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 many malwares use HTTP protocol to connect to suspicious host or send the sensitive data because it's a commonly open channel in the networks. Therefore malwares try to hide in the normal traffic to evade the detections.

As in Analyzing HTTP User Agent Anomalies for Malware Detection [1], they claim with up to 60% of malware showing at least one outgoing HTTP connection. There are differen research indicate there are many botnet using HTTP protocl to communicate with the C&C server for waiting command instead of IRC channel [2].

Our system use fingerprint to detect outbound http traffic. The idea come from DECANTeR.

But the main concept is around the HTTP headers. But as we know, we can use some tool or library to eaily modify the HTTP headers. Some research indicate some malwares also use modified HTTP header to evade the latest detections system. In the [Malware Detection Using HTTP User-Agent Discrepancy Identification] point out there are 40% malwares using browser-like user-agent since browser's connection behaviors are more complex than application connections.

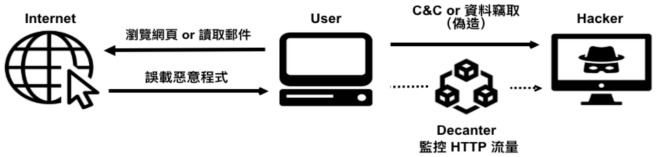
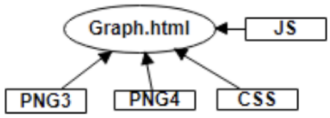
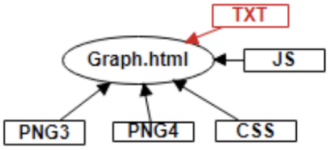


Fig. 1: A process of attacking scenario



(a) Normal Referrer Correlation Graph



(b) Malicious Referrer Correlation Graph

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

Thence we represent a detection system to captical these Counterfeit abnormal traffic. We evaluate our system's validity through simulated dataset and real world dataset.

Contributions of this work are briefly summarized as followings.

- **Contribution 1**
  Description here...
- **Contribution 2**
  Description here...
- **Contribution 3**
  Description here...

II. Related Work

III. Proposed Approach

This section gives the details about our proposed method which aims at detecting counterfeit fingerprints from applica-

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

tions' 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 $B = \{b_1, \ \ldots \ , b_n\}$, 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 [3] 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 [3], 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.

B. Fingerprint Constructor

Single feature (e.g., *User-agent*) isn't effective enough to filter normal network activities [3] [1]. 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* $U = \{u_1, \ \ldots \ , u_n\}$, and a set of *Accept-Lang* $L = \{l_1, \ \ldots \ , l_m\}$ 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 [3]. 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.

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

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 [4], 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 $E(G) \subseteq V(G) \times V(G)$, 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:

$$ADJ(i) = \left[\ tp_{i,1} \quad \ldots \quad tp_{i,j} \quad \ldots \quad tp_{i,ns} \ \right]$$

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.

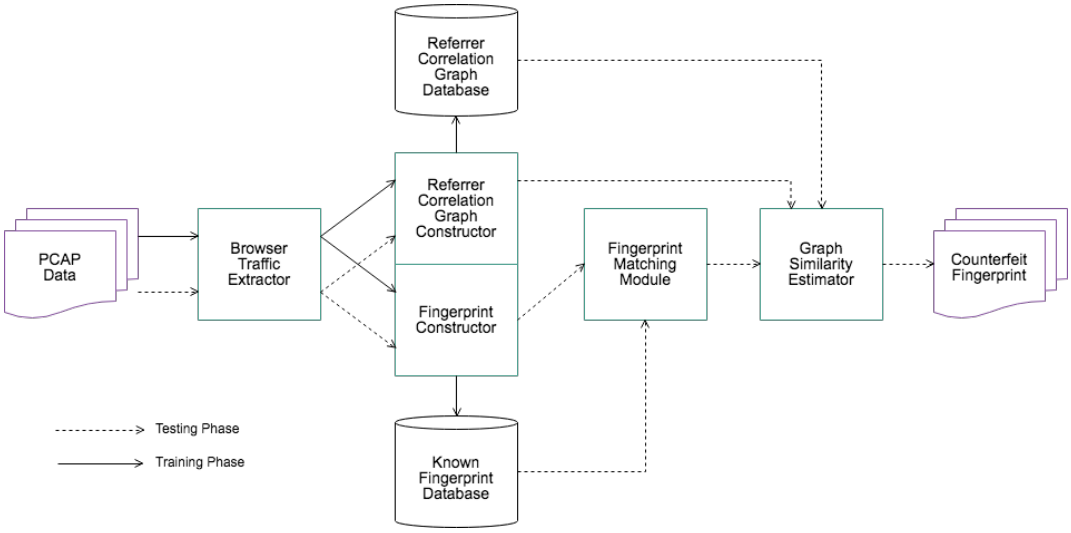


Fig. 3: An overview of our counterfeit fingerprint detection system. Five subsystems are depicted: (1) data preprocessor subsystem, (2) fingerprint constructor subsystem, (3) fingerprint matching subsystem, (4) referrer correlation graph constructor subsystem, and (5) graph similarity estimator subsystem. 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.

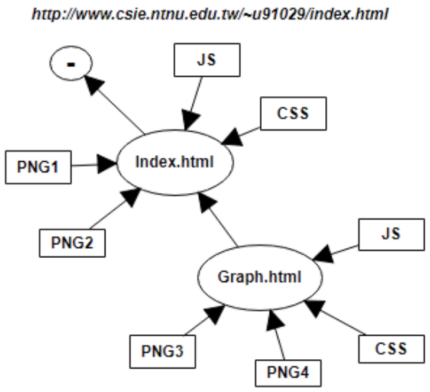


Fig. 4: An Example of a referrer correlation graph generated from a browser traffic.

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

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):

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

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

Tony, pliz write overview here!!!

A. Experimental Settings

Tony, pliz write like the followings in this subsection!

Malware analyzed in following experiments are malicious behavioral sequences from collected malicious executable files in real world. In proposed method implementation, collected

TABLE II: Datasets used in this paper's experiments

Datasets	Operation Type	Malware Families	Malware
Ahmadi et al. [**?**]	API	4	3,829
Ki et al. [**?**]	Opcode	9	10,867

malware were paresed and information of four major fields in table II. Table II also lists the the number of malware families, the size of gathered data, and operation types in malicious behavioral sequences.

B. Evaluation Metrics

Tony, pliz replace malware to fingerprint!!!

Essentially, malware clustering is a multiple classification problem aim to identify malware comes from which families. Four well-known metrics for evaluating effectiveness of proposed method are adopted as followings: "true positive"(TP) means the number of malware which belong to same malware families. "False negative"(FN) is the number of malware which's families are wrongly predicted. Similarly, "true negative"(TN) means the number of malware which aren't same families and being viewed as others, while "false positive"(FP) is the number of false alarms that other families' malware being detected as the same ones. Based on 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 malware are successfully classified by proposed approach.

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

C. Effectiveness Analysis

Tony, pliz show your exp graph and table here!!!

D. Limitation and Future Work

V. Conclusion

Acknowledgment

References

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

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

[3] 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.

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