Hindawi
Security and Communication Networks
Volume 2019, Article ID 2182615, 11 pages
https://doi.org/10.1155/2019/2182615

Research Article

Using XGBoost to Discover Infected Hosts Based on HTTP Traffic

Weina Niu,[1] **Ting Li**,[1] **Xiaosong Zhang**,[1,2] **Teng Hu**,[1,3] **Tianyu Jiang**,[1] **and Heng Wu**[4]

[1]*School of Computer Science and Engineering, Institute for Cyber Security, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China*
[2]*Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, Guangdong 518040, China*
[3]*Institute of Computer Application, China Academy of Engineering Physics, Mianyang, Sichuan 621900, China*
[4]*Glasgow College, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China*

Correspondence should be addressed to Teng Hu; mailhuteng@gmail.com

Received 26 April 2019; Revised 27 September 2019; Accepted 9 October 2019; Published 6 November 2019

Guest Editor: Mehdi Hussain

In recent years, the number of malware and infected hosts has increased exponentially, which causes great losses to governments, enterprises, and individuals. However, traditional technologies are difficult to timely detect malware that has been deformed, confused, or modified since they usually detect hosts before being infected by malware. Host detection during malware infection can make up for their deficiency. Moreover, the infected host usually sends a connection request to the command and control (C&C) server using the HTTP protocol, which generates malicious external traffic. Thus, if the host is found to have malicious external traffic, the host may be a host infected by malware. Based on the background, this paper uses HTTP traffic combined with eXtreme Gradient Boosting (XGBoost) algorithm to detect infected hosts in order to improve detection efficiency and accuracy. The proposed approach uses a template automatic generation algorithm to generate feature templates for HTTP headers and uses XGBoost algorithm to distinguish between malicious traffic and normal traffic. We conduct a performance analysis to demonstrate that our approach is efficient using dataset, which includes malware traffic from MALWARE-TRAFFIC-ANALYSIS.NET and normal traffic from UNSW-NB 15. Experimental results show that the detection speed is about 1859 HTTP traffic per second, and the detection accuracy reaches 98.72%, and the false positive rate is less than 1%.

1. Introduction

With the booming of the Internet and the popularity of computers, today's computers are facing serious security problems, whose biggest cause is the explosive growth of malicious code. The malicious code refers to a computer code that is intentionally written by individuals or organizations to pose a security risk to a computer or network. It usually contains malicious sharing software, adware Trojans, viruses, worms, etc., each of which has different kinds of variants [1–5]. In the first half of 2018, China Internet Security News from 360 Internet Security Center shows that a total of 140 million new malicious programs were intercepted and an average of 795,000 new malicious programs were intercepted every day. Among them, the number of malicious programs on the PC side was 14,098,000, and an average of 779,000 new malicious programs were intercepted every day [6]. In the fourth quarter of 2017, McAfee Labs detected the highest number of new malware in history, with a total of 63.4 million new samples. McAfee Labs records an average of eight new malware samples per second, a significant increase from the four new samples recorded in the third quarter [7]. The malware not only brings huge economic losses to users, but also rapid changes have brought great trouble and pressure to the antikilling technology of malicious programs. The current technology has been difficult to detect malware before the host is infected.

Based on this background, detecting malware-infected hosts in network traffic can make up for the shortcoming [8] because most malware will communicate with externally hosted command and control (C&C) servers using the HTTP protocol after infecting the device. The C&C server is

the control center that sends malware execution commands, and it is where malware collects data. After an attacker attacks the host with malware, the controlled host sends a connection request to the C&C server. The traffic generated by the connection is malicious external traffic. Currently, there are two main ways to detect malicious external traffic. One is to filter malicious domain names based on blacklists, and the other is to use rules to match malicious external traffic. Both of these solutions have certain limitations. The blacklist-based filtering scheme can only identify malicious external traffic when connecting to a known malicious website and has no perception of domain name changes. However, based on the feature detection scheme, it is necessary for the security practitioner to analyze the samples one by one, which consumes large manpower and is difficult to detect the malicious external connection traffic of the variant.

As a supplement to the prior art, malicious traffic can be detected through machine learning. Using machine learning to discover the commonality between malicious traffic and use it as a basis to detect malicious traffic, a good algorithm can greatly reduce the workload of security practitioners. Specifically, the contributions of this work are specified as follows:

(1) We propose an approach-combined machine learning and HTTP header template to discover traffic involved in malware infection and develop it into the MalDetector system.

(2) We use the statistical technique to aggregate similar features of HTTP header fields, which is also called HTTP header template, from large-scale network traffic.

(3) We use the GridSearchCV function to coordinate the eXtreme Gradient Boosting (XGBoost) algorithm and verify their effectiveness in the dataset consisting of malicious external traffic generated from malicious samples from MALWARE-TRAFFIC-ANALYSIS.NET [9] running in the sandbox and the UNSW-NB 15 dataset [10].

The structure of this paper is arranged as follows. We introduce the related work in Section 2. Section 3 presents an overview of the proposed approach. The process of template automatic generation from the HTTP header is described in Section 4. Section 5 completes the experimental evaluation metrics and illustrates the experimental results. We make a conclusion of the paper in Section 6.

2. Related Work

At present, the malware traffic identification approach based on HTTP traffic mainly focuses on two aspects [11–24]; one is based on the request and response statistical features [11–16] and the other is based on the content of the HTTP packet [17–26].

2.1. The Request and Response Statistical Features. The approach mainly analyzes the behavior characteristics of HTTP request/response time interval, quantity, and packet size to model malicious behavior and identify malware traffic. Perdisci et al. [11] developed a novel network-level behavioral malware clustering system. They performed coarse-grained clustering through statistical features, such as the total number of HTTP requests, the number of GET requests, the number of POST requests, the average length of the URLs, the average number of parameters in the request, the average amount of data sent by POST requests, and the average response length. Then, they performed fine-grained clustering by calculating the difference in URL structure between two malware samples. At last, they merged together fine-grained clusters of malware variants that behave similarly enough. Their work can be able to unveil similarities among malware samples that may not be captured by current system-level behavioral clustering systems. Ogawa et al. [12] extracted new features such as HTTP request interval, body size, and header bag-of-words from HTTP request/response pairs and calculated cluster appearance ratio per communication host pairs and identified malware originated communication host pairs. However, the identification approach based on the request and response statistical features is limited to malware samples that perform some interesting actions (i.e., malicious activities) during the execution time T. The identification approach based on the content of HTTP requests and responses can overcome this limitation.

