# 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

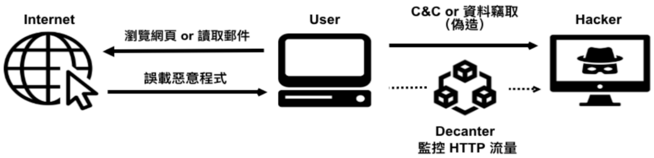
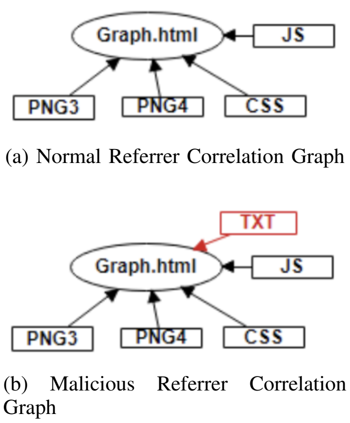


Fig. 1: A process of attacking scenario



(a) Normal Referrer Correlation Graph

(b) Malicious Referrer Correlation Graph

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

***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

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 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 [1] 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.

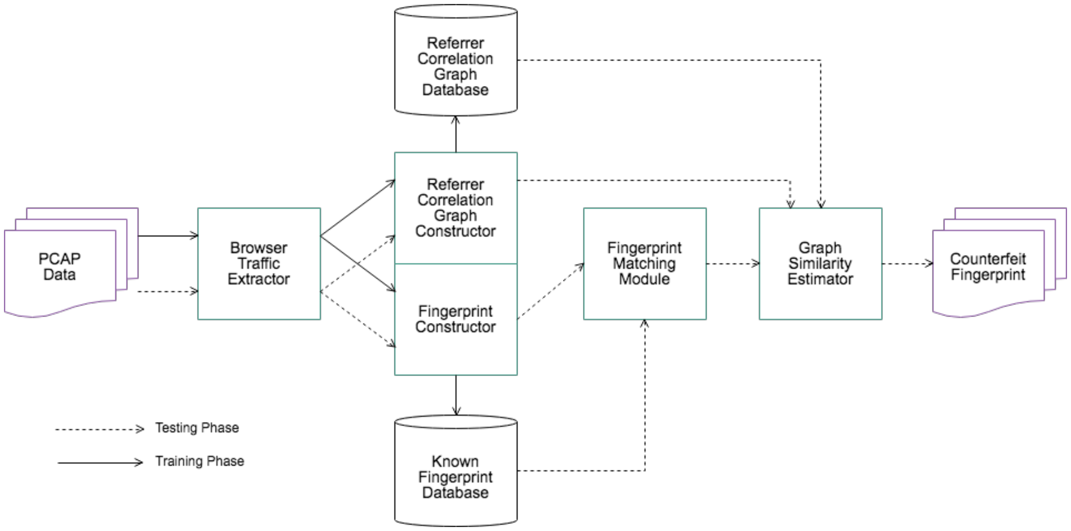

Referrer Correlation Graph Database
Referrer Correlation Graph Constructor
PCAP Data
Browser Traffic Extractor
Fingerprint Constructor
Fingerprint Matching Module
Graph Similarity Estimator
Counterfeit Fingerprint
Known Fingerprint Database
Testing Phase
Training Phase

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.

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 |

### 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 [1], 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 [1] [2]. 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 [1]. 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.

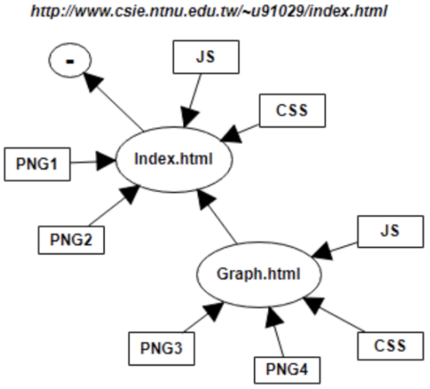


Fig. 4: An Example of a referrer correlation graph generated from a browser traffic.

