accurate detection of malware-infected hosts. We focused on the key idea that malicious infrastructures, such as malware samples or command and control, tend to be reused instead of created from scratch. Our research verifies this idea and proposes here a new system to profile the variability of substrings in HTTP requests, which makes it possible to identify invariable keywords based on the same malicious infrastructures and to generate more accurate templates. The results of implementing our system and validating it using real traffic data indicate that it reduced false positives by up to two-thirds compared to the conventional system and even increased the detection rate of infected hosts.

key words: *malware, botnet, dynamic analysis, template*

1. Introduction

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

C&C is an essential function of a botnet, i.e., C&C communications from bots must occur in a network. C&C communications have involved various protocols such as IRC, HTTP, P2P, and HTTP+P2P [2]. Recently, attackers have tended to use more general protocols for their C&C communications to prevent them from being analyzed or detected. Thus, over 60% of botnets use HTTP or HTTP+P2P as their C&C protocol [3]. In enterprise networks, in particular, filtering for outbound traffic is applied to limit their protocol only to HTTP and HTTPS. Therefore, using HTTP as a C&C protocol is effective for attackers.

Countermeasures against botnet consist of defeating botnets themselves, referred to as a *takedown*, or detecting bots or C&C communications in a botnet. For example, takedowns of the Zeus botnet, the Citadel botnet, and the GameOver Zeus botnet were conducted in 2012, 2013, and 2014, respectively [4]–[6]. Takedowns can help prevent cyber attacks. However, they are not easily implemented because it is necessary to keep precise track of C&C infrastructure and to ensure cooperation among relevant organizations.

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

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

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

Manuscript received August 27, 2015.
Manuscript revised November 20, 2015.

[†]The authors are with NTT Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.
[††]The authors are with Waseda University, Tokyo, 169-8555 Japan.

*An earlier version of this paper appeared in [1].

a) E-mail: chiba.daiki@lab.ntt.co.jp

DOI: 10.1587/transcom.2015AMP0001

botnets [7]. Perdisci et al. proposed a method of generating signatures with regular expressions to detect the URLs used in C&C communications based on HTTP traffic captured in a controlled environment, which is a sandbox system to dynamically analyze malware samples [8]. Nelms et al. improved upon the above research [8] and proposed a system called ExecScent, which introduced the concept of *templates* that can cover not only URLs but also HTTP request headers [9].

However, a potential problem with such signatures or templates is that they may falsely regard benign communications as malicious, resulting in false positives, due to an inherent aspect of regular expressions. Given that the cost of dealing with malware infection is high, false positives should be kept to a minimum. The cost can be enormous in organizations. A study showed that an average of 395 hours a week is spent responding to false alerts of malware infection, which costs about $1.27 million per year [10].

We therefore propose a system, called BotProfiler, to generate templates that cause fewer false positives than with the conventional system ExecScent [9] in order to achieve more accurate detection of malware-infected hosts. We focused on the key idea that malicious infrastructures, such as malware and C&C, are prone to be reused instead of created from scratch. Our research verifies this idea and proposes here BotProfiler to profile the variability of substrings in HTTP requests. BotProfiler identifies invariable keywords based on the same malicious infrastructures and makes it possible to generate more accurate templates.

Our main contributions are as follows:

- We propose a system called BotProfiler that generates templates to detect infected hosts after profiling invariable substrings of URL paths, URL queries, and user agents in HTTP requests from infected hosts.
- Our research presents the first-ever analysis of the existence and its reason of invariable substrings used by attackers and contributes to automatically generating more accurate and valuable templates than ExecScent.
- The effectiveness of BotProfiler was validated with actual large traffic datasets. The results indicated that it reduced the number of false positives by up to two-thirds compared to ExecScent and even increased the detection rate of infected hosts.

The rest of this paper is organized as follows. The detection methodology of BotProfiler is introduced in Sect. 2. The datasets and the results of our evaluation are described in Sect. 3. The limitation of our system is discussed in Sect. 4. Section 5 reviews related work. Finally, Sect. 6 concludes the paper.

2. Detection Methodology

2.1 System Overview

BotProfiler generates templates to detect infected hosts in a network. Figure 1 is an overview of the system. It

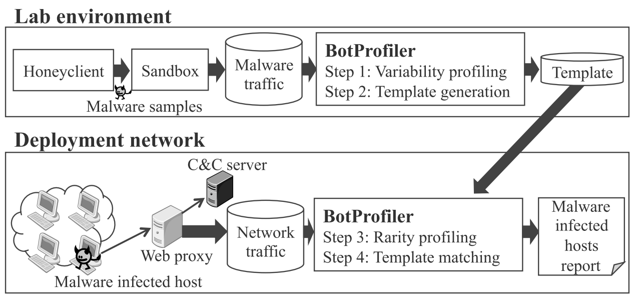


Fig. 1 BotProfiler system overview.

involves four steps; step 1: variability profiling, step 2: template generation, step 3: rarity profiling, and step 4: template matching. This matching concept was originally introduced with the conventional system ExecScent; however, BotProfiler generates more precise and valuable templates than ExecScent. Steps 1 and 2 involve generating templates from outbound traffic captured in our sandbox system [11] running malware samples (malware traffic). Our malware samples were obtained from our high-interaction honeyclient [12], [13]. Steps 3 and 4 involve matching traffic with templates based on two criteria: the similarity to the templates and the *rarity* of each element in the templates. For example, an element that has high rarity means that it appears very infrequently in a deployment network. BotProfiler determines the rarity to use the characteristics of modern malware samples, which tend to affect a limited percentage of all hosts in a network, and to reduce false positives. Figure 1 also shows that the architecture of BotProfiler is divided into a lab environment and deployment network. The lab environment requires a honeyclient and a sandbox; however, the deployment network does not require them and only uses templates from the lab environment. This architecture readily enables us to deploy BotProfiler in multiple deployment networks. The details of BotProfiler are explained step by step in the following sections.