2.2. The Content of HTTP Packets. The approach performs an analysis of the content of HTTP requests and responses, extracts relevant field information to process it, and combines machine learning algorithm to identify malware traffic. Zhang et al. [17, 18] used a learning-based approach to discover dependencies of network with the help of HTTP request features and thus detect malicious traffic. Srivastava et al. [19] developed a system called ExecScent that is closest to this work. They used all the HTTP header fields to detect botnet traffic. They manually created templates by themselves, such as URL-Path, Query, and User-Agent, and formatted them using regular expressions. Zhang et al. [20] proposed a method that used the User-Agent field to detect malicious external traffic generated by malware. They used regular expressions to format HTTP header information and used the operating system's fingerprint technology to identify whether it was a fake user agent domain to infer if there was a malware infection. Grill and Rehak [21] also used the User-Agent field to detect the presence of malicious external traffic. They found that all User-Agent field information can be divided into five categories: legitimate user browser information, null, specific, spoofed, and inconsistent. According to their findings, some malware deliberately forged requests that were sent from a web browser, making it difficult to detect malicious outbound traffic from the User-Agent field. Li et al. [22] proposed MalHunter based on behavior-related statistical characteristics. They detected malware communication patterns from three types of features: character distribution of the URL, HTTP header fields, and HTTP header sequence. However, these

approaches are either based on a single field or based on all fields, and their feature validity is low.

Moreover, Zhang et al. [23] presented a system SMASH that uses unsupervised data mining methods to detect various attack activities and malicious communication activities, focusing on detecting malicious HTTP activity from the perspective of server-side communication. Mekky et al. [24] put forward a method for identifying HTTP redirected malicious links. They built per-user chains from passively collected traffic and extracted new statistical features from them to capture the inherent characteristics of malicious redirect cases. The supervised decision tree classifier is then applied to identify malicious links. Liu et al. [25] proposed an identification approach by analyzing HTTP connections established by clients in a monitored network and combining stream classification with graph-based fractional propagation methods to identify previously undetected Internet Service Provider (ISP) networks.

3. HTTP-Based Infected Host Detection Approach

The proposed HTTP-based infected host detection system includes four modules: HTTP traffic filtering, header feature extraction, template automatic generation, and infected host detection. Figure 1 gives an overview of the framework of our proposed infected host detection approach using HTTP traffic.

3.1. HTTP Traffic Filtering and Header Feature Extraction. We save the HTTP header to reduce the amount of stored traffic. We also select the important information from the HTTP header for further analysis. The number of distinct HTTP header fields could be roughly 10 K. Moreover, some unrelated features may expose the machine learning model to the risk of overfitting. Rare fields are nonversatile, so the selection criteria are that we do not extract fields that appear less than 10 times or never appear in training data.

In addition, we mainly focus on the detection of malware that leverages the HTTP as the primary channel to communicate with the C&C server or to launch attack activities. Thus, our approach mainly focuses on HTTP requests rather than responses. If the C&C server is temporarily offline or changes its response content, there is little impact on our detection capabilities. Therefore, the selected fields are URI, Host, User-Agent, Request-Method, Request-Version, Accept, Accept-Encoding, Connection, Content-type, Cache-Control, Content-length, and some identification fields like Frame-time, srcIP (source IP), srcPort (source port), dstIP (destination IP), and dstPort (destination port).

Table 1 lists the description of the selected fields. The reason for selecting them is that they are often used in HTTP traffic and may be helpful in distinguishing legitimate traffic and malicious traffic.

3.2. Template Automatic Generation. When malware communicates with externally hosted C&C servers, malware developers typically use custom formats to construct packets. The network traffic generated by the malware belonging to the same family usually has a similarity. Therefore, we use statistical techniques to aggregate similar features of the HTTP header fields, that is, to generate similar templates for malicious traffic, and then use the template to detect new malicious traffic. A template is a series of strings, the character part represents the same part of the value of an HTTP header field, and $*$ represents the different parts of the value of the header field. Templates are generated to display the variability of words constituting the HTTP header fields and aim to compress their information. The template automatic generate module consists of three steps: scoring, clustering, and generating templates [27], which is explained in detail in Section 4.

3.3. Infected Host Detection. Many winners in Kaggle's competitions like to use XGBoost [28] due to Parallelization, Distributed Computing, Out-of-Core Computing, and Cache Optimization of data structures and algorithms. Thus, we use the XGBoost algorithm to classify malicious traffic and normal traffic in this work.

4. Template Automatic Generation

This section introduces focuses on how template automatic generation algorithm works.

4.1. Scoring. We first calculate the score for each value of the selected HTTP header fields by using the score calculation method, and then sort each selected HTTP header field's values according to their scores. Each field in the HTTP header is divided by the following four separators: space, "/", "=", and ",". Thus, the score calculation method is that we split each selected HTTP header field by separator and then calculate the percentage of their values' scores. For a value w in the field F, its score is $S(w; F)$, which can be calculated using

$$\begin{aligned} S(w; F) &= P(w \mid \mathrm{pos}(w, F), \mathrm{len}(F)) \\ &= \frac{n(w, \mathrm{pos}(w, F), \mathrm{len}(F))}{n(\mathrm{pos}(w, F), \mathrm{len}(F))}, \end{aligned} \tag{1}$$

where $\mathrm{pos}(w, F)$ is the position of the value in the field F, $\mathrm{len}(F)$ is the number of values in the field F. For example, $F = \{\mathrm{foo}, \mathrm{bar}, \mathrm{baz}, \mathrm{quz}\}$, $w = \mathrm{bar}$, $\mathrm{pos}(w, F) = 2$, and $\mathrm{len}(F) = 4$. $n(X)$ is the number of times that X appears in all the HTTP header, $n(w, \mathrm{pos}(w, F), \mathrm{len}(F))$ represents the number of times that w appears in all the pos field of all data, and $n(\mathrm{pos}(w, F), \mathrm{len}(F))$ indicates the number of times that the pos field appears in all the HTTP header. As shown in Figure 2, the score of "rv: 19.0" is 0.33 ($S(w, F) = 1/3 = 0.33$).

