Counterfeit Fingerprint Detection of Outbound HTTP Traffic with Graph Edit Distance

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—We present DECANTeR, a system to detect anomalous outbound HTTP communication, which passively extracts fingerprints for each application running on a monitored host. The goal of our system is to detect unknown malware and backdoor communication indicated by unknown fingerprints extracted from a hosts network traffic. We evaluate a prototype with realistic data from an international organization and datasets composed of malicious traffic. We show that our system achieves a false positive rate of 0.9% for 441 monitored host machines, an average detection rate of 97.7%, and that it cannot be evaded by malware using simple evasion techniques such as using known browser user agent values. We compare our solution with DUMONT [24], the current state-of-the-art IDS which detects HTTP covert communication channels by focusing on benign HTTP traffic. The results show that DECANTeR outperforms DUMONT in terms of detection rate, false positive rate, and even evasion-resistance. Finally, DECANTeR detects 96.8% of information stealers in our dataset, which shows its potential to detect data exfiltration.**

***Index Terms*—Anomaly Detection, Data Exfiltration, Data Leakage, Application Fingerprinting, Network Security**

I. Introduction

Nowadays, malware usually uses HTTP protocol to connect suspicious host for data leakage and exfiltration, because it's a common network channel that Intrusion Detection/Prevention Systems (IDS/IPS) never block the HTTP traffics. Therefore, malware tries to hide their penetrations in the HTTP traffic to evade the detections in Figure 1. In the previous research, there are many botnet using HTTP protocl to communicate with the C&C server for waiting command instead of IRC channel [1]. However, the proposed method in the past that uses fingerprint to detect malware hide in outbound HTTP traffics [2], and which can't efficiently detect malware when hacker generates counterfeit fingerprints.

The main idea concept of fingerprint is around the HTTP headers. But, as we know, hacker can use exploit tool or library to eaily modify the contents of a HTTP header. Previous research also indicates that malware uses modified HTTP header to evade the latest detections system [3], and which points out most malware using browser-like user-agent since browser's connection behavior is various and complex. Therefore, we represent the problem define as following:

- **Problem Definition**

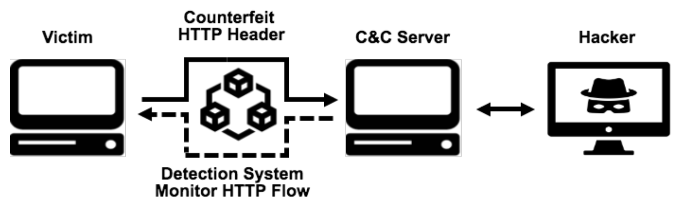
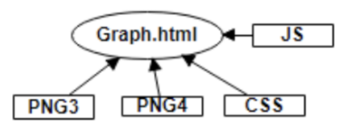
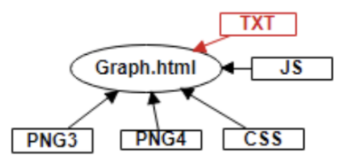


Fig. 1: A process of attacking scenario



(a) Normal Referrer Correlation Graph



(b) Malicious Referrer Correlation Graph

Fig. 2: Difference between normal and malicious referrer correlation graph.

To evade intrusion detections, malware could make counterfeit fingerprint (e.g., user-agent, accept language, and so on) in a HTTP header. But, referrer correlation (e.g., domain and referrer fields of HTTP header) usually is fixed when browser connects common domain, therefore, correlation would be changed when malware connects C&C server even using fake HTTP header. The detail is shown as Figure 2.

To resolve this problem, we propose an approach based on deviation estimating when given referrer correlation graphs in training and testing phases, and the contributions of this work are briefly summarized as followings:

- **Contribution 1**
  Description here...
- **Contribution 2**

Description here...

- **Contribution 3**

  Description here...

In the remaining parts of this report, Section 2 surveys related work, and Section 3 describes the detail components of the proposed approach. The effectiveness, performance, and case studies of the proposed framework are evaluated and discussed in Section 4. At last, Section 5 concludes this project.

