IEICE TRANS. INF. & SYST., VOL.E101–D, NO.5 MAY 2018

PAPER

Detecting Malware-Infected Devices Using the HTTP Header Patterns*

Sho MIZUNO[†a)], ***Nonmember***, **Mitsuhiro HATADA**[†,††b)], **Tatsuya MORI**[†c)], ***and*** **Shigeki GOTO**[†d)], ***Members***

SUMMARY Damage caused by malware has become a serious problem. The recent rise in the spread of *evasive malware* has made it difficult to detect it at the pre-infection timing. Malware detection at post-infection timing is a promising approach that fulfills this gap. Given this background, this work aims to identify likely malware-infected devices from the measurement of Internet traffic. The advantage of the traffic-measurement-based approach is that it enables us to monitor a large number of endhosts. If we find an endhost as a source of malicious traffic, the endhost is likely a malware-infected device. Since the majority of malware today makes use of the web as a means to communicate with the C&C servers that reside on the external network, we leverage information recorded in the HTTP headers to discriminate between malicious and benign traffic. To make our approach scalable and robust, we develop the *automatic template generation* scheme that drastically reduces the amount of information to be kept while achieving the high accuracy of classification; since it does not make use of any domain knowledge, the approach should be robust against changes of malware. We apply several classifiers, which include machine learning algorithms, to the extracted templates and classify traffic into two categories: malicious and benign. Our extensive experiments demonstrate that our approach discriminates between malicious and benign traffic with up to 97.1% precision while maintaining the false positive rate below 1.0%.

key words: *botnet detection, malicious traffic, HTTP header, automatic template generation*

1. Introduction

As the targets of malware have been diversified, any operating systems and devices can become infected by new malware. For instance, while the common malware has been aimed at Windows OSes, Ref. [2] reported that the number of malware samples for Mac OS X has increased to approximately about 460,000 in 2016. ESET predicts that Internet-of-Things (IoT) devices will become increasingly attacked in 2016 [3]. Nearly 25,000 security cameras installed in bots throughout 105 countries have been subjected to springboard attack [4].

These observations indicate that today, any devices could become infected with new malware, which cannot be detected with the existing solutions; i.e., the malware detection at the pre-infection timing has become infeasible to some extent. Also, the recent rise in the spread of *evasive malware* has also made it difficult to detect malware at the pre-infection timing [5]. Therefore, malware detection at post-infection timing is a promising approach that can fulfill this gap.

The goal of this study is detecting malware-infected devices as early as possible. Since the malware-infected devices start to communicate with external Command and Control (C&C) servers, we make a hypothesis that Internet traffic monitoring is a useful approach to finding malware-infected devices. The advantage of the traffic-measurement-based approach is that it enables us to monitor a large number of endhosts. If we find an endhost as a source of malicious/suspicious traffic, the endhost is likely a malware-infected device.

To validate our hypothesis, we build the system called *BotDetector*, which takes the following approaches. First, we make use of the HTTP packets originated from devices because the majority of malware today uses HTTP protocol as a means to communicate with the C&C servers that reside in the external network [6]. Second, we leverage machine learning techniques to classify traffic into two categories: malicious and benign. One key technical challenge is to make our approach scalable. Since the values that can be recorded in the HTTP headers have a high degree of freedom, they could include roughly 10K of distinct feature vectors which we confirmed. To reduce the amount of information to be kept while achieving the high accuracy of classification, we develop the *automatic template generation* scheme that aggregates similar features with the statistical technique. We note that this automation enables us to make our system *robust* to change in the malware traffic because it does not rely on the domain knowledge in selecting useful features. Through the extensive experiments using a large-scale traffic dataset, we demonstrate the *BotDetector* can detect malware-infected devices accurately in a scalable manner.

The remainder of this paper is organized as follows. Section 2 describes the overview of the *BotDetector* system. Section 3 presents the dataset we used to evaluate the performance of the *BotDetector*. Section 4 shows the results. In Sect. 5, we discuss limitations and practical aspects of our approach. Section 6 summarizes the related works,

Manuscript received September 16, 2017.
Manuscript revised January 15, 2018.
Manuscript publicized February 8, 2018.

[†]The authors are with the Dept. of Communication Engineering, Waseda University, Tokyo, 169–8555 Japan.
[††]The author is with the NTT Communications Corporation, Tokyo, 108–8118 Japan.

*An earlier version of this paper was presented at IEEE International Conference on Communications (ICC 2017), May 2017 [1]. The authors have confirmed that the copyright issue can be resolved before the camera-ready submission.

a) E-mail: mizuno@nsl.cs.waseda.ac.jp
b) E-mail: m.hatada@nsl.cs.waseda.ac.jp
c) E-mail: mori@nsl.cs.waseda.ac.jp
d) E-mail: goto@goto.info.waseda.ac.jp

DOI: 10.1587/transinf.2017EDP7294

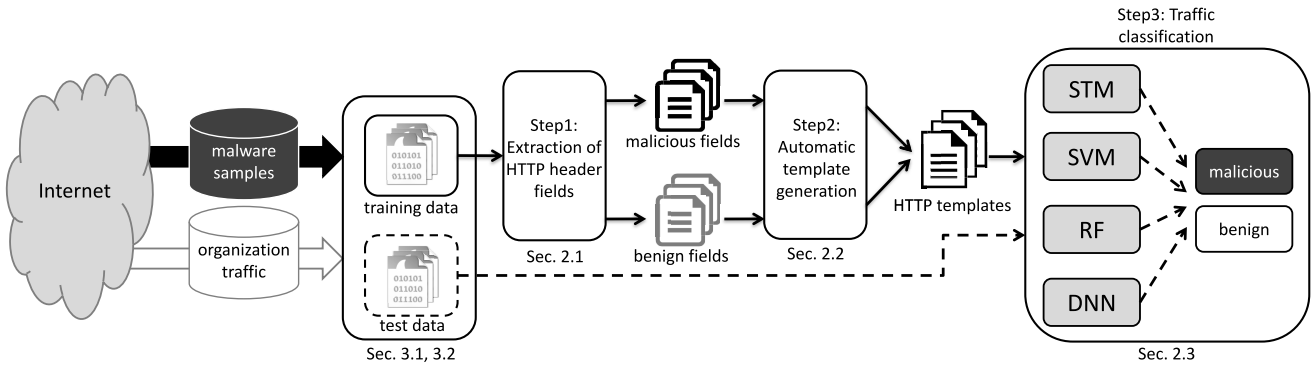

Fig. 1 Overview of BotDetector.