4.2. Clustering. We use the idea of the DBSCAN [29, 30] algorithm to cluster the values of the selected HTTP header fields. In the selected HTTP header field, when the score of the next value differs from the score of the previous value by

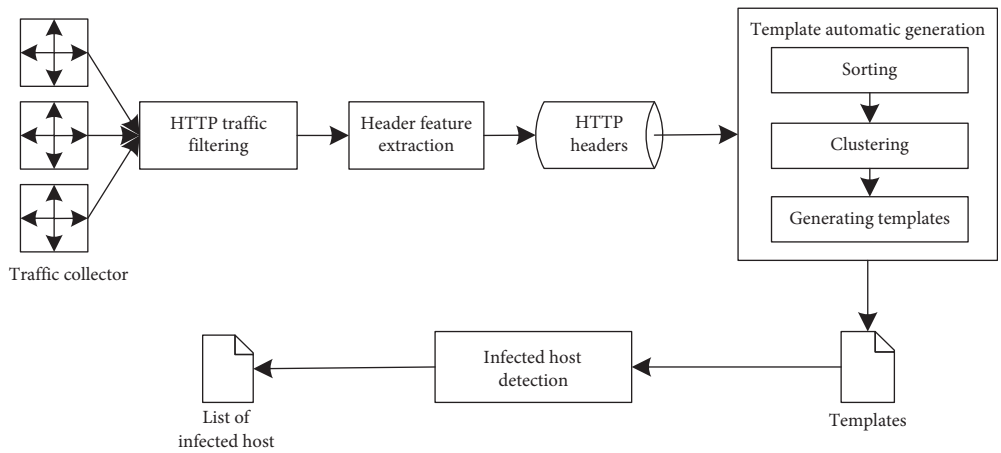


Figure 1: The framework of our proposed approach.

Table 1: The description of the selected fields in HTTP request header.

The selected fields name	Description
URI	URI is uniform resource identifier, a reference for resources available on the Internet such as HTML documents, images, or videos, and the URI field plays an important role in detection for malicious traffic [19, 22]
Host	The Host field recorded the domain name of the server and the TCP port number monitored by the server
User-Agent	The User-Agent field is still an effective indicator of compromised hosts because malware may carry the fake browser-like information or its own unique identification
Request-Method	Request type
Request-Version	HTTP protocol version
Accept	This field contains media type information and relative priority of media type
Accept-Encoding	The information in Accept-Encoding field is an encoding method of the content received by the client, and it is usually some kind of compression algorithm
Connection	The Connection field represents a connection state of the client and the server
Content-type	The value of the Content-type could help our model filter some legal traffic. The HTTP protocol carries data transmission of various types, such as text, pictures, sounds, videos, and others. Legal traffic tends to vary significantly. In contrast, most malware chooses text-related values such as textl/html; charset = UTF-8
Cache-Control	Cache-Control message indicating a request caching mechanisms need to be implemented
Content-length	This field indicates the size of the entity-body

less than δ, the next value is added as the current cluster; otherwise, the next value is added to the other clusters. Repeat the above process until all values have been added to the cluster. Here, the DBSCAN algorithm requires two parameters: scan radius (*eps*) and minimum inclusion points (*minPts*). The working process of the DBSCAN algorithm is as follows.

Starting with an unvisited point and finding all nearby points within the *eps* (including *eps*). If the number of nearby points is not smaller than *minPts*, the current point forms a cluster with its nearby points, and the starting point is marked as visited. Then recursively, all the points in the cluster that are not marked as visited are processed in the same way, thereby expanding the cluster. If the number of nearby points is smaller than *minPts*, the point is temporarily marked as a noise point. If the cluster is fully extended, i.e., all points within the cluster are marked as accessed, then the same algorithm is used to process the unvisited points.

Finally, we descript our clustering approach with the scoring method and DBCSAN algorithm in the following.

First, we need to introduce the following two parameters: ($\delta \geq 0$) and β ($0 < \beta < 1$), δ is the minimum distance between two clusters, $\beta \times \text{len}(F)$ for the minimum number of points in the cluster, and len(F) refers to the number of value in a field. In this work, the δ is set to 0.1 and β is set to 0.5.

Then, we sorted each word in descending score. When the score of the next word differs from the mean score of a cluster by less than δ, the next word is added to the current cluster. Otherwise, the next word is assigned to a new current cluster. This process is repeated until all words are included in either cluster.

4.3. Generating Templates. The results of the clustering are filtered to preserve only the clusters whose values are larger than $\beta \times \text{len}(F)$ and the remaining clusters are replaced with "$*$", where $\ddot{a}$ is the minimum distance between two clusters, whose value is not smaller than 0; $\beta \times \text{len}(F)(0 < \beta < 1)$ is the minimum number of points in the cluster, and len(F) is the number of values of the field. The overall generation process is shown in Figure 2.

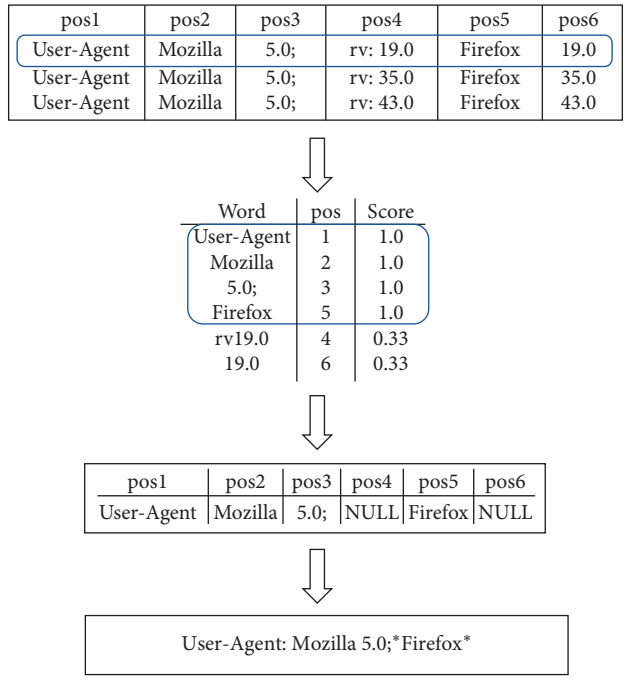


Figure 2: Template generation process.