II. Related Work

III. Proposed Approach

This section gives the details about our proposed method which aims at detecting counterfeit fingerprints from applications' outbound HTTP traffics. Before going further, all PCAP files collected by an enterprise's host is network activities generated by a set of applications such as browsers $B = \{b_1, \ \dots \ , b_n\}$, and which are all installed in hosts. Each browser b_i has several PCAP files which contain specific network characteristics, and our proposed approach possibly create a fingerprint f_{b_i} for each browser. The PCAP files of a host H include union of all browser fingerprints which is defined as $H = \cup_j^n f_{b_i}$. The proposed counterfeit fingerprint detection process consists of training and testing phases. In training phase, we assume enterprise hosts aren't compromised. This method mainly arises from the first one that is a data-driven and unsupervised flow responsible for a browser's fingerprint [2] and referrer correlation construction. This step takes the fields of a PCAP file as input and classifies browser traffics, and then construct fingerprints and referrer correlation graphs. In the testing phase, given a browser outbound HTTP traffic reconstructed by fingerprint and referrer correlation graph, and the second step filters benign browser traffics through fingerprint matching. Continuously, compare its and trained referrer correlation graph using Graph Edit Distance (GED) for counterfeit fingerprint detection. The proposed method is depicted in figure 3 and following paragraphs describe the details of each component.

A. Browser Traffic Extractor

For most cases of client-side attacking, hackers whose general goal is to steal valuable data before malware connects to C&C server. As a result, PCAP files, that contain specific network characteristics of an application (e.g., browser) for each host in the enterprise.

To generate fingerprint for each browser, our approach first extracts various entities from PCAP files. Table I shows 4 heterogeneous fields which can be extracted from each one-line log, including domain (*Domain*), user-agent (*User-agent*), accept language (*Accept-Lang*), and referrer (*Referrer*). The reason for choosing these 4 fields for browser traffic classification can be summarized as followings and fingerprint construction is represented in next subsection.

In previous research [2], Bortolameotti et al. identified two types of HTTP applications (e.g., $browser$ and $background$). This subsection aims to filter logs of a PCAP file according to the *User-agent*, because we focus on counterfeit fingerprints of browser network activities. To identify browser activities, the browser flags we defined are ”Mozilla”, ”Opera”, ”MQQBrowser”, ”UCWEB”, ”NOKIA5700”, ”Openwave”, ”Safari”, and ”Chrome”, and which are used for string matching in field *User-agent*. Furthermore, in the testing phase, an implementation time-slot t is a fixed time window of T minutes, and the filtered logs is passed to the next module after t ends.

TABLE I: Fields and Values of Database in a PCAP File

Field	Value for Instance
Domain	www.yongchang-yc.com.tw
User-agent	Mozilla/5.0 (Windows NT 6.1; Win64; x64) ...
Accept-Lang	zh-TW,zh;q=0.9,en-US;q=0.8,en;q=0.7
Referrer	www.yongchang-yc.com.tw

B. Fingerprint Constructor

Single feature (e.g., *User-agent*) isn't effective enough to filter normal network activities [2] [4]. Therefore, we consider multiple features such as *User-agent* and *Accept-Lang* for fingerprint generation, and *Domain* and *Referrer* would be used for constructing the correlation graph in other subsection. In our assumption, hacker can't be so lucky to guess all parameters of *User-agent* and *Accept-Lang* at the same time. In this subsection, we denote a set of *User-agent* $U = \{u_1, \ \dots \ , u_n\}$, and a set of *Accept-Lang* $L = \{l_1, \ \dots \ , l_m\}$ where $|U| = n$ and $|L| = m$. Furthermore, our approach makes fingerprint $f = (u_i, l_j)$ where $i = 1 \sim n$, $j = 1 \sim m$, and $|f| = n \times m$. Matching testing browser fingerprint to knowns which is trained and stored in database, and we would briefly show the similarity estimation in following module.