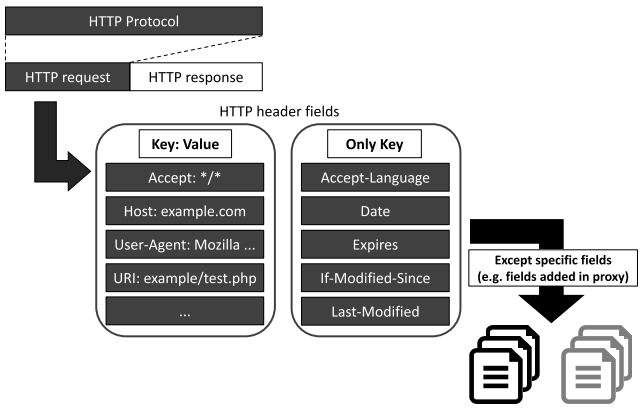

Fig. 2 The process of extracting HTTP header fields.

and Sect. 7 concludes our work.

2. BotDetector Framework

In this section, we present the high-level overview of our system – *BotDetector*. Figure 1 shows the summary of the *BotDetector* system. The goal of the system is to find malware-infected devices by discriminating benign from malicious traffic. The system composes of the three steps: **Step1**: extraction of HTTP header fields (Sect. 2.1), **Step2**: automatic template generation (Sect. 2.2), and **Step3**: traffic classification (Sect. 2.3). In the following, we describe the details of each step.

2.1 Step1: Extraction of HTTP Header Fields

As we mentioned earlier, the mainstream malware uses the HTTP protocol when it is installed or communicates with the C&C server [6]. The *BotDetector* exploits the information listed in the HTTP headers. Figure 2 illustrates the process of extracting HTTP header fields. We first extract the keys and corresponding values from the HTTP header fields. While the URI information is not strictly the HTTP header fields, we also extract the URI information, which composes of path and query information. For the URI information, we use "URI" as their key.

To make our analysis fair, we intentionally eliminate several values that could depend on the data collection environment, e.g., date or language of the devices. After several trials, we found that these values are not useful. So, we decided to remove these values from our analysis. Namely, we remove the following keys/values.

- Accept-Language
- Date
- Expires
- If-Modified-Since
- Last-Modified

We also disposed of HTTP header fields appearing less than ten times in all training data, and fields which are added when the data packets pass through middleboxes such as proxy servers. We empirically determine the threshold. If we did not use the threshold, the cost of analysis increases as we need to use all the observed fields. On the other hand, if we increase the threshold, the accuracy will be sacrificed as the information we can use for learning also decreases, We tried to make a balance between the cost and accuracy and found our threshold hit the good trade-off. Rarefields are non-versatile, and fields passing through middleboxes depend on the particular environment, so their inclusion is not useful.

2.2 Step2: Automatic Template Generation

The number of distinct HTTP header fields extracted from our dataset could be roughly 10K. Moreover, the presence of many unnecessary features risks overfitting in the machine learning. Therefore, we focus on the variability of words constituting the HTTP header fields and aim to compress their information. We make use of the template generation technique [7] based on the DBSCAN algorithm [8].

2.2.1 Scoring

We first present the scoring method. Using all the HTTP header fields in training data, for each field, we assign scores to the words within the field. Note that each field is divided into the words using the following separator: space, "/", "=" and ",". We then calculate the conditional probability of each word as the score. For a word, w, in a given field,

Table 1 Example of creating HTTP templates from HTTP header fields.

original HTTP header fields	automatically generated templates
Accept: text json	Accept: text *
Accept-Encoding: gzip deflate	Accept-Encoding: gzip *
Connection: Keep-Alive	Connection: *
User-Agent: Mozilla 4.0 (compatible; MSIE 6.0; Windows NT 5.1)	User-Agent: Mozilla * (compatible; MSIE * Windows NT *

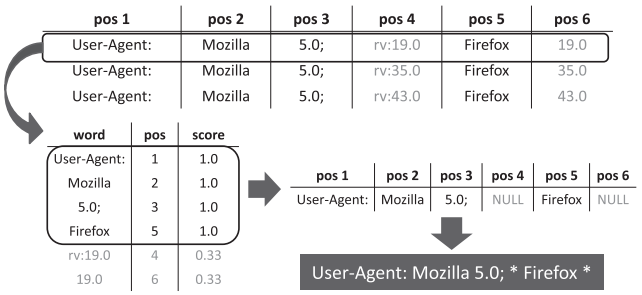

Fig. 3 The process of creating templates. There are three User-Agent fields, $\delta = 10^{-1}$, and $\beta = 0.5$ in this case. The templates are created from each field, and duplications are removed.