The generated HTTP header field information and HTTP template are shown in Table 2.

We also performed statistics on the templates generated by the training data. The statistical results are shown in Figure 3.

As can be seen from Figure 3, the number of templates for malicious traffic is generally several times larger than the number of templates for normal traffic. The maximum number of templates generated is the URI and User-Agent fields. It can be inferred that malicious traffic may be distinguished mainly based on templates of these several fields. It has been observed that some fields do not even have the generation of malicious traffic templates. It can be inferred that the HTTP request information of malicious traffic may be short, including only information of several fields. Probably because normal HTTP request traffic is usually a connection made through a browser, the browser logs information for many fields. Malicious traffic is a connection made to the C&C server through malware, and the data format is usually constructed by a malware developer, so the HTTP request message is shorter.

5. Experiments and Results

This section introduces the dataset, the experimental setup, the performance metrics, and the obtained results.

5.1. Dataset. The malware traffic used in this work is from MALWARE-TRAFFIC-ANALYSIS.NET [9]. We collect malicious external traffic by running malicious samples collected from June 2013 to December 2017 in the sandbox and use SecurityOnion (a tool for network security monitoring) to detect traffic and get the result. The normal traffic samples are from the UNSW-NB 15 dataset shared by the Cyber Range Lab of the Australian Cyber Security Center (ACCS) in 2015 [10]. They used the tcpdump tool to capture 100 GB of raw traffic (PCAP files) for evaluating network intrusion detection systems and gave a labeled dataset. The labeled file contains the time period, the source port, the source IP address, the destination port, the destination IP address, the protocol type, and other information of the threat traffic, which is shown in Table 3. There are 373864 HTTP request records and only 6401 malicious traffic records in the 100G raw traffic data. We remove malicious HTTP traffic based on source IP, destination IP, source port, destination port, and the time period (from the start time to the last time) in the given labeled file. When the protocol type is HTTP and the time period, source port, source IP, destination port and destination IP address are matched successfully, the traffic is labeled as malicious traffic.

We set the ratio of the training set to the testing data as 7 : 3. Thus, the dataset in the experiment is shown in Table 4, which consists of 34,239 malicious HTTP requests and 35,481 normal HTTP requests.

5.2. Experimental Setup. The system had been implemented in Python 3.5, and all experiments were performed using an off-the-shelf server with 64 GB of RAM memory and 6-core processor. In order to evaluate the true positive rates and false positive rates of our detection approach, we tune the model parameters on the training set. The initial key parameters of the XGBoost model are shown in Table 5.

Table 5 shows that the accuracy of cross-validation of the training set with the initial parameters is 99.5%, but the accuracy of the testing set is only 92.89% due to over-fitting.

Table 2: HTTP header field information and template comparison.

HTTP header field information	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 ∗

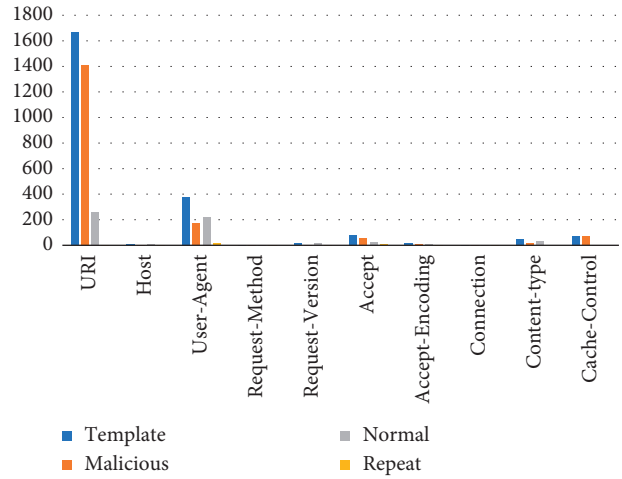


Figure 3: Template statistical histogram.

Table 3: The labeled file of UNSW-NB 15.

Start time	Last time	Attack category	Attack subcategory	Protocol	Src IP	Src port	Dst IP	Dst port
1421927415	1421927415	Exploits	Unix'r' Service	udp	175.45.176.3	21223	149.171.126.18	32780
1421927416	1421927415	Exploits	Brower	tcp	175.45.176.2	23357	149.171.126.16	80
1421927418	1421927415	Exploits	Cisco IOS	tcp	175.45.176.4	26939	149.171.126.10	80
1421927420	1421927415	DoS	IXIA	tcp	175.45.176.1	23910	149.171.126.15	80
1421927421	1421927415	Generic	Brower	tcp	175.45.176.2	23909	149.171.126.14	3000

Table 4: Dataset in the experiment.

Dataset	Traffic type	The number of HTTP request
Training set	Malicious traffic	27789
	Normal traffic	28915
Testing data	Malicious traffic	6450
	Normal traffic	6566

In order to further improve the accuracy of the prediction, we further adjust the parameters of the XGBoost algorithm.

We use the GirdSerachCV function in the SCIKIT-learn [31] package to adjust the parameters, which traverses the value range of parameters. We adjust three of the key parameters, and the adjustment steps are as follows:

(1) We first adjust two parameters *max_depth* and *min_child_weight* that play a decisive role in the model. The value range of *max_depth* is set to [4, 6, 8, 10, 12]. The value range of *min_child_weight* is very large and seriously affects the experimental results. If *min_child_weight* is over-fitting, the value of *min_child_weight* should be increased. Thus, its value range is set to [1, 10, 100, 1000]. The results of the parameter adjustment are shown in Table 6. The experimental results show that the model performs optimally when *max_depth* = 10 and *min_child_weight* = 1.