C. Fingerprint Matching Module

Matching fingerprint we used is an easy comparison in this module [2]. In training phase, we take fingerprints f_{b_i} for each browser b_i. If our approach constantly runs in testing mode, we must obtain other browser b_j fingerprints f_{b_j}. Then, we use edit distance to estimate fingerprint matching result $d(f_{b_i}, f_{b_j})$ which is shown as in Equation 1.

$$d(f_{b_i}, f_{b_j}) = \sum_k \left| f_{b_{i_k}} - f_{b_{j_k}} \right| \quad (1)$$

D. Referrer Correlation Graph Constructor

A call graph models a connection between URLs as a directed graph whose vertices, representing the domain name is interconnected through directed edges which have reference correlation. According to fields *Domain* and *Referrer*, a vertex could be represented as domain name which is extracted from a URL of the field, and an directed edge shows the reference correlation from *Referrer* to *Domain*. The example directed graph is depicted in figure 4. According to [5], call graphs are formally defined as a directed graph G with vertex $V = V(G)$, representing the domain name, and edge $E = E(G)$, where

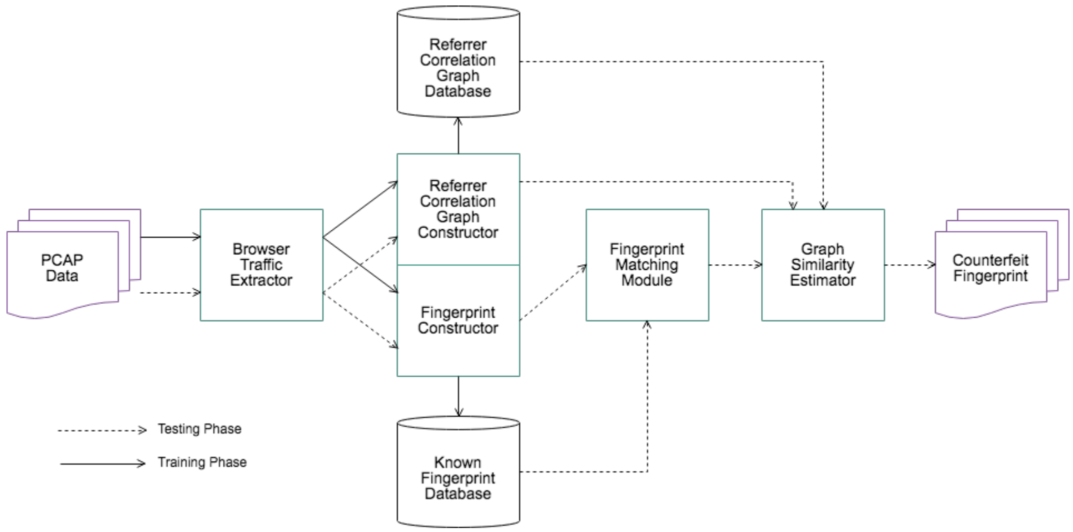


Fig. 3: An overview of our counterfeit fingerprint detection system. Five subsystems are depicted: (1) data preprocessor subsystem, (2) fingerprint constructor subsystem, (3) fingerprint matching subsystem, (4) referrer correlation graph constructor subsystem, and (5) graph similarity estimator subsystem. The system only takes the PCAP files of outbound HTTP traffics as input. In training phase, subsystem (1) and (2) passively extract the benign fingerprint from an application's outbound HTTP traffic, and subsystem (3) could use fingerprints to classify benign traffic in the testing phase. We note that referrer correlation extraction in the subsystem (4) is a key step, in the sense that if it can extract discriminative features for counterfeit fingerprint detection, the detection in the subsystem (5) is relatively straightforward.