F, the score of the word, $S(w; F)$, is given by the following conditional probability:

$$S(w; F) = P(w \mid \mathrm{pos}(w, F), \mathrm{len}(F)) = \frac{n(w, \mathrm{pos}(w, F), \mathrm{len}(F))}{n(\mathrm{pos}(w, F), \mathrm{len}(F))}$$

where $\mathrm{pos}(w, F)$ is the position of the word in the given field, F, and $\mathrm{len}(F)$ is the number of words in the field, F, respectively. If $F = \{\mathrm{foo}, \mathrm{bar}, \mathrm{baz}, \mathrm{qux}\}$ and $w = \mathrm{bar}$, $\mathrm{pos}(w, F) = 2$ and $\mathrm{len}(F) = 4$. $n(X)$ is the number of occurrences of the variate X over the entire fields.

2.2.2 DBSCAN

DBSCAN [8] is one of the clustering algorithms that do not require a predefined number of clusters and its algorithm extracts clusters with any shape. The thresholds for the minimum distance and the minimum number of elements in a cluster are denoted ϵ and m, respectively. Given a certain data, D, and two elements p and q, we define a set $N_\epsilon(p)$ as follows

$$N_\epsilon(p) = \{q \in D \mid d(p, q) \leq \epsilon\},$$

where $d(x, y)$ is the Euclidian distance between x and y. $N_\epsilon(p)$ is a set of points that are within the distance of ϵ from a given point p. If p and q satisfy the following conditions, they are grouped into the same cluster.

$$p \in N_\epsilon(q),$$
$$|N_\epsilon(q)| \geq m.$$

2.2.3 Clustering

Finally, we describe the algorithm that combines the scoring method and DBSCAN. Table 1 and Fig. 3 present the overview of the automatic template generation. We introduce two thresholds δ ($\delta \geq 0$) and β ($0 < \beta < 1$). The thresholds will be empirically determined later. Note that δ is the threshold that determines the minimum distance between clusters. The clustering process composes of the following steps.

Sort each word:
We sort each word in descending order of its score.

Cluster generation:
Using the DBSCAN algorithm, we cluster the words. When the score of the next word differs from the mean score of a cluster by less than δ, we add the next word to the current cluster. Otherwise, we assign the next word to a new current cluster. We repeat the process until all words are assigned to either cluster.

Output template:
Using the cluster results, we obtain a template by choosing the top clusters so that the number of words is greater than $\beta \times \mathrm{len}(F)$. For the rest of clusters that did not exceed the thresholds, we replace the words with the wildcard character '*' as shown in Fig. 3.

2.3 Step3: Traffic Classification

In this step, we detect malicious traffic using the best classifier. We test several classifiers and evaluate their performance. As features, we use the HTTP templates created in the last step. In this work, we test the following classifiers: Simple Template Matching (STM), Support Vector Machine (SVM), Random Forest (RF), and Deep Neural Network (DNN). These algorithms are selected because it is known that these models give a good performance. Of the classifiers, STM is not a machine learning algorithm. However, we added it as a baseline for the performance evaluation. STM is a quite simple algorithm that if templates used in only malicious traffic of training data is used in packets of the test data even once, the packets are judged to be malicious. To implement SVM, RF, and DNN, we use libsvm [9], scikit-learn [10], and TensorFlow [11], respectively.

Using the training set, we determined the most suitable models of the SVM algorithm and RF algorithm using the grid search approach, respectively. For each parameter, we applied the 5-fold cross-validation tests and computed the mean accuracy. We then picked up the best parameter that maximizes the mean accuracy. The DNN model consists of four layers: an input layer, two hidden layers, and an output layer. We use Adaptive Moment Estimation to optimize the classifier. The error and activated functions are based on

Table 2 Collected malware samples overview.

	# of collected samples	Malware type (Kaspersky)
Training data	24,260	not-a-virus:AdWare (39%), not-a-virus:HEUR:AdWare (29%), Undetectable (8.3%), not-a-virus:Downloader (7.9%), Trojan (5.2%), Trojan-Downloader (3.6%), HEUR:Trojan (2.3%), not-a-virus:WebToolbar (1.3%), not-a-virus:HEUR:Downloader (1.0%), Others (2.4%)
Test data	15,000	Undetectable (24%), not-a-virus:HEUR:WebToolbar (15%), HEUR:Trojan (12%), Trojan (8.0%), not-a-virus:AdWare (4.7%), P2P-Worm (4.0%), not-a-virus:Downloader (3.7%), Worm (3.0%), Trojan-Ransom (3.0%), Others (23%)

Table 3 Only malware samples using HTTP traffic overview.

	# of samples using HTTP	Malware type (Kaspersky)
Training data	18,164	not-a-virus:HEUR:AdWare (37%), not-a-virus:AdWare (33%), Undetectable (10%), not-a-virus:Downloader (9.5%), Trojan-Downloader (3.9%), HEUR:Trojan (2.3%), not-a-virus:WebToolbar (1.7%), Others (2.6%)
Test data	3,593	not-a-virus:HEUR:WebToolbar (34%), Undetectable (22%), HEUR:Trojan (11%), not-a-virus:AdWare (5.4%), Trojan-Ransom (4.0%), not-a-virus:HEUR:AdWare (3.6%), Trojan (3.1%), Others (17%)

cross-entropy and Rectified Linear Unit, respectively. We set the learning rate as 0.0001, the batch size as 200, and the number of training sessions as 9,000. The numbers of nodes in each layer (from the input layer) were X, 500, 50, and 2, respectively where we change the parameter X as the number of nodes in the input layer depends on the number of templates. Further details will be given in Sect. 4. To evade overfitting, we used the Dropout [12] technique, which invalidates the nodes that are randomly set at a specified ratio to mimic ensemble learning. Here, we configure the ratio to 50%, which was previously reported as the most efficient ratio [13].