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

### *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 [3], 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.

### *E. Graph Similarity Estimator*

We denote the graph edit distance between two referrer correlation graphs $g_1$ and $g_2$ as $D(g_1, g_2)$, which is the sum of VertexCost and EdgeCost. In which, VertexCost and EdgeCost indicate the numbers of insertions, deletions and substitutions of vertices and edges, respectively, where determine the minimum amount of distortion for transforming one graph into another graph [4]. Let $g_1$ = $(V_1, E_1)$ and $g_2$ = $(V_2, E_2)$: $g_1$ is the source graph, and $g_2$ is the target graph. The graph edit distance between $g_1$ and $g_2$ is defined as follows:

$$D(g_1, g_2) = \textbf{VertexCost} + \textbf{EdgeCost} \tag{2}$$

$$\sigma(g_1, g_2) = \frac{D(g_1, g_2)}{|V(g_1)| + |V(g_2)| + |E(g_1)| + |E(g_2)|} \tag{3}$$

Finally, the similarity $\sigma(g_1, g_2)$ of two graphs $g_1$ and $g_2$ is obtained from the graph edit distance $D(g_1, g_2)$. The details are described in Equation 3, which is a real value on the interval [0,1], where 0 indicates that graphs $g_1$ and $g_2$ are identical, whereas a value near 1 implies that the pair is highly dissimilar. As mentioned before, finding the minimum graph edit distance, i.e., $min_{D(g_1,g_2)}$, is an NP-hard problem but can be approximated. Riesen et al. and Francesc Serratosa introduced an approximation algorithm with a good trade-off between accuracy and speed [5] [6]. Given two graphs $G_1$ and $G_2$, where $\mid G_1 \mid$ = n and $\mid G_2 \mid$ = m, their algorithms, respectively, use a $(n + m) \times (n + m)$ cost matrix $C_1$ and a $n \times m$ cost matrix $C_2$, which both give the cost of mapping a vertex $v \in V(G_1)$ to a vertex $v \in V(G_2)$. Next, Munkres algorithm [7], also known as the Hungarian algorithm, which runs in polynomial time, is applied to find an exact one-to-one vertex assignment that minimizes the total mapping cost. Each entry in the cost matrix represents the cost of matching vertex $v \in V(G_1)$ to a vertex $u \in V(G_2)$. The cost of matching a pair of nodes $c_{ij}$ could equal the transformed cost as defined for the graph edit distance in the following subsection.

In this subsection, more accurate cost estimation allows the discovery of better graph matchings and hence more accurate edit distances. The cost of matching a pair of vertices, element $c_{ij}$ of the matrix, could equal the transformed cost as defined for the graph edit distance, which consists three components: Host jaccard similarity, In-degree, jaccard similarity and Out-degree jaccard similarity can be observed developing between vertices $v$ and $u$ but can be presented as $\delta\left(v, u\right)$, $\delta^-\left(v, u\right)$ and $\delta^+\left(v, u\right)$, and their details can be defined as follows:

$$\delta\left(v, u\right) = \left| 1 - \frac{Bit_{op}\left(v\right) \cap Bit_{op}\left(u\right)}{Bit_{op}\left(v\right) \cup Bit_{op}\left(u\right)} \right| \tag{4}$$

$$\delta^-\left(v, u\right) = \left| 1 - \frac{Neighbor^-\left(v\right) \cap Neighbor^-\left(u\right)}{Neighbor^-\left(v\right) \cup Neighbor^-\left(u\right)} \right| \tag{5}$$

$$\delta^+\left(v, u\right) = \left| 1 - \frac{Neighbor^+\left(v\right) \cap Neighbor^+\left(u\right)}{Neighbor^+\left(v\right) \cup Neighbor^+\left(u\right)} \right| \tag{6}$$

where $Bit_{op}\left(v\right)$ indicates the extraction of vertex $v$'s opcode components, presented by 218 binary bits defined as the above section. Additionally, $Neighbor^-\left(v\right)$ and $Neighbor^+\left(v\right)$ , respectively, present the Neighbor vertices of vertex $v$, which are defined