2.2 Step 1: Variability Profiling

Step 1 enables us to generate templates with regular expressions that cause fewer false positives than ExecScent. We focused on the key idea that malicious infrastructures, such as malware and C&C, tend to be reused instead of created from scratch. On the basis of this idea, in step 1, the variability of substrings in malware-generated HTTP requests is profiled to identify *invariable keywords* and variable substrings. Figure 2 shows the detailed procedure of step 1. First, substrings composed of two or more characters in URL paths, URL queries, and user agents in HTTP requests of malware traffic are extracted as *candidate keywords*. From the candidate keywords, *invariable keywords* are then detected based on the number of malware samples using the keywords. If there are more malware samples that use the same candidate keyword, it is more likely that the keyword is invariable and reused based on the same malicious infrastructure.

It is claimed that ExecScent [9] observed the existence

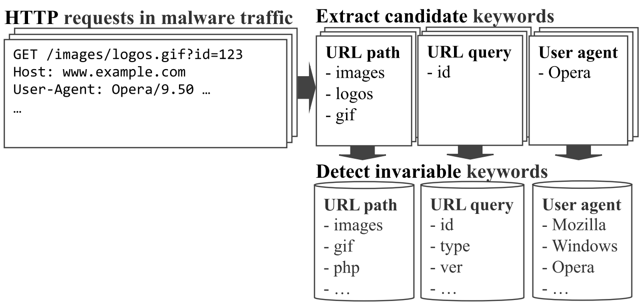


Fig. 2 Example of variability profiling.

of stable URL paths in some malware-generated HTTP requests. However, it does not reveal how to detect and use them. On the other hand, we focused on invariable keywords not only in URL paths but also in URL queries, and user agents. Moreover, step 1 contributes to automatically detecting such invariable keywords to generate more accurate and valuable templates without using any particular ground truth data. The effectiveness of BotProfiler is validated later in Sect. 3.

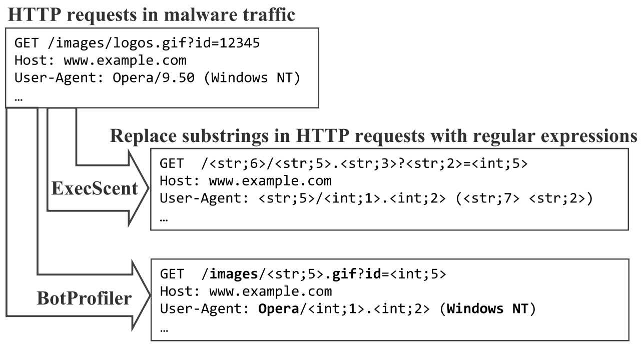
2.3 Step 2: Template Generation

2.3.1 Replacing Substrings in HTTP Requests with Regular Expressions

Step 2 involves generating templates using invariable keywords produced in step 1. These templates contain features of URL paths, URL queries, and user agents, based on the result of clustering HTTP requests in malware traffic. Our templates include not only URLs but also user agents in HTTP request headers to reduce false positives with considering only URLs. To detect polymorphic patterns used by attackers, HTTP requests are segmented into substrings with symbols (e.g., '/', '?', '=', '-', '.') and replaced with regular expressions (e.g., <str; *length*>) containing the data type (e.g., string (str), integer (int), hexadecimal (hex), base64) and length indicated in Table 1. Figure 3 shows examples of how substrings are replaced with regular expressions using both ExecScent and BotProfiler. ExecScent basically replaces all substrings with regular expressions to effectively aggregate HTTP requests into templates. However, this system has a potential problem of falsely matching benign HTTP requests with templates, which results in false positives, due to an inherent aspect of regular expressions. For example, even if the structure of the URL path in a benign HTTP request (e.g., /foobar/index.htm) and that in malware traffic (e.g., /images/logos.gif) are totally different, the regular expressions that correspond to these structures are the same (e.g., /<str;6>/<str;5>.<str;3>). BotProfiler avoids such false positives based on regular expressions. Thus, it does *not* replace the substrings that match any invariable keywords produced in step 1 since these substrings tend to be reused by the same malicious infrastructure. The efficiency for aggregating HTTP requests into templates in ExecScent is higher

Table 1 Example of patterns in regular expressions.