3. Dataset

We prepared two data sets for the detection model; a training data (Table 4) and a test data (Table 5). Section 3.1 and Sect. 3.2 explain the training data and test data in detail. We note that to demonstrate the robustness of the approach, the training and test data are collected independently. We also note that malware samples were collected from different sources and there were no duplications among the training set and test set. Table 2 summarizes the overview of the malware samples we collected. For the types of malware samples, we adopted the notation used by Kaspersky, which established the highest detection rate. Of the malware samples shown in Table 2, we picked up the samples that generated HTTP communication. Table 3 summarizes the results. As shown in the table, the breakdown of malware samples used for training and test are quite different from those used for training. This difference reflects the fact that we collected samples using different sources as we will explain later. We note that as we shall see later, our approach works well despite the differences of malware breakdown. We also note that for test set, the fraction of malware samples that use HTTP communication is smaller than that for training set. We conjecture that the difference comes from the fact that we used Cuckoo Sandbox for test data. As Cuckoo Sandbox is an open source software, it is possible that some malware samples we collected for test successfully employed anti-sandbox techniques to evade from the dynamic analysis.

Table 4 Details of training data.

	# of samples	# of HTTP requests	collection periods
malicious	18,164	117,407	Dec 2014 - Sep 2015
benign	-	130,619	Aug 2016

Table 5 Details of test data.

	# of samples	# of HTTP requests	collection periods
malicious	3,593	77,110	Jun 2016
benign	-	79,751	Sep 2016

3.1 Training Data

3.1.1 malicious traffic

The training data for malicious traffic detection were compiled from 24,260 malware samples and their HTTP traffic data. Each malware sample was detected as malicious by using two antivirus softwares, TrendMicro [14] and Kaspersky [15]. No overlap exists among the samples. These malware samples were collected using the server-type honeypot system, which is proposed in [16] and the client-type honeypot system, which is proposed in [17], respectively. The malware samples were collected from December 2014 to September 2015. All the collected malware samples were analyzed using a commercial sandbox system that can perform dynamic malware analysis with the controlled Internet access. The system starts a guest Windows machine and executes each malware sample within a maximum of 5 minutes. In order to avoid any damages to our system and other hosts, these honeypots do not allow executing malware and reverting to initial state of a virtual machine. Several access controls are applied to the outgoing traffic that is often used to send spam e-mail and infect other host, e.g., SMTP and Net-BIOS, for possible efforts not to cause any damage

to other hosts. From the monitored packet data, we extracted the sequence of HTTP headers.

3.1.2 benign traffic

As the training data of benign traffic, we used the data collected at a campus network, which has the network prefix of /16. As the network is used by more than 30K of people regularly, we believe that the data represents an example of benign traffic. These data, of type pcap, were collected at the gateway in August 2016 using tcpdump. The number of concurrent unique IP addresses observed at the time of measurement was 1,110. We eliminated packets accessing the URLs that are registered to publicly available URL blocklists such as MalwareDomainBlocklist [18]. From the remaining packet data, we extracted the sequence of HTTP headers. Although not all of the devices connected to the campus network are guaranteed malware-free, the proportion of malicious traffic is expected to be extremely small so that we can assume these data as benign traffic.

3.2 Test Data

3.2.1 malicious traffic

The test data of malicious traffic was compiled from 15,000 malware samples and their traffic data. To avoid bias among samples, we collected 5,000 samples each from Malwr [19], MalShare [20], and VirusShare [21] in June 2016. We note that these malware data were randomly sampled from each site. All the sampled malware were registered to each site within one year from the date of our data collection; i.e., the sampled malware samples were fairly new at the time of our experiments. We also note that all the sampled malware were detected as malware by at least one anti-virus checker included in VirusTotal [22]. We note that there were no overlaps between the training and test data. The traffic data of the malware samples were collected over the same period as the malware collection using Cuckoo Sandbox [23] that is open source dynamic malware analysis system. The sandbox system is also connected to the Internet and analyzes on Windows machine to execute malware samples for 90 seconds. We extracted the sequence of HTTP headers from the monitored packet data.

3.2.2 benign traffic

The benign traffic as test data were collected at the same vantage point in September 2016. Again, by using publicly available URL blocklists, We eliminated all the packets that are likely associated with malicious activities, and extracted the sequence of HTTP headers from the remaining packet data.

4. Results

In this section, we first present how we determined the threshold used for the automatic template generation. We then show the primary results, the accuracy of the classification models. Finally, we demonstrate the effectiveness of the automatic template generation.

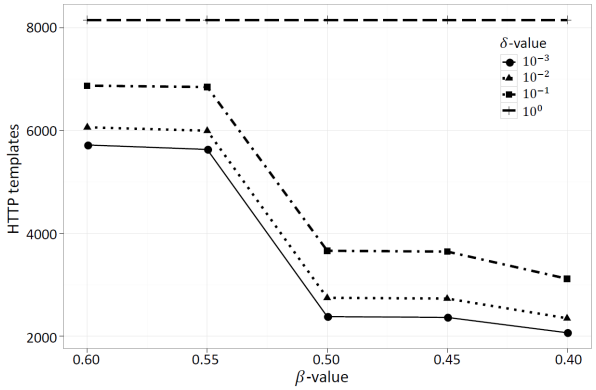

Fig. 4 The change in the number of templates given the different thresholds.

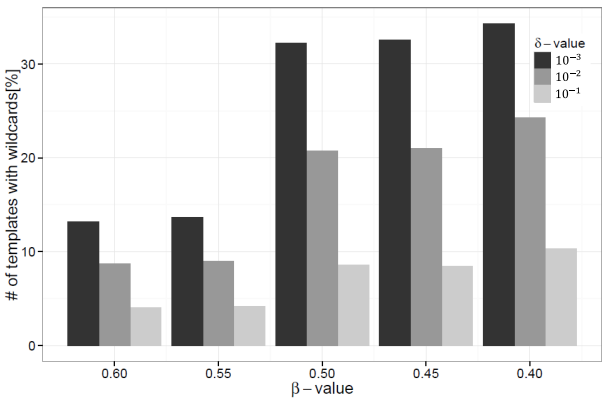

Fig. 5 The ratio of templates inserted wildcard characters.