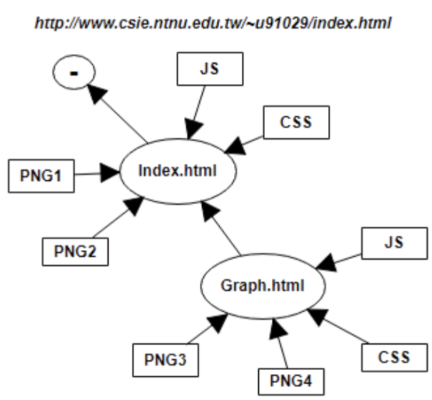


Fig. 4: An Example of a referrer correlation graph generated from a browser traffic.

$E(G) \subseteq V(G) \times V(G)$, in correspondence with the reference correlation.

A candidate set $S = \{st_1, st_2, ..., st_{ns}\}$ that contains all domain names filtered by fingerprint matching module and derived from D_i. Note that ns is the total number of derived domain names in D_i. Given the dataset D_i containing i^{th} domain name; $i = 1, ..., ns$, and its referrer correlation based on the dataset should includes an $1 \times ns$ adjacency vector (ADJ), as following:

$$ADJ(i) = \left[\begin{array}{ccccc} tp_{i,1} & \ldots & tp_{i,j} & \ldots & tp_{i,ns} \end{array} \right]$$

where for each i and j, $tp_{i,j}$ represents a directed edge which is referrer correlation from j^{th} domain name to i^{th} candidate domain.

$$\forall\ i, j = 1, ..., ns,$$
$$tp_{i,j} = \text{\#connections from } st_j \text{ to } st_i \text{ in } D_i$$

E. Graph Similarity Estimator

Our proposed approach relies on appropriate estimating deviation from domain name's new referrer correlation to its benign one. In this paper, domain name's referrer correlation is summarized as patterns represented by adjacency vector, as a result, deviation measuring can be realized by using graph edit distance (GED) to quantify similarity (or dissimilarity) between different vectors. The formal graph edit distance between two graphs G_1 and G_2, written as $GED(G_1, G_2)$ can be defined as following:

$$GED(G_1, G_2) = \min_{(e_1,...,e_k) \in P(G_1,G_2)} \sum_{i=1}^{k} cost(e_i), \quad (2)$$

where $P(G_1, G_2)$ denotes the universal set of editing paths isomorphically transforming G_1 into G_2, and $cost(e_i)$ is the cost of each graph editing operation, e_i.

With respect to referrer correlation graph in our method, calculation of GED on two graphs can then be implemented by following equation (3):

$$GED(ADJ(a), ADJ(b)) = \sum_{j=1}^{ns} \left| tp_{a,j} - tp_{b,j} \right|, \quad (3)$$

where $ADJ(a)$ and $ADJ(b)$ are adjacency vectors of two referrer correlation graphs, as well as $tp_{a,j}$ and $tp_{b,j}$ are the corresponding references in $ADJ(a)$ and $ADJ(b)$, respectively. The ns is the number of candidate domain names after fingerprint matching.

IV. Experiment Results

In this section, we would describe the datasets that we used to perform our experiments. For starting our experiments we have used two different datasets, simulated and real-world data. The simulated data is enable to evaluate the detection performance of our system and compare with DECANTeR [2].

A. Experimental Settings

In the following, we briefly present the datasets we used for evaluation in our system. The dataset information is represented in table II.

- **Real-world Data**
  The outbound HTTP traffics we collect from more than hundreds of machines in a technology industry. Users of these machines include accountants, engineers, sales executive, and administrative personnel. Since the users vary from different occupations that lets data become various and complexity. The real world dataset has split into two sets, one is training and the other is testing. Training set has covered first few days and testing set is the traffics of last days. In training set, it contains 1,690,869 HTTP requests. and the testing set contains 68,234 HTTP requests in this real-world dataset.
- **Simulated Data**
  In this paper, our goal is a detection of counterfeit fingerprint which would pretend to be browser activities in outbound HTTP traffics. However, this kind of attack is secret and hidden penetration, and hard to collect in the real world. Therefore, we build a botnet malware and monitor its outbound HTTP traffics. As we know, early botnets generally used Internet Relay Chat (IRC) channel to communicate C&C server. In recent years, botnets also start to communicate C&C server through HTTP protocol. Furthermore, we need to build a botnet with the spoofing headers which can evade other detection systems. That is why we collect three different simulated botnet traffics. In dataset_01, it has no spoofing headers from infected host's outbound HTTP traffics. Dataset_02 consists of the botnet traffics with simple spoofing, and which means our botnet sending the requests to web-like user agent through HTTP protocol. Finally, dataset_03 is totally modifying the headers information, and the malware investigates which browser is used by the user or host, and fills the field User-Agent with that specific browser. Moreover, it also fills up some common requests header fields, like Accept, Accept-Encoding, Accept-Language, and Referrer.