(2) Based on the adjusted *max_depth* and *min_child_weight* parameters, we adjust the parameter *gamma*, which participates in the pruning of the decision tree. The larger the value of the parameter is, the less the impact on the model is. Here, we set the value range of *gamma* to [0~8]. The results of the parameter adjustment are shown in Table 7. The experimental results show that the model with the best performance when *gamma* = 0.

(3) We adjust the two parameters *subsample* and *colsample_bytree* at last, which is related to the proportion of samples used. If the sampling setting ratio is too small, the accuracy may be reduced. Here, the value range of the *subsample* is set to [0.7~1], and the value range of *colsample_bytree* is also to [0.7~1]. The

Table 5: Experimental parameters settings.

Parameter	Description	Value
booster	Tree model	gbtree
gamma	For pruning	1
max_depth	Depth of decision tree	12
scale_pos_weight	Balance positive and negative sample weights	1
subsample, colsample_bytree	Proportion of random collected samples each time	1
min_child_weight	Number of leaf nodes in the decision tree	1000
eta	Learning rate	0.1

Table 6: Tuning results of *max_depth* and *min_child_weight*.

max_depth	*min_child_weight*	Auc
4	1	0.99826
4	10	0.99770
4	100	0.99667
4	1000	0.99535
6	1	0.99830
6	10	0.99782
6	100	0.99726
6	1000	0.99561
8	1	0.99838
8	10	0.99799
8	100	0.99780
8	1000	0.99574
10	1	0.99854
10	10	0.99827
10	100	0.99798
10	1000	0.99576
12	1	0.99852
12	10	0.99818
12	100	0.99806
12	1000	0.99576

Table 7: Tuning results of *gamma*.

gamma	Auc
0	0.99925
1	0.99854
2	0.99820
3	0.99793
4	0.99776
5	0.99771
6	0.99702
7	0.99689
8	0.99656

results of the parameter adjustment are shown in Table 8. The experimental results show that the model performs best when *subsample* = 0.8 and *colsample_bytree* = 0.8.

5.3. Evaluation Metrics. The evaluation metrics of our proposed infected host detection approach using malicious external HTTP traffic are expressed as follows: TP refers to the number of malicious HTTP requests that are recognized as malware HTTP requests, TN indicates that the number of normal HTTP requests that are recognized as normal HTTP requests, FP refers to the number of normal HTTP requests that have been mistaken for malware HTTP requests, and

Table 8: Tuning results of *subsample* and *colsample_bytree*.

colsample_bytree	*subsample*	Auc
0.7	0.7	0.99793
0.7	0.8	0.99799
0.7	0.9	0.99782
0.7	1	0.99774
0.8	0.7	0.99912
0.8	0.8	0.99926
0.8	0.9	0.99891
0.8	1	0.99857
0.9	0.7	0.99802
0.9	0.8	0.99840
0.9	0.9	0.99846
0.9	1	0.99793
1	0.7	0.99789
1	0.8	0.99821
1	0.9	0.99844
1	1	0.99817

FN indicates that the number of normal HTTP requests that are incorrectly identified as malware HTTP requests. The higher the value of precision, recall, and $F1$, the better the recognition effect of the infected host detection approach.

(1) $\mathrm{ACC} = (\mathrm{TP} + \mathrm{TN})/(\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN})$

(2) ROC curve whose horizontal axis is FRP and vertical axis is TRP, where $\mathrm{FPR} = \mathrm{FP}/(\mathrm{TN} + \mathrm{FP})$ and $\mathrm{TPR} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$

(3) PRC curve whose vertical axis is precision and horizontal axis is recall, where precision $(P) = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$ and recall $(R) = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$

(4) $F1 = (2 * P * R)/(P + R)$

5.4. Experimental Results. When the ratio of the number of HTTP requests in the training set and testing set is 7 : 3, the experimental results are shown in Table 9.

The accuracy of the testing set is 98.72%, and the false positive rate is less than 1%. The total testing time is about 7 s. Therefore, the proposed approach can quickly detect the network traffic and conclude whether the host is infected by malware so that the user can respond to the action as soon as possible. The PRC curve matching the threshold is shown in Figure 4. It can be seen that the algorithm has maintained a high precision with the increase of the recall rate. Finally, 0.8 is selected as the matching threshold. At this time, the accuracy of the algorithm is 93.56%, the recall rate is 97.14%, and the F-value is 0.9532.

Table 9: The experimental results when the ratio of the number of HTTP requests in the training set and testing set is 7 : 3.

Best iteration	218
Train-auc	99.8944%
Cross-validation-auc	99.8599%
Test-auc	98.726487%
Cost time	7.28223S

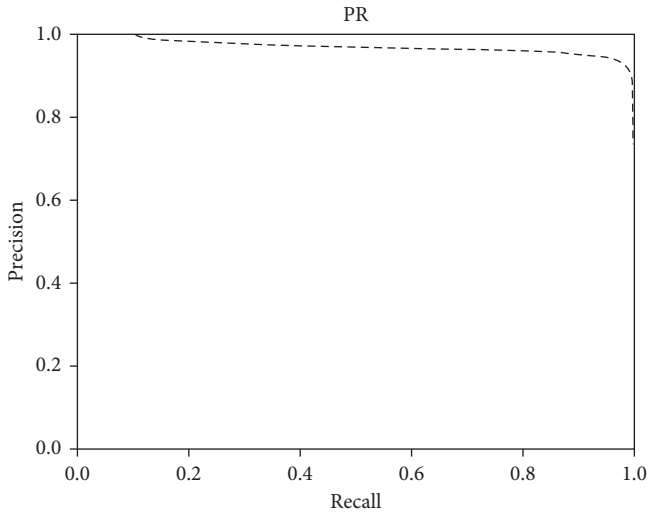


Figure 4: PRC curve of the detection approach.

To better validate our proposed approach, we also compare our approach to the other two methods of Ogawa et al. [12] and Li et al. [22]. We reproduced these two comparison experiments using our own data set. The experimental results are shown in Table 10.