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


HTTP requests in malware traffic
Replace substrings in HTTP requests with regular expressions
ExecScent
BotProfiler

```
GET /images/logos.gif?id=12345
Host: www.example.com
User-Agent: Opera/9.50 (Windows NT)
…
```

```
GET  /<str;6>/<str;5>.<str;3>?<str;2>=<int;5>
Host: www.example.com
User-Agent: <str;5>/<int;1>.<int;2> (<str;7> <str;2>)
…
```

```
GET  /images/<str;5>.gif?id=<int;5>
Host: www.example.com
User-Agent: Opera/<int;1>.<int;2> (Windows NT)
…
```

Fig. 3 Example of replacing substrings in HTTP requests with regular expressions.

than that of BotProfiler because more HTTP requests are converted into the same regular expression patterns. However, BotProfiler generates more specific templates since it uses the reused nature of malicious infrastructures.

2.3.2 Aggregating HTTP Requests

To reduce the number of templates and matching cost using the templates, ExecScent aggregates similar HTTP requests into one template. BotProfiler also uses this concept to generate practical templates. Specifically, it introduces two stages of clustering of HTTP requests with regular expressions. An overview of this is given in Fig. 4.

The first stage consists of clustering using the criterion of destination IP addresses. This clustering process groups HTTP requests that share the same destination IP address range or prefix to generate *IP range clusters*. The second stage consists of applying agglomerative hierarchical clustering to HTTP requests within each IP range cluster. The agglomerative hierarchical clustering sequentially combines similar requests based on the predefined similarity metric to output a dendrogram, which is a tree-like diagram representing the distance between clusters [14]. Cutting the dendrogram at a certain height, which is called the *cut height* and is determined empirically, divides the IP range cluster into *similar request clusters* that include multiple similar HTTP requests. BotProfiler finally selects 0.5 as the cut height based on the best result of preliminary experiments. In each similar request cluster, BotProfiler extracts the centroid, which refers to one of the HTTP requests that maximizes the sum of similarities between the request and all other requests. From the centroids, the *templates* that contain URL paths, URL queries, and user agents are extracted.

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

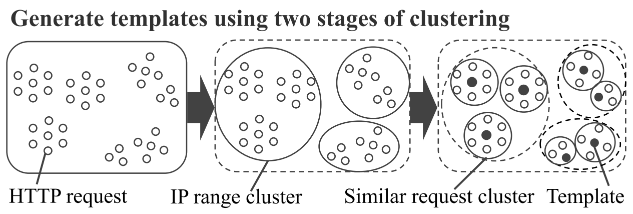


Fig. 4 Overview of process to generate templates using clustering.

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

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

Note that the host header of HTTP requests, such as a domain name or an IP address, is not included as an element in the templates. Under the assumption that attackers reuse their malicious infrastructure, such as web servers and toolkits, to conduct other attacks at a different domain name or IP address, a host header might be changed. For example, using a domain generation algorithm (DGA) enables attackers to change domain names frequently. However, elements in our templates (URL paths, URL queries, and user agents) are more stable than the host header. Thus, our templates can be used to detect attacks caused by the reuse of malicious infrastructure by attackers.

2.4 Step 3: Rarity Profiling

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

$$\rho_{t,k} = 1 - \frac{n_{t,k}}{\max_i n_i} \tag{2}$$

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

2.5 Step 4: Template Matching

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

$$Score(h, t) = \frac{\sum_{k=1}^{n} \sigma_k(h, t) \cdot \omega(\sigma_k(h, t), \rho_{t,k})}{\sum_{k=1}^{n} \omega(\sigma_k(h, t), \rho_{t,k})} \cdot \rho_{h,d} \tag{3}$$

Here, $\sigma_k(h, t)$ is defined in the same way as explained in Sect. 2.3.2, $\rho_{t,k}$ is the rarity of k in t, $\rho_{h,d}$ is the rarity of a destination fully qualified domain name (FQDN) d in h and is calculated in the same way as step 3, and ω is the weight function between $\sigma_k(h, t)$ and $\rho_{t,k}$ and is defined by the following equation.

$$\omega(\sigma_k(h, t), \rho_{t,k}) = 1 + \frac{1}{(2 - \sigma_k(h, t) \cdot \rho_{t,k})^m} \tag{4}$$

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

3. Evaluation

3.1 Evaluation Overview

BotProfiler was evaluated using extensive actual datasets. This section explains how we evaluated it in terms of feasibility and detection performance in comparison to the conventional system ExecScent. The feasibility is based on the effectiveness of invariable keywords and the results of our variability profiling and template generation. Detection performance was measured by the detection rate of infected hosts and the false positive rate in the deployed network. Note that ExecScent is closed-source software; thus, we reimplement it based on the paper [9] to compare the detection performance in both systems.

3.2 Datasets

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

Malware traffic was captured from the sandbox system [11] running malware samples. The sandbox supports executable files only in Microsoft Windows environments. These malware samples were collected using the honeyclient [12], [13] crawling public blacklists such as MDL [15] and hpHosts [16] and some commercial blacklists from August 2011 to August 2013. Table 2 lists the number of malware samples in each dataset, number and ratio of samples detected by antivirus software, number of unique malware family names, and number of HTTP requests. The *Current* dataset was used for generating templates and composed of HTTP requests generated by 2,507 unique malware samples. Datasets labeled *Future* were used to evaluate detection performance and consisted of datasets divided into months *Future_1–8*. Note that there were no overlaps in malware samples between the Current and Future_1–8 datasets.