B. Evaluation Metrics

Essentially, Our system is a flow filter aim to identify the suspicious requests with the headers field. Four well-known metrics for evaluating the effectiveness of proposed method are adopted as followings: "true positive"(TP) means the number of normal requests which belong to normal traffic. "False negative"(FN) is the number of normal traffic and its results are wrongly predicted. Similarly, "true negative"(TN) means the number of abnormal traffic and the system predict it as malicious requests, while "false positive"(FP) is the number of abnormal traffic that the system predicts it as normal traffic. Based on the accumulation of TP, FN, TN, and FP, one extended metrics ($accuracy$) popularly used in machine learning problems are also adopted here to evaluate proposed method and listed in equations below. Note that the optimal $accuracy$ of 1.0 means all of the malware are successfully picked out by the proposed approach.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

C. Effectiveness Analysis

Just as we know, malware can easily modify the HTTP headers. Hence, we build three similar malware to evaluate our approach. With these botnets, they all have the same purpose. Moreover, they would steal some sensitive information (such as OS information, system account, and the password) and send requests to the C&C server periodically waiting for commands to execute. The difference between them is the degree of the spoofing HTTP headers. The botnet in dataset_01 does not spoof any HTTP headers. We fill up with the empty to the User-Agent field. The botnet in datset_02 only simply sets the User-agent as a common browser which calls Internet Explore (IE). In the dataset_03 , botnet would specifically detect the victim's browser version, system language and then fill them into the HTTP header fields. In addition, for the reference field in HTTP headers we default to point to the google website. The result show in table III, we can see both detection system has the great performance with dataset_01 and dataset_02. However, we can notice that our system has a better performance for botnets that based on advanced spoofing methods.

D. Limitation and Future Work

V. Conclusion

Acknowledgment

References

[1] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[2] R. Bortolameotti, T. van Ede, M. Caselli, M. H. Everts, P. Hartel, R. Hofstede, W. Jonker, and A. Peter, "Decanter: Detection of anomalous outbound http traffic by passive application fingerprinting," in *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 2017, pp. 373–386.

[3] M. Grill and M. Rehák, "Malware detection using http user-agent discrepancy identification," in *Information Forensics and Security (WIFS), 2014 IEEE International Workshop on*. IEEE, 2014, pp. 221–226.

TABLE II: Overview of the Datasets

Dataset	Features	Type	All Samples	Malicious Samples
Industy_flow01	Packets	Malicious Flows	1690869	N/A
Industy_flow02	Packets	Malicious Flows	68234	N/A
Dataset_01	Packets	Botnet	220	220
Dataset_02	Packets	Botnet	216	216
Dataset_03	Packets	Botnet	1045	168

TABLE III: Simulated Data

Dataset	System	HTTP Requests	Evaluation Metrics				Accuracy
			TP	TN	FP	FN	
Dataset_01	DECANTeR [2]	220					
	Our System						
Dataset_02	DECANTeR [2]	216					
	Our System						
Dataset_03	DECANTeR [2]	1045	869	0	168	8	0.8315
	Our System		869	168	0	8	0.9923

[4] N. Kheir, "Analyzing http user agent anomalies for malware detection," in *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 2013, pp. 187–200.

[5] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in computer virology*, vol. 7, no. 4, pp. 233–245, 2011.