4.1 Automatic Template Generation

We first present how we set the threshold β used in the automatic template generation. Figure 4 shows how the number of templates depends on the threshold, β. $\delta = 10^0$ corresponds to the case where no templates are created. The smaller the δ and β, the effect of reducing HTTP header fields is great. Also, according to Fig. 5, it is shown that the ratio of the templates with wildcard characters also increase with a similar tendency, and these occupy a maximum of about 35%, which shows that automatic template generation is very effective. In either case, the change in the number of templates is especially remarkable at $\beta = 0.5$.

Figure 6 shows the CDFs of len(F). As shown in the graph, roughly 50% of the fields had len(F) = 2. This result indicates that templates of these fields are created when β is less than 0.5. Given these observations, we set $\beta = 0.5$ as the most suitable threshold.

4.2 Classification Results

We compared the result of STM, SVM, RF, and DNN for test data. The ACC (Accuracy)s, FPR (False Positive Rate)s, FNR (False Negative Rate)s, and prediction times at various values of the threshold δ are listed in Table 6. Each result is averaged from five results, and prediction times are performed with the Ubuntu 13.10 (CPU: Intel Xeon CPU E5-2620 v2 @ 2.10 GHz, Memory: 64 GB). The ACC, FPR, and FNR are respectively calculated as follows:

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

TP and FP denote true positive and false positive, respectively. Similarly, TN and FN are true negative and false negative, respectively. Setting δ is 10^{-1} increases the ACC. The highest ACC 97.1% is achieved by DNN, closely followed by SVM. Although the DNN and SVM yield almost the same ACC, the prediction time of SVM is approximately 1.5 times that of DNN. As expected, the STM algorithm gives the best prediction time. However, its ACC is not high. RF also improves the prediction time at the expense of the ACC. From these results, we judged DNN to be the best model.

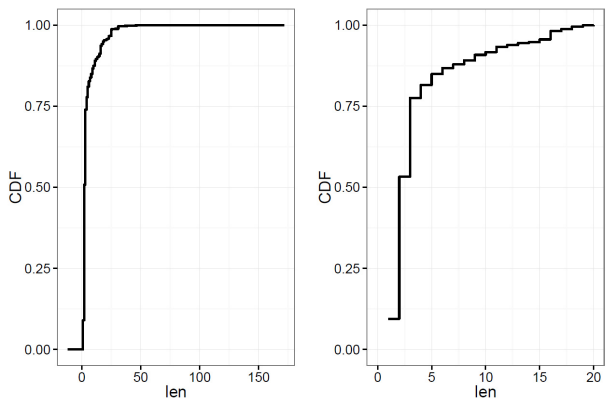

Fig. 6 CDFs of the number of words in the fields (len(F)). The left panel is the full-size view, and the right panel is the enlarged view.

4.3 Effectiveness of Automatic Template Generation

We study how the automatic template generation algorithm affects our results. For all the templates extracted with $\delta = 10^{-1}$ and $\beta = 0.5$, we calculated the mutual information (MI), which is one of the metrics that can tell you the contribution of a given feature to the classification. MI is formulated as follows:

$$MI(X;Y) = \sum_{i=1}^{m}\sum_{j=1}^{n} p(x_i, y_j) \log_2 \frac{p(x_i, y_j)}{p(x_i)p(y_j)},$$

where $p(x_i)$ refers to probability of x_i in random variable $X = \{x_1, x_2, \ldots, x_m\}$, $p(y_j)$ refers to probability of y_j in random variable $Y = \{y_1, y_2, \ldots, y_n\}$, and $p(x_i, y_j)$ expresses the simultaneous occurrence probability of x_i and y_j.

Table 7 shows the top-10 templates that had the largest MI. As we see from the table, the templates with the wildcard characters inserted had high MI. This result indicates that automatic template generation was effective not only in reducing the number of templates but in extracting useful features that contributed to the classification.

Figure 7 shows the keys that had the highest numbers of occurrences. The left panel shows the top-10 keys without applying the template generation and the right panel shows the top-10 keys after applying the template generation. We

Table 6 Performance of each model. ACC, FPR, and FNR stand for accuracy, false positive rate, and false negative rate, respectively.

δ	# of features	model	ACC (mean/std)	FPR (mean/std)	FNR (mean/std)	prediction time (mean/std) [s]
10^{0}	2,405	STM	0.856/0.000	0.006/0.000	0.361/0.000	0.234/0.005
10^{0}	8,155	SVM	0.923/0.000	0.004/0.000	0.152/0.000	310.7/12.67
10^{0}	8,155	RF	0.772/0.006	0.004/0.000	0.460/0.012	120.7/2.853
10^{0}	8,155	DNN	0.930/0.002	0.005/0.001	0.137/0.006	196.0/0.215
10^{-1}	616	STM	0.842/0.000	0.006/0.000	0.314/0.000	0.210/0.013
10^{-1}	3,662	SVM	0.962/0.000	0.007/0.000	0.070/0.000	134.4/3.591
10^{-1}	3,662	RF	0.824/0.095	0.007/0.000	0.350/0.193	47.54/1.527
10^{-1}	**3,662**	**DNN**	**0.971/0.000**	**0.007/0.000**	**0.052/0.000**	**88.38/0.182**
10^{-2}	413	STM	0.605/0.000	0.006/0.000	0.797/0.000	0.208/0.007
10^{-2}	2,745	SVM	0.960/0.000	0.008/0.000	0.074/0.000	108.5/3.192
10^{-2}	2,745	RF	0.706/0.013	0.008/0.002	0.589/0.026	39.12/1.589
10^{-2}	2,745	DNN	0.876/0.080	0.011/0.003	0.240/0.165	65.12/0.093
10^{-3}	366	STM	0.605/0.000	0.009/0.000	0.794/0.000	0.210/0.000
10^{-3}	2,382	SVM	0.959/0.000	0.008/0.000	0.074/0.000	99.26/2.470
10^{-3}	2,382	RF	0.711/0.012	0.007/0.000	0.581/0.025	34.77/1.545
10^{-3}	2,382	DNN	0.894/0.060	0.012/0.003	0.204/0.124	56.52/0.179

Table 7 The top-10 templates with the highest MI. We truncated the long templates.