All malware samples were checked with multiple antivirus software programs using the VirusTotal [17] as of January 6, 2015. Kaspersky was selected to label samples with malware family names for the following two reasons. One reason is that it achieved the best detection rate excepting the uninformative labels (e.g., generic, heuristic). The other reason is that the rule for family names is openly available [18]. The latest malware definition file of the software as of January 6, 2015 was used for the evaluation. Table 2 indicates that the detection rate for antivirus software was not very good in any of the datasets even if the latest definition file was used. The reason is that most of the malware samples were not suitably collected or analyzed by the antivirus vendors. Table 3 lists the top 10 malware family names detected by the antivirus in both the Current and Future_1–8 datasets. This result indicates that the malware samples in our datasets were unbiased in terms of malware families. Moreover, the result shows the difference in distribution of families between Current and Future_1–8. Note that malware family information was used as reference in our evaluation and was *not* used in BotProfiler.

Benign traffic was captured in a large, real enterprise network from December 2012 to August 2013. The traffic was inspected by security engineers using commercial ground truth data to filter out the possibility of containing malicious traffic in January 2015. Table 4 shows the number of source IP addresses and number of outbound HTTP requests. The Training dataset was used to calculate rarities, and the Testing dataset was used to evaluate the false positive rate.

Table 2 Malware traffic datasets.

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

Table 3 Malware families in malware traffic datasets.

(A) Current

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

(B) Future_1–8

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

Table 4 Benign traffic datasets.

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

3.3 Verifying the Effectiveness of Invariable Keywords

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

Table 5 Classification of URL path structure.

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

The following results are the basis of the superiority of BotProfiler.

Table 5 lists the results of analyzing the structure of the *URL paths* in each malware family. The results indicate that 68% of malware families in the Current dataset send HTTP requests that have a fixed URL path structure, meaning that it contains fixed and fixed-length substrings. Invariable keywords in BotProfiler are effective for such fixed URL path structures. Two reasons can be considered for the large portion of fixed URL path structures. One is the use of publicly available APIs by malware samples. Some attackers use APIs (e.g., GeoIP) to obtain information on infected hosts such as IP addresses and locations. Using such APIs forces attackers to comply with the requirements of the APIs and to send HTTP requests that have a fixed URL path. The other reason is the high cost for attackers to accept random URL path structures. For example, attackers need to receive and interpret HTTP requests with such random URL paths on their C&C servers.

Table 6 lists the result of analyzing the structure of the *URL queries* in each malware family. The results indicate that 32% of malware families in the Current dataset send HTTP GET requests that have fixed URL query field names. Invariable keywords in BotProfiler are effective for such fixed URL query field names. The reasons for the fixed URL query field name are the same as those for having a fixed URL path, namely, the utilization of publicly available APIs and the high cost for attackers to accept a random URL query field name. In Table 6, the item listed as *no queries* in the HTTP GET method includes instances of checking the Internet connection using malware samples, and *no queries* in the HTTP POST method includes instances of sending information using the body of the request.

The results of analyzing the structure of *user agents* in each malware family are listed in Table 7. The results indicate that 77% of malware families in the Current dataset send HTTP requests that have the same user agent as general web browsers such as Internet Explorer and Firefox. User agents of general web browsers consist of fixed substrings, which represent the name of the browser or OS, and the version number (e.g., Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)). Therefore, using such fixed substrings as invariable keywords in BotProfiler is effective in generating accurate templates. The reason that most attackers use general web browsers as user agents is that they try to hide their C&C communications in legitimate HTTP communications. On the other hand, 11% of malware families send HTTP requests with unusual user agents (e.g., _Converter_agent, VBTagEdit). However, most of these user agents include fixed substrings. Thus, using such fixed substrings as invariable keywords is also considered to be effective.

Table 6 Classification of URL query structure.

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

Table 7 Classification of user agent structure.

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

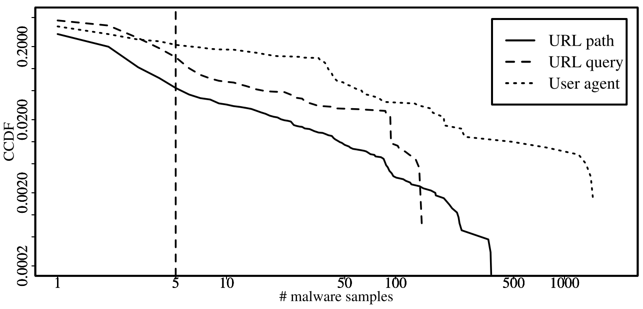


Fig. 5 CCDF of number of malware samples for each keyword.

3.4 Results of Variability Profiling

This section explains the results of conducting variability profiling (step 1) with the Current dataset listed in Table 2. Specifically, we analyzed the relationship between candidate keywords in URL paths, URL queries, and user agents, and the number of malware samples that use each candidate keyword. The more malware samples that use a candidate keyword, the more likely it is that the candidate keyword is the invariable keyword that is based on the same malicious infrastructure. Figure 5 is the complementary cumulative distribution function (CCDF) showing the relationship between candidate keywords and the number of malware samples that use the keywords. In this case, CCDF corresponds to the probability of the candidate keywords being greater than or equal to the specified number of malware samples.