Table 10 shows that the ACC, P, R, and $F1$ of our proposed approach are the largest, and they are 0.9827, 0.9356, 0.9714, and 0.9532, respectively. Therefore, our proposed approach using XGBoost and HTTP header statistical template is better to detect HTTP malware traffic than the method that uses HTTP header combined machine learning. The main reason is that Ogawa et al.'s approach and Li et al.'s approach are either based on a single field or based on all fields, their feature validity is low. Our proposed approach uses statistical techniques to aggregate similar features of the malicious HTTP header fields. Thus, our approach can more effectively characterize malware traffic characteristics, which can further improve the accuracy of malware HTTP traffic recognition.

In addition, we select 10%, 20%, 30%, . . . , 90% of the samples as the training set and set the matching threshold to 0.8 to test other sample data. The correct rate and false positive rate of malicious traffic and normal traffic are separately measured, whose results are shown in Figure 5. It can be seen that the detection rate of the normal HTTP requests has been maintained above 99%. For malicious samples, the detection accuracy rate is based on the diversity of the model. In the case that the training set is only 10% and the model data is insufficient, the algorithm can still detect 77.65% of malicious traffic, indicating that the algorithm has better generalization ability for malicious traffic variants.

Table 10: The experimental result of comparative testing.

Approach	ACC (%)	P (%)	R (%)	$F1$ (%)
Our approach	98.72	93.56	97.14	95.32
Ogawa et al.'s approach	96.25	92.84	94.99	93.9
Li et al.'s approach	97.17	93.25	96.19	94.7

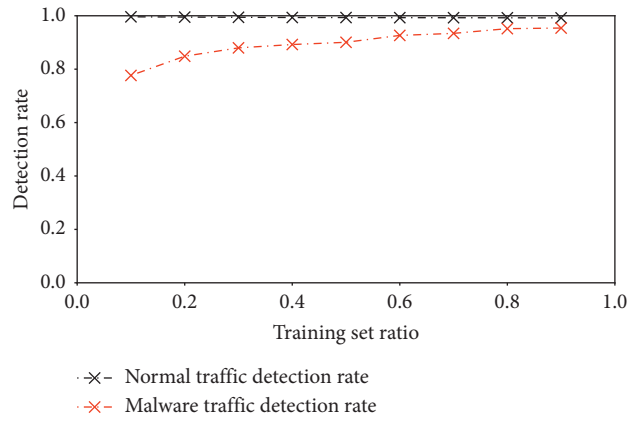


Figure 5: The impact of different ratios between the training set and the testing set.

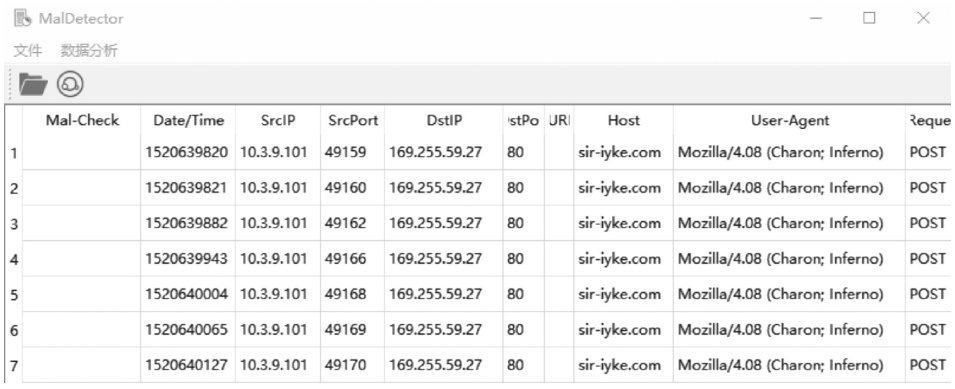
Table 11: The experimental result under different malware traffic ratios.

Malicious traffic ratio in dataset (%)	ACC (%)	P (%)	R (%)	$F1$ (%)
40	94.81	94.58	92.90	93.57
30	91.99	87.41	92.11	89.70
20	97.16	95.87	87.03	92.02
10	97.52	94.04	78.14	85.36

We also change the malicious traffic and normal traffic ratio in our training set and testing set. The experimental results are shown in Table 11.

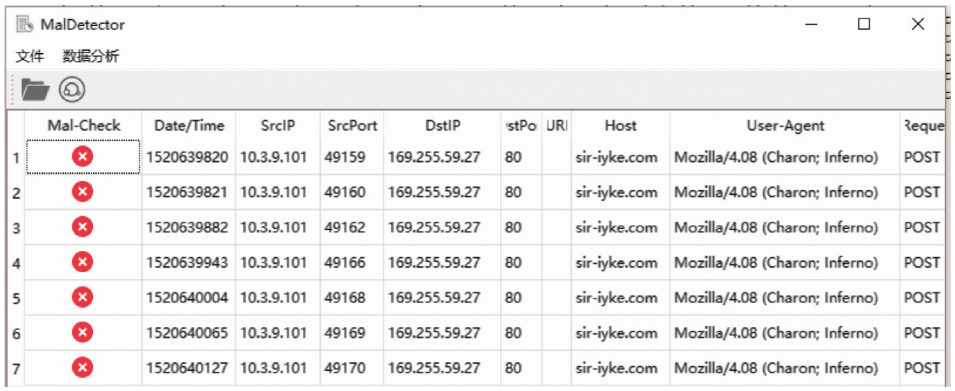
The accuracy rates under different malware traffic ratios all remained above 90%. However, the model has high precision but a low recall rate when malicious traffic accounted for 10% and 20%, respectively. The main reason is that the proportion of malicious traffic is too small, resulting in insufficient training of the model. The results show that if we want to build a machine learning model which can correctly identify malicious traffic, the proportion of malicious traffic and the normal flow ratio needs to be maintained at a relative balance. Malicious traffic accounts for less than 1% of the data in real-world samples. Thus, it is necessary to further process the sample, such as subsampling or oversampling, to increase the proportion of malicious traffic, thereby improving detection accuracy.