HTTP templates	MI
User-Agent: Mozilla 4.0 (compatible; MSIE * Windows NT 5.1; Trident 4.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; ...	0.1523
User-Agent: Mozilla 4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident 4.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; ...	0.1371
Cache-Control: * 0	0.1298
Cache-Control: max-stale 0	0.1177
Accept-Encoding: gzip deflate *	0.1067
Accept-Encoding: gzip deflate sdch	0.1055
Pragma: *	0.0995
Pragma: no-cache	0.0993
Connection: *	0.0843
Content-length: *	0.0786

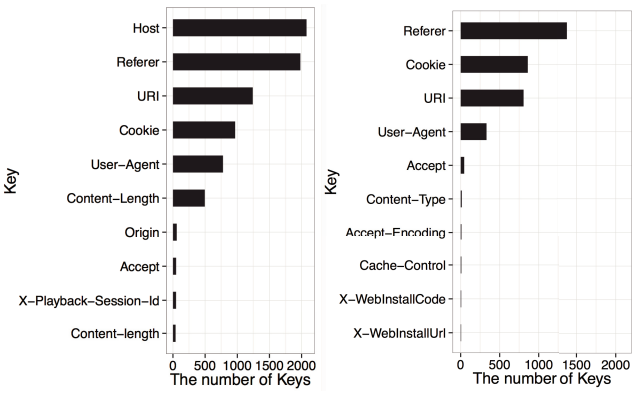

Fig. 7 The effect of template generation. Top-10 keys in the features. The left panel shows the result when β is set to 0.5 and δ is set to 1.0 (no template created). The right panel shows the result when β is set to 0.5 and δ is set to 0.1.

see that many of Host fields and Content-Length fields are removed from the keys by applying the template generation algorithm. This observation demonstrates that the template generation approach works reducing feature vectors. We also see that the template generation does not remove other fields such as Referer field, Cookie field, URI field and User-Agent field. Referer field and Cookie field are the fields when a client host accesses to a web server via a web browser; it implies that their existence is important while their values are less important. URI field and User-Agent field are known to be useful features as previous studies demonstrated [24]–[27]. Thus, the template generation algorithm can reduce less meaningful features while keeping important features.

4.4 Summary

Finally, we summarize our findings. The results shown in the previous subsections indicate that our methodology is robust and scalable. For instance, as shown in Table 7, we can see that wildcard character represents the variations of version information for "MSIE". The result indicates that the template can also express newer versions of "MSIE" that may emerge in the future. Also, since our training data and test data were collected from different vantage points, the high accuracy of 97% indicates that our scheme is robust against the data variations. We note that the processing time required for the machine learning algorithms is short. For instance, for DNN, which is the most accurate model, it took 88 seconds to process 156,861 HTTP requests, i.e., processing rate is 1,783 requests/s, which is fast enough to cope with Internet backbone traffic.

5. Discussion

In this section, we first discuss the results of each model and causes of errors. We then discuss several limitations of our approach. Finally, we discuss the deployment of the *BotDetector*.

5.1 Results of Each Model

We review the results of each model shown in Table 6. By generating templates, only the STM accuracy is gradually decreasing. Templates that have successfully inserted wildcard characters are more handy features, so it is reasonable to suppose that the accuracy of this model which depends on only malicious features decreases. We first notice that RF did not result in the good accuracy for our dataset. Other two Machine Learning (ML) algorithms established fairly good accuracies. SVM is one of the most popular machine learning algorithms. It showed the second highest accuracy after DNN in this experiment. DNN achieved the highest accuracy among the algorithms we used. The disadvantage of DNN is the cost for training, which can be shortened by using GPU.

5.2 Causes of Errors

Although the accuracy of our approach was high, the prediction errors are not zero. Namely, for DNN, FPR and FNR are 0.7% and 5.2% when $\beta = 0.5$ and $\delta = 10^{-1}$, respectively. We found that many of false positives are associated with the number of HTTP header fields in a packet. Figure 8 shows the CDFs of the number of HTTP header fields in each packet. Malicious traffic tends to comprise of a small number of fields. We can interpret the result that benign traffic originated from a browser, which may utilize many options; malicious traffic is originated from a program, but not browsers. Due to this nature, if a benign packet contains a small number of HTTP header fields, it could be falsely detected as malicious. One way to fix this problem

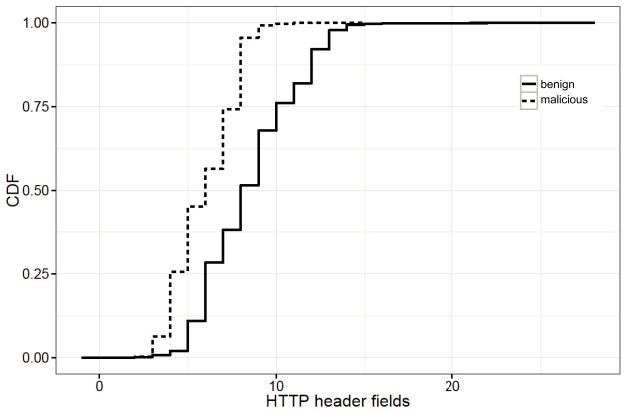

Fig. 8 The CDF of the number of HTTP header fields in HTTP request packets.

is not to make our decision when we do not have enough information. We also found that false negatives are associated with the malware that likely communicates through real web browsers. As we mentioned earlier, the majority of benign web traffic is originated from web browsers. We discuss this issue in the next subsection. We note that simply using the information about browser/non-browser is not useful to distinguish between benign and malicious traffic. That is, benign traffic includes not only browser traffic, but also traffic generated by dedicated software such as online storage sync tool. Likewise, malicious traffic includes not only non-browser traffic, but also traffic generated by using a web browser or a software that uses HTTP User-agent mimicking a real browser as shown in Table 7.