Figure 5 reveals the existence of invariable keywords that are reused across multiple malware samples in the URL paths, URL queries, and user agents of HTTP requests generated by infected hosts. BotProfiler sets a threshold for the number of malware samples, and any candidate keyword that exceeds the threshold is identified as an invariable keyword. In our further evaluation, the threshold was set to five malware samples based on the best result in our preliminary verification. Table 8 lists the number of candidate keywords and that of invariable keywords in each URL path, URL query, and user agent. Moreover, Table 9 gives examples of invariable keywords identified with BotProfiler.

Table 8 Summary of variability profiling.

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

Table 9 Example of detected invariable keywords.

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

3.5 Results of Generating Templates

This section describes the results of generating templates (step 2) with the Current dataset and invariable keywords output from step 1. Table 10 lists the number of templates output from ExecScent, which does *not* use invariable keywords in templates, and that from BotProfiler, which uses invariable keywords. As explained in Sect. 2.3.1, the number of templates in BotProfiler exceeds that in ExecScent because the efficiency of aggregating requests in BotProfiler is lower than that in ExecScent.

Examples of templates generated with BotProfiler are shown in Fig. 6. For example, Template #1 was created by keep-alive C&C communications used by the Sality botnet. Template #2 was produced by C&C communications that involved counting the number of infected hosts by the ZeroAccess botnet, which mainly uses P2P as a C&C protocol. However, BotProfiler succeeded in generating templates from a limited percentage of HTTP requests in ZeroAccess. Template #3 was generated by communications in which infected hosts are forced to download and install fake antivirus software. Note that these templates are automatically generated with BotProfiler *only* from HTTP requests in the Current dataset without using any other information such as malware family names obtained by antivirus software. With BotProfiler, generated templates are delivered to the deployment network and matched with traffic using pre-calculated rarities in the deployment network to reduce false positives.

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

Table 10 Summary of template generation.

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

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

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

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

Fig. 6 Examples of generated templates.

Table 11 Malware families in generated templates.

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

Table 11 lists the top 10 malware family names detected by the antivirus software in the generated templates. A comparison between Table 3 (A) and Table 11 reveals that the composition of malware families in the Current dataset and that in templates generated from the Current dataset differ widely. This is due to the characteristics of C&C communications. For example, the number of inputted ZeroAccess (Backdoor.Win32.ZAccess) malware samples was large; however, it mainly uses P2P as the C&C protocol and sends a few variations of HTTP requests. Thus, the number of generated templates is small.

3.6 Detection Rate

This section compares the detection rate in both ExecScent and BotProfiler. The detection rate is defined by the ratio of correctly detected infected hosts. For simplicity, an infected host is only infected with one malware sample at a

Table 12 Detection rate on malware traffic datasets.

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

time. That is, a malware sample in a dataset was considered to be detected if at least one of the HTTP requests in each malware sample was detected with BotProfiler. In our evaluation, we determined the detection rate of BotProfiler, which generates templates with invariable keywords output from step 1, and that of ExecScent, which does *not* use invariable keywords.

Table 12 lists the results of detecting malware samples in *Future* datasets using templates generated by the *Current* dataset. This table shows detection rates with a variable matching score threshold. If $Score(h, t)$ exceeds the matching score threshold, BotProfiler determines the HTTP request h as being generated by an infected host, as described in Sect. 2.5. The Future datasets consisted of datasets divided by month *Future_1–8*. Each Future dataset contains malware samples collected after the Current dataset was compiled. Our evaluation using the Future datasets enabled us to track changes in the detection rate of templates from the Current dataset over time.

The results reveal three facts concerning detection rates in both systems. First, the detection rates of both systems decreased linearly over time (from Future_1 to Future_8) in each matching score threshold. This is due to matching Future datasets with the same templates generated from the Current dataset, even if the malware samples in the Future datasets vary over time. This fact led us to conclude that templates should be updated using the latest malware samples on a regular basis. In this case, the templates should be updated after two months (Future_2) to obtain a detection rate of at least 80% at a matching score threshold of 0.85.

Second, the detection rates of BotProfiler tended to be lower than those of ExecScent as time advanced. For example, at a matching score threshold of 0.75, the detection rates in Future_1 and Future_2 were more than those of ExecScent. On the other hand, the detection rates between Future_3 and Future_8 were lower than those of ExecScent. These results indicate that the effective period of templates in ExecScent is longer than that of BotProfiler since the matching area covered by the regular expressions of ExecScent is wider than that of BotProfiler. However, the actual

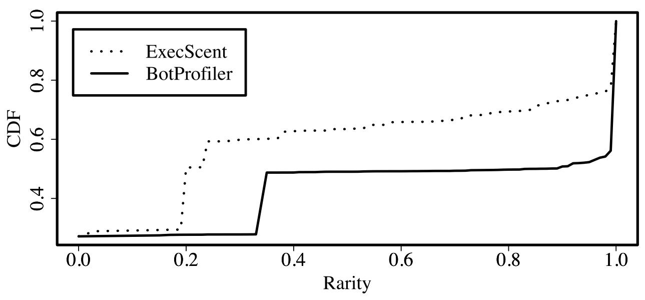


Fig. 7 CDF of URL path rarities in malware traffic datasets.

operation of BotProfiler in the deployment network includes downloading templates from the lab environment at least once every month. In such a case, the detection rate of BotProfiler is superior to that of ExecScent.