5.5. MalDetector System Testing. We also use the malicious traffic samples that do not exist in the training data and testing data to verify if the system has the ability to detect new malware and its variants. The selected malicious traffic



	Mal-Check	Date/Time	SrcIP	SrcPort	DstIP	stPo	URI	Host	User-Agent	Reque
1		1520639820	10.3.9.101	49159	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
2		1520639821	10.3.9.101	49160	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
3		1520639882	10.3.9.101	49162	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
4		1520639943	10.3.9.101	49166	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
5		1520640004	10.3.9.101	49168	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
6		1520640065	10.3.9.101	49169	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
7		1520640127	10.3.9.101	49170	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST

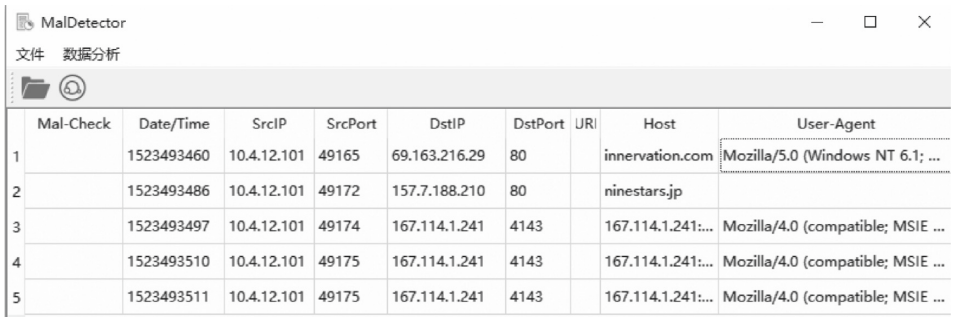
(a)



	Mal-Check	Date/Time	SrcIP	SrcPort	DstIP	stPo	URI	Host	User-Agent	Reque
1	✕	1520639820	10.3.9.101	49159	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
2	✕	1520639821	10.3.9.101	49160	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
3	✕	1520639882	10.3.9.101	49162	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
4	✕	1520639943	10.3.9.101	49166	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
5	✕	1520640004	10.3.9.101	49168	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
6	✕	1520640065	10.3.9.101	49169	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST
7	✕	1520640127	10.3.9.101	49170	169.255.59.27	80		sir-iyke.com	Mozilla/4.08 (Charon; Inferno)	POST

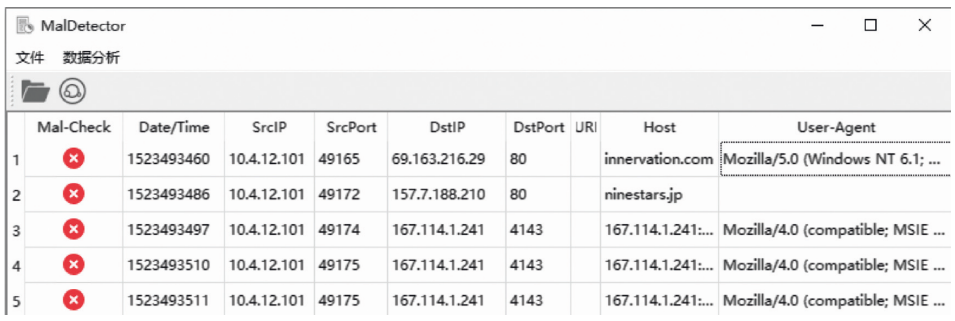
(b)

Figure 6: Loki-Bot traffic and the detection result of MalDetector.



	Mal-Check	Date/Time	SrcIP	SrcPort	DstIP	DstPort	URI	Host	User-Agent
1		1523493460	10.4.12.101	49165	69.163.216.29	80		innervation.com	Mozilla/5.0 (Windows NT 6.1; ...
2		1523493486	10.4.12.101	49172	157.7.188.210	80		ninestars.jp	
3		1523493497	10.4.12.101	49174	167.114.1.241	4143		167.114.1.241:...	Mozilla/4.0 (compatible; MSIE ...
4		1523493510	10.4.12.101	49175	167.114.1.241	4143		167.114.1.241:...	Mozilla/4.0 (compatible; MSIE ...
5		1523493511	10.4.12.101	49175	167.114.1.241	4143		167.114.1.241:...	Mozilla/4.0 (compatible; MSIE ...

(a)



	Mal-Check	Date/Time	SrcIP	SrcPort	DstIP	DstPort	URI	Host	User-Agent
1	✕	1523493460	10.4.12.101	49165	69.163.216.29	80		innervation.com	Mozilla/5.0 (Windows NT 6.1; ...
2	✕	1523493486	10.4.12.101	49172	157.7.188.210	80		ninestars.jp	
3	✕	1523493497	10.4.12.101	49174	167.114.1.241	4143		167.114.1.241:...	Mozilla/4.0 (compatible; MSIE ...
4	✕	1523493510	10.4.12.101	49175	167.114.1.241	4143		167.114.1.241:...	Mozilla/4.0 (compatible; MSIE ...
5	✕	1523493511	10.4.12.101	49175	167.114.1.241	4143		167.114.1.241:...	Mozilla/4.0 (compatible; MSIE ...

(b)

Figure 7: Emotet traffic and the detection result of MalDetector.

samples have the same source as the training data, both of which are MALWARE-TRAFFIC-ANALYSIS.NET.

5.5.1. Loki-Bot. Loki-Bot [32] uses a malicious website to push fake "Adobe Flash Player," "APK Installer," "System Update," "Adblock," "Security Certificate," and other application updates to induce user installation. The Loki-Bot malware is a bank hijacking Trojan, a variant of the BankBot Trojan. The traffic sample of running Loki-Bot and the testing result using MalDetector are shown in Figure 6. The experimental results show that MalDetector detects all the malicious HTTP traffic of Loki-Bot.

5.5.2. Emotet. Emotet [33] is a new type of banking Trojan in Germany. The sample flow is a new variant of Emotet that appeared in September 2017. It has its own ability to evade safety detection and cannot be recognized by antivirus software. The traffic sample of running Emotet and the testing result using MalDetector are shown in Figure 7. The experimental results show that MalDetector detects all the malicious HTTP traffic of Emotet.