5.3 Limitations

BotDetector makes use of the HTTP header fields; hence, the system cannot react when traffic is encrypted, i.e., when communication is made with HTTPS. In addition, although the HTTP protocol is one of the most popular protocols used by malware, the current system cannot detect malware-infected traffic based on other protocols. For instance, UDP-based protocols may not be able to be captured with our approach. We note that the scope of our work does not cover certain types of malware samples such as those used for spear-phishing email attack. As we have shown, our approach successfully detected malware samples uch as *Adware*, *Trojan*, *Worm*, *Downloader*, *Ransomware*, *etc.*, which all make use of HTTP as a means of communication. We believe our approach works in a wide range as HTTP is a common way of communication widely used for various malware families. Another limitation of *BotDetector* is that malware developers can change the HTTP headers to evade detection; i.e., the traffic originated from malware can mimic the traffic originated from a browser. Although the case has not been major so far, such evasion could become standard in the future, at which point, we need to change the feature extraction and classification model. We also note that our targets were limited to Windows malware. As the analysis of HTTP headers is independent from the system architecture, our approach should work for malware of other platforms. Verification of our approach using malware samples of other platforms is left for future study.

Despite of these limitations, we believe that the fundamental idea behind this work – finding useful features automatically – remains beneficial to discover invariants that could be used to detect malicious activities.

5.4 Deployment

Given the features of the *BotDetector*, we suggest that this system is deployed as a part of security appliance system, which provides a set of functionalities to protect users from malicious traffic. The most suitable place to install *BotDetector* is the backbone network link where many customer traffic is aggregated. As we validate the high scalability of our approach, it can handle a huge volume of incoming/outgoing HTTP request packets. Another possible implementation form is to install the *BotDetector* functionality into web proxy servers where HTTP header information for many end-users can be monitored. We leave the actual field test for our future study.

6. Related Works

In this section, we review several studies that used HTTP information for detecting malware [24]–[30]. We also discuss how our approach is different from these past studies.

Zhang et al. [24] made use of the User-Agent field as the useful feature for detecting malware. They demonstrated that regular expression could be used to characterize HTTP headers. They also confirmed that a fake User-Agent could be identified together with the information extracted with the OS fingerprinting technique. Grill et al. [25] also used the User-Agent field to detect malware communication. They found that User-Agent can be classified into the five patterns: *Legitimate users browser*, *Empty*, *Specific*, *Spoofed*, and *Discrepant*. According to their findings, some malware uses User-Agent of the web browser utilized by the actual owner of the device; in such a case, it's hard to detect malware communication simply from the User-Agent information. Nelms et al. [26] proposed a system called *ExecScent*, which is the closest work to ours. It aims to detect bot using the entire HTTP header fields. They manually created templates using the domain knowledge, i.e., URL path, query, User-Agent, etc. They characterize the templates with a regular expression. We note that our approach automated the template generation scheme; thus, we do not need to employ manual inspection to create the useful templates.

Chiba et al. [27] developed a system called *BotProfiler*. *BotProfiler* is a system that aimed to improve the performance of *ExecScent* by using URL path, URL query, and User-Agent information. Like the *ExecScent* system, the *BotProfiler* system also requires building manually crafted templates. Xie et al. [28] proposed the system called

AutoRE which generates regular expression signatures from URL structure to detect botnet-based spam emails and botnet membership. Zallas et al. [29] also proposed the idea of generating templates, which are the name-value pairs of the HTTP headers. The generated templates can be used to detect HTTP-based malware. While these prior studies generated templates using several domain-specific heuristics, our method makes use of the statistical approach, which enables us to construct the templates without requiring any domain-knowledge in advance. Thus, our approach is more generic and tolerant to the changes of HTTP header patterns.

Perdisci et al. [30] proposed a system that performs the clustering of network-level malware behavior. Using the clusters, they generated signatures that can be used to detect malware. As their approach makes use of both the HTTP request and response packets, it cannot handle a case where a C2 server has changed its IP address; i.e., a request packet originated from a client will not reach to the server and no response packets will be observed. Such cases are common when malware samples are executed. In contrast, our approach works by just using HTTP request packets; thus, it can handle the case where a client does not receive a response packet from a server-side. We also note that our approach establishes high accuracy despite the fact that we only used the request packets.

While all these past studies heavily rely on the domain knowledge when they extract useful features to detect malware, our approach aimed to automate the feature extraction process by making use of the automatic template generation algorithm. We also note that DNN enables us to perform the *feature learning*; i.e., it automatically expresses useful features with the neural network. We believe that our approach has an advantage over the strategies used in the past studies because ours is robust to the change in the features of malware communication.