Finally, BotProfiler improved the detection rates with all thresholds between 0.75 and 0.95 in Future_1 and Future_2, even when it introduced invariable keywords to generate more specific templates. This is not a surprising result and is reasonable. The only reason is that BotProfiler determines not only the similarity to the templates but also the rarity of each element in the templates. For example, the rarity of the URL path in BotProfiler (e.g., /images/<str;5>.gif) is higher than that of ExecScent (e.g., /<str;6>/<str;5>.<str;3>) because the latter matches various types of HTTP requests in the deployment network. Figure 7 shows the CDF of URL path rarities in the deployment network when calculating a matching score of Future datasets. This graph illustrates that URL path rarities in BotProfiler are higher than those of ExecScent. This difference contributes to raising the matching scores of HTTP requests to improve the detection rates in BotProfiler.

3.7 False Positive Rate

This section compares the false positive rates for both ExecScent and BotProfiler. The false positive rate is defined by the ratio of falsely detected benign HTTP requests. Table 13 presents the false positive rates with variable matching score thresholds. The results indicate that the false positive rates of BotProfiler are always lower than those of ExecScent for all matching score thresholds between 0.75 and 0.95. In particular, at the threshold of 0.75, the false positive rate of BotProfiler (1.18%) was less than one-third that of ExecScent (3.89%). This is due to the effect of using invariable keywords in BotProfiler. Our invariable keywords enable BotProfiler to generate more accurate templates, which causes fewer false positives. We conclude that BotProfiler achieves a higher detection rate and lower false positive rate simultaneously under the condition in which templates are regularly updated. In operating BotProfiler in the deployment network, a matching score threshold is set based on an acceptable false positive rate in the network. For example, setting the threshold at 0.85 enables BotProfiler to reduce the false positive rate to 0.06%.

Table 13 False positive rate on benign traffic datasets.

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

4. Limitation

4.1 Detecting Invariable Keywords

Our variability profiling (step 1) heuristically set the threshold of the number of malware samples to five, based on the result of the preliminary verification. That threshold is used to identify invariable keywords; that is, any candidate keywords that exceed the threshold are identified as invariable keywords. Specifically, increasing the threshold reduces the number of invariable keywords and vise versa, as shown in Fig. 5. Reducing the number of invariable keywords causes an expansion of the matching area of regular expressions in our templates. This leads to an increase in both the detection rate and the false positive rate. Our future task is to automatically set the optimal threshold to meet the requirement for detection rate or false positive rate in each deployment network.

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

Dynamic analysis with code tainting can be used as another implementation of the concept of our variability profiling. Code tainting is an approach to track data propagation on a running system and is combined with dynamic analysis or a sandbox system [19]. Under the assumption that the data propagation reveals the existence of fixed substrings in HTTP requests generated by malware samples, such fixed substrings can be utilized as invariable keywords. However, code tainting is a resource-hungry technique to run. Therefore, our lightweight approach, which is based on counting the number of malware samples, has a competitive advantage over code tainting.

4.2 Generating Templates from Dynamic Malware Analysis Results

As shown in Fig. 1, BotProfiler generates templates from malware traffic. That is, to generate *ideal* templates, our lab environment needs to collect malware samples exhaustively and to analyze the collected samples adequately. These tasks may be problematic regarding cyber security for two reasons. One reason is that it is virtually impossible to collect all malware samples because attackers mass-produce their malware samples using a toolkit, and they have recently been targeting particular environments or organizations using drive-by downloads or advanced persistent threats (APTs). The other reason is that some malware samples can evade detection by identifying the sandbox environment. If malware samples include such a function, they cannot be dynamically analyzed using the sandbox. Kirat et al. recently proposed a malware analysis system using a *real* environment that does not include a monitoring component inside the system [20]. Such an analysis method may be a solution to analyze sophisticated malware samples.

4.3 Update of Rarities on Deployment Network

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

5. Related Work

5.1 Generating Network-Based Signatures or Templates

Much research has been done on generating network-based signatures or templates as a network-based countermeasure against infected hosts. Xie et al. proposed a system called AutoRE for generating signatures with regular expressions to detect polymorphic URLs in spam emails sent from botnets based on the nature of similar substrings in malicious URLs [7]. BotProfiler differs from AutoRE in that the focus is not spam emails but infected hosts, and the suffix-array algorithm proposed for AutoRE cannot appropriately be applied to substrings in HTTP requests. Perdisci et al. proposed a method of generating signatures with regular expressions to detect the URLs used in C&C communications based on HTTP traffic captured in a controlled environment [8]. BotProfiler targets not only URLs but also HTTP requests and determines not only similarities but also rarities to reduce false positives. Nelms et al. improved Perdisci et al.'s method [8] in their system called ExecScent, which covers HTTP requests [9]. ExecScent is one of the most advanced systems for generating templates for infected hosts using regular expressions and was a significant influence in developing BotProfiler. We expanded ExecScent to introduce the concept of invariability in substrings in HTTP requests. Zarras et al. proposed the BotHound system to focus on the sequence of components in HTTP headers to generate templates [21]. BotProfiler does not use the sequence of HTTP headers, that is, it is a more lightweight system than BotHound. Zand et al. proposed a method of generating signatures to detect C&C communications by focusing on frequent words in C&C communications [22]. The idea is similar to our invariable keywords, although this method cannot generate accurate templates composed of URL paths, URL queries, and user agents and cannot determine both invariable and variable features in templates.