6. Conclusion

The diversification of malware and the complication of its technologies have brought new challenges to cybersecurity. Unfortunately, rule-based traditional malware traffic detection methods are unable to detect malware variants. Machine learning-based methods can make up for this defect, and most malware uses the HTTP protocol to send malicious external traffic to the C&C server. Thus, we propose an approach to detect infected hosts using HTTP traffic combined with a machine learning algorithm. We mainly extract the common templates for the HTTP traffic header, so it still works for the traffic generated by the confusing malware. We also use the most popular XGBoost algorithm to detect infected hosts, which has the advantages of high efficiency and high accuracy. The experimental results show that the accuracy of the method reaches 98.72% and the false positive rate is less than 1%, where the experimental data is from MALWARE-TRAFFIC-ANALYSIS.NET and UNSW-NB 15. We also used two real samples that are Loki-Bot and Emotet to verify the effectiveness of the MalDetector system. We plan to combine the approach with malware dynamic analysis to further improve its detection accuracy in the future. Furthermore, some malware utilizes HTTPS to hide its content from the analyzer so that it further reduces detection possibility. Because the header information of HTTPS traffic has been encrypted, our method cannot be applied. We will consider new fields and combine with DNS traffic to refine the templates to detect anomaly-based malware infection in the future.

Data Availability

The experimental data were collected and synthesized by ourselves. It has not been published online yet.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Key Research and Development Project (Grant no. 2016QY04W0800), the National Defense Innovation Special Zone Program of Science and Technology (Grant no. JG2019055), and the National Natural Science Foundation of China (Grant nos. 61902262 and 61572115).

References

[1] D. Zhao, I. Traore, B. Sayed et al., "Botnet detection based on traffic behavior analysis and flow intervals," *Computers & Security*, vol. 39, pp. 2–16, 2013.

[2] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[3] Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," *Security and Communication Networks*, vol. 2017, Article ID 4184196, 10 pages, 2017.

[4] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting APT malware infections based on malicious DNS and traffic analysis," *IEEE Access*, vol. 3, pp. 1132–1142, 2015.

[5] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-Centric Computing and Information Sciences*, vol. 8, no. 1, p. 3, 2018.

[6] 360 Internet Security Center, China Internet Security Report for the Third Quarter of 2017, 2017.

[7] McAfee Labs, McAfee Labs Threats Report: December 2017, McAfee, Santa Clara, CA, USA, 2017.

[8] X. Hu, J. Jang, M. P. Stoecklin et al., "BAYWATCH: robust beaconing detection to identify infected hosts in large-scale enterprise networks," in *Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 479–490, Toulouse, France, June 2016.

[9] Malware-Traffic-Analysis.net, http://malware-traffic-analysis.net/.

[10] The UNSW-NB15 Data Set, https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/.

[11] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*, vol. 10, p. 14, San Jose, CA, USA, April 2010.

[12] H. Ogawa, Y. Yamaguchi, H. Shimada et al., "Malware originated http traffic detection utilizing cluster appearance ratio," in *Proceedings of the 2017 International Conference on Information Networking (ICOIN)*, pp. 248–253, IEEE, Da Nang, Vietnam, January 2017.

[13] M. Piskozub, R. Spolaor, and I. Martinovic, "MalAlert: detecting malware in large-scale network traffic using statistical features," *ACM Sigmetrics Performance Evaluation Review*, vol. 46, no. 3, pp. 151–154, 2019.

[14] N. Moustafa, B. Turnbull, and K. K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4815–4830, 2019.

[15] A. Liu, Z. Chen, S. Wang, L. Peng, C. Zhao, and Y. Shi, "A fast and effective detection of mobile malware behavior using network traffic," in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, pp. 109–120, Springer, Guangzhou, China, November 2018.

[16] M. Yeo, Y. Koo, Y. Yoon et al., "Flow-based malware detection using convolutional neural network," in *Proceedings of the 2018 International Conference on Information Networking (ICOIN)*, pp. 910–913, IEEE, Chiang Mai, Thailand, January 2018.

[17] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '14)*, pp. 39–50, Kyoto, Japan, June 2014.

[18] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Causality-based sensemaking of network traffic for android application security," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security (ALSec '16)*, pp. 47–58, Madrid, Spain, April 2016.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[20] Y. Zhang, H. Mekky, Z.-L. Zhang et al., "Detecting malicious activities with user-agent-based profiles," *International Journal of Network Management*, vol. 25, no. 5, pp. 306–319, 2015.

[21] M. Grill and M. Rehak, "Malware detection using http user-agent discrepancy identification," in *Proceedings of the IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 221–226, Atlanta, Georgia, December 2014.

[22] K. Li, R. Chen, L. Gu et al., "A method based on statistical characteristics for detection malware requests in network traffic," in *Proceedings of the 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 527–532, Guangzhou, China, June 2018.

[23] J. Zhang, S. Saha, G. Gu et al., "Systematic mining of associated server herds for malware campaign discovery," in *Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems*, pp. 630–641, Columbus, OH, USA, June 2015.

[24] H. Mekky, R. Torres, Z. L. Zhang et al., "Detecting malicious http redirections using trees of user browsing activity," in *Proceedings of the IEEE Conference on Computer Communications (IEEE INFOCOM 2014)*, pp. 1159–1167, Toronto, Canada, April 2014.

[25] L. Liu, S. Saha, R. Torres et al., "Detecting malicious clients in isp networks using http connectivity graph and flow information," in *Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pp. 150–157, Beijing, China, August 2014.

[26] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics," *Computers & Electrical Engineering*, vol. 66, pp. 353–368, 2018.

[27] S. Mizuno, M. Hatada, T. Mori, and S. Goto, "Botdetector: a robust and scalable approach toward detecting malware-infected devices," in *Proceedings of the 2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, Paris, France, May 2017.

[28] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, New York, NY, USA, August 2016.

[29] K. M. Kumar and A. R. M. Reddy, "A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method," *Pattern Recognition*, vol. 58, pp. 39–48, 2016.

[30] A. Malhotra and K. Bajaj, "A hybrid pattern based text mining approach for malware detection using DBScan," *CSI Transactions on ICT*, vol. 4, no. 2–4, pp. 141–149, 2016.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "SCIKIT-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] D. Rendell, "Understanding the evolution of malware," *Computer Fraud & Security*, vol. 2019, no. 1, pp. 17–19, 2019.

[33] H. Huang, H. Deng, J. Chen, L. Han, and W Wang, "Automatic multi-task learning system for abnormal network traffic detection," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 13, no. 4, 2018.