7. Conclusion

We proposed a system called *BotDetector* for detecting malicious traffic, thereby searching for malware-infected devices. The key ideas of our research were to create "templates" automatically for gathering information of each HTTP header field and using machine learning technique for detection. As a result of the extensive experiments using large-scale datasets, we demonstrated that *BotDetector* successfully detected malicious traffic with up to 97.1% precision and a low false detection rate below 1.0%. One noteworthy technical contribution of this work is that introduction of the automatic template generation algorithm, which contributes not only to reduce the amount of information to be kept but also to extract useful features. We believe that the key ideas and approaches used in this paper are useful for other studies that attempt to classify malicious activities given a large number of features.

Acknowledgments

We thank Dr. Tatsuaki Kimura for his valuable comments on the automatic template generation algorithm. A part of this work was supported by JSPS Grant-in-Aid for Scientific Research B, Grant Number 16H02832.

References

[1] S. Mizuno, M. Hatada, T. Mori, and S. Goto, "Botdetector: A robust and scalable approach toward detecting malware-infected devices," IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017, pp.1–7, 2017.
[2] "McAfee Labs Threats Report: April 2017." https://www.mcafee.com/us/security-awareness/articles/mcafee-labs-threats-report-mar-2017.aspx.
[3] "ESET predictions and trends for cybercrime in 2016." http://www.welivesecurity.com/2015/12/23/eset-predictions-for-cybercrime-trends-in-2016/.
[4] "Large CCTV Botnet Leveraged in DDoS Attacks." https://blog.sucuri.net/2016/06/large-cctv-botnet-leveraged-ddos-attacks.html.
[5] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: Bare-metal analysis-based evasive malware detection," Proc. 23rd USENIX Conference on Security Symposium, USENIX Security '14, pp.287–301, Aug. 2014.
[6] J.P. John, A. Moshchuk, S.D. Gribble, and A. Krishnamurthy, "Studying spamming botnets using botlab," NSDI, pp.291–306, 2009.
[7] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, "Spatio-temporal factorization of log data for understanding network events," IEEE INFOCOM 2014-IEEE Conference on Computer Communications, pp.610–618, IEEE, 2014.
[8] M. Ester, H.P. Kriegel, J. Sander, X. Xu, et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," Kdd, pp.226–231, 1996.
[9] "LIBSVM – A Library for Support Vector Machines." http://www.csie.ntu.edu.tw/~cjlin/libsvm/.
[10] "scikit-learn: machine learning in Python." http://scikit-learn.org/stable/.
[11] "TensorFlow - an Open Source Software Library for Machine intelligence." https://www.tensorflow.org/.
[12] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol.15, no.1, pp.1929–1958, 2014.
[13] P. Baldi and P.J. Sadowski, "Understanding dropout," in Advances in Neural Information Processing Systems 26, ed. C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, pp.2814–2822, Curran Associates, Inc., 2013.
[14] "Content security software - Internet Security & Cloud - Trend Micro USA." http://www.trendmicro.com.
[15] "Kaspersky Lab |Antivirus Protection & Internet Security Software." http://www.kaspersky.com/.
[16] K. Aoki, Y. Kawakoya, M. Akiyama, M. Iwamura, T. Hariu, and M. Itoh, "Investigation and understanding active/passive attacks," IPSJ Journal, vol.50, no.9, pp.2147–2162, 2009. (in Japanese).
[17] 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, 2010.
[18] "DNS-BH - Malware Domain Blocklist." http://www.malwaredomains.com/.
[19] "Malwr - Malware Analysis by Cuckoo Sandbox." https://malwr.com/.

[20] "MalShare." http://malshare.com/.
[21] "VirusShare.com." https://virusshare.com/.
[22] "VirusTotal - Free Online Virus, Malware and URL Scanner." https://www.virustotal.com/.
[23] "Cuckoo Sandbox: Automated Malware Analysis." https://www.cuckoosandbox.org/.
[24] Y. Zhang, H. Mekky, Z.-L. Zhang, R. Torres, S.-J. Lee, A. Tongaonkar, and M. Mellia, "Detecting malicious activities with user-agent-based profiles," International Journal of Network Management, vol.25, no.5, pp.306–319, 2015.
[25] M. Grill and M. Rehak, "Malware detection using http user-agent discrepancy identification," 2015 National Conference on Parallel Computing Technologies (PARCOMPTECH), pp.221–226, IEEE, 2015.
[26] T. Nelms, R. Perdisci, and M. Ahamad, "Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates," Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13), pp.589–604, 2013.
[27] 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," 2015 IEEE Trustcom/BigDataSE/ISPA, pp.758–765, IEEE, 2015.
[28] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: signatures and characteristics," ACM SIGCOMM Computer Communication Review, vol.38, no.4, pp.171–182, 2008.
[29] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of http-based malware," 2014 Twelfth Annual International Conference on Privacy, Security and Trust (PST), pp.249–256, IEEE, 2014.
[30] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," NSDI, p.14, 2010.

Sho Mizuno was born in 1993. He recieved B.E. degree in computer science and engineering from Waseda University in 2016. He is a graduate student at Department of Computer Science and Communication Engineering, Waseda University. He has been conducting projects in network system and security.

Mitsuhiro Hatada was born in 1978. He is currently a Ph.D. student with particular interest in anti-malware. He received his B.E. and M.E. degrees in computer science and engineering from Waseda University in 2001 and 2003, respectively. He joined NTT Communications Corporation in 2003 and has been engaged in the R&D of network security and anti-malware. He is a member of IEICE and IPSJ.

Tatsuya Mori is currently an associate professor at Waseda University, Tokyo, Japan. He received B.E. and M.E. degrees in applied physics, and Ph.D. degree in information science from the Waseda University, in 1997, 1999 and 2005, respectively. He joined NTT lab in 1999. Since then, he has been engaged in the research of Internet measurement and Internet security. Dr. Mori is a member of ACM, IEEE, IEICE, IPSJ, and USENIX.

Shigeki Goto is a professor at 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 a 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.