5.2 Modeling and Detecting Botnets

Another countermeasure against infected hosts includes modeling and detecting multiple infected hosts by using characteristics of botnets. Gu et al. proposed BotSniffer [23] and BotMiner [24] to detect simultaneous and similar network behaviors between multiple hosts in order to identify the activities of infected hosts based on the key idea that infected hosts of the same botnet have similar characteristics. Unlike with BotProfiler, controlling the false positive rate is generally difficult with these anomaly-based systems, and they are also difficult to deploy in a large and real network. Rossow et al. proposed a method to observe and model P2P botnets such as Zeus, ZeroAccess, and Kelihos [25]. Also, Zhang et al. proposed a method to detect such P2P botnet activities in a network [26]. BotProfiler does not focus on P2P botnets because our deployment networks basically only accept web protocols and do not accept P2P protocol. Caballero et al. proposed a method called protocol reverse engineering to analyze messages in unknown or undocumented protocols used in C&C communications [27]. BotProfiler only focuses on HTTP request headers, as HTTP is a known and documented protocol. Thus, our system does not need to use such techniques.

5.3 Detecting C&C Domain Names

The other approaches focus on the characteristics of domain names used in C&C communications to detect accesses to such domain names as infected hosts' activities. Holz et al. and Passerini et al. proposed methods to utilize features of Fast-Flux, which is a technique used by attackers to frequently change the mappings of their C&C domain names and a lot of IP addresses, and to identify C&C domain names [28], [29]. Also, Schiavoni et al. proposed a method focusing on the characteristics of DGAs, which are frequently used in C&C to generate multiple domain names, and to detect such DGA domain names [30]. These methods focus exclusively on the relationship of domain names and differ from BotProfiler in the way that our templates only focus on URL paths, URL queries, and user agents. Therefore, these methods could be combined with our templates to detect infected hosts more accurately.

6. Conclusion

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

References

[1] D. Chiba, T. Yagi, M. Akiyama, K. Aoki, T. Hariu, and S. Goto, "BotProfiler: Profiling variability of substrings in HTTP requests to detect malware-infected hosts," Proc. 2015 IEEE 14th International Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom), Helsinki, Finland, pp.758–765, Aug. 2015. DOI: 10.1109/Trustcom.2015.444.

[2] R.A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and taxonomy of botnet research through life-cycle," ACM Comput. Surv., vol.45, no.4, pp.1–33, Aug. 2013. DOI:10.1145/2501654.2501659.

[3] McAfee Labs, "Periodic connections to control server offer new way to detect botnets," Oct. 2013. http://blogs.mcafee.com/mcafee-labs/periodic-links-to-control-server-offer-new-way-to-detect-botnets, accessed July 27. 2015.

[4] Microsoft, "Microsoft and financial services industry leaders target cybercriminal operations from zeus botnets," March 2012. http://blogs.microsoft.com/blog/2012/03/25/microsoft-and-financial-services-industry-leaders-target-cybercriminal-operations-from-zeus-botnets/, accessed July 27. 2015.

[5] Microsoft, "Microsoft works with financial services industry leaders, law enforcement and others to disrupt massive financial cybercrime ring," June 2013. http://blogs.microsoft.com/blog/2013/06/05/microsoft-works-with-financial-services-industry-leaders-law-enforcement-and-others-to-disrupt-massive-financial-cybercrime-ring/, accessed July 27. 2015.

[6] Microsoft, "Microsoft helps FBI in GameOver Zeus botnet cleanup," June 2014. http://blogs.microsoft.com/blog/2014/06/02/microsoft-helps-fbi-in-gameover-zeus-botnet-cleanup/, accessed July 27. 2015.

[7] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: signatures and characteristics," Proc. ACM SIGCOMM 2008 Conf. on Data Communication, pp.171–182, Aug. 2008. DOI:10.1145/1402958.1402979.

[8] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," Proc. 7th USENIX Symp. on Networked Systems Design and Implementation (NSDI), pp.391–404, San Jose, CA, USA, April 2010.

[9] T. Nelms, R. Perdisci, and M. Ahamad, "ExecScent: Mining for new c&c domains in live networks with adaptive control protocol templates," Proc. 22th USENIX Security Symp., pp.589–604, Washington, DC, USA, Aug. 2013.

[10] Ponemon Institute, "The cost of malware containment," Jan. 2015. http://www.ponemon.org/library/the-cost-of-malware-containment, accessed July 27. 2015.

[11] K. Aoki, T. Yagi, M. Iwamura, and M. Itoh, "Controlling malware HTTP communications in dynamic analysis system using search engine," Proc. 3rd IEEE International Workshop on Cyberspace Safety and Security (CSS), pp.1–6, Milan, Italy, Sept. 2011. DOI:10.1109/CSS.2011.6058563.

[12] M. Akiyama, M. Iwamura, Y. Kawakoya, K. Aoki, and M. Itoh, "Design and implementation of high interaction client honeypot for drive-

by-download attacks," IEICE Trans. Commun., vol.E93-B, no.5, pp.1131–1139, May. 2010. DOI:10.1587/transcom.E93.B.1131.

[13] M. Akiyama, T. Yagi, Y. Kadobayashi, T. Hariu, and S. Yamaguchi, "Client honeypot multiplication with high performance and precise detection," IEICE Trans. Inf.& Syst., vol.E98-D, no.4, pp.775–787, April 2015. DOI:10.1587/transinf.2014ICP0002.

[14] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data clustering: A review," ACM Comput. Surv., vol.31, no.3, pp.264–323, Sept. 1999. DOI:10.1145/331499.331504.

[15] "Malware domain list." http://www.malwaredomainlist.com/, accessed July 27. 2015.

[16] Malwarebytes, "hpHosts." http://www.hosts-file.net/, accessed July 27. 2015.

[17] "VirusTotal." https://www.virustotal.com/, accessed July 27. 2015.

[18] Kaspersky Lab, "Rules for naming." http://securelist.com/threats/rules-for-naming/, accessed July 27. 2015.

[19] G. Jacob, R. Hund, C. Kruegel, and T. Holz, "JACKSTRAWS: Picking command and control connections from bot traffic," Proc. 20th USENIX Security Symp., San Francisco, CA, USA, Aug. 2011.

[20] D. Kirat, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal analysis-based evasive malware detection," Proc. 23rd USENIX Security Symp., pp.287–301, San Diego, CA, USA, Aug. 2014.

[21] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of HTTP-based malware," Proc. 12th Annual International Conference on Privacy, Security and Trust (PST), Toronto, ON, Canada, pp.249–256, July 2014. DOI:10.1109/PST.2014.6890946.

[22] A. Zand, G. Vigna, X. Yan, and C. Kruegel, "Extracting probable command and control signatures for detecting botnets," Proc. 29th Annual ACM Symp. on Applied Computing (SAC), Gyeongju, Republic of Korea, pp.1657–1662, April 2014. DOI:10.1145/2554850.2554896.

[23] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, Feb. 2008.

[24] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," Proc. 17th USENIX Security Symp., pp.139–154, San Jose, CA, USA, July 2008.

[25] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. Dietrich, and H. Bos, "SoK: P2PWNED — Modeling and evaluating the resilience of peer-to-peer botnets," Proc. IEEE Symp. on Security and Privacy (S&P), pp.97–111, Berkeley, CA, USA, May 2013. DOI:10.1109/SP.2013.17.

[26] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, "Building a scalable system for stealthy P2P-botnet detection," IEEE Trans. Information Forensics and Security, vol.9, no.1, pp.27–38, Jan. 2014. DOI:10.1109/TIFS.2013.2290197.

[27] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering," Proc. 16th ACM Conf. Computer and Communications Security (CCS), pp.621–634, Chicago, IL, USA, Nov. 2009. DOI:10.1145/1653662.1653737.

[28] T. Holz, C. Gorecki, K. Rieck, and F.C. Freiling, "Measuring and detecting fast-flux service networks," Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, Feb. 2008.

[29] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi, "FluXOR: Detecting and monitoring fast-flux service networks," in Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Lecture Notes in Computer Science, vol.5137, pp.186–206, Springer Berlin Heidelberg, July 2008. DOI:10.1007/978-3-540-70542-0_10.

[30] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: DGA-based botnet tracking and intelligence," in Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Lecture Notes in Computer Science, vol.8550, pp.192–211, Springer International Publishing, July 2014. DOI:10.1007/978-3-319-08509-8_11.

Daiki Chiba is currently a researcher at NTT Secure Platform Laboratories, Tokyo, Japan. He is also a Ph.D. student at Waseda University, Tokyo, Japan. He received his B.E. and M.E. degrees in computer science and engineering from Waseda University in 2011 and 2013, respectively. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2013, he has been engaged in research on cyber security through data analysis.

Takeshi Yagi received his B.E. degree in electrical and electronic engineering and his M.E. degree in science and technology from Chiba University, Japan in 2000 and 2002. He also received his Ph.D. degree in information science and technology from Osaka University, Osaka, Japan in 2013. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2002, he has been engaged in research and design of network architecture, traffic engineering, and his current research interests include network security, web security, honeypots, security-data analysis based on machine learning, and security intelligence technologies such as URL/domain/IP blacklisting and reputation. He is now a senior research engineer in the Cyber Security Project of NTT Secure Platform Laboratories. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Institute of Electrical Engineers of Japan (IEEJ).

Mitsuaki Akiyama received his M.E. and Ph.D. degrees in Information Science from Nara Institute of Science and Technology, Japan in 2007 and 2013. Since joining Nippon Telegraph and Telephone Corporation NTT in 2007, he has been engaged in research and development of network security, especially honeypot and malware analysis. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.

Kazufumi Aoki received the M.S degree in Graduate School of Information Science from Tohoku University, Japan in 2006. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2006, he has beed engaged in research and development of network security. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.

Takeo Hariu is a senior research engineer, and supervisor at Nippon Telegraph and Telephone Corporation. He received his M.S. in electro-communications from The University of Electro-Communications. Contact him at hariu.takeo@lab.ntt.co.jp.

Shigeki Goto is a professor at the Department of Computer Science and Engineering, Waseda University, Japan. He received his B.S. and M.S. in Mathematics from the University of Tokyo. Prior to becoming a professor at Waseda University, he has worked for NTT for many years. He also earned his Ph.D. in Information Engineering from the University of Tokyo. He is the president of JPNIC. He is a member of ACM and IEEE, and he was a trustee of Internet Society from 1994 to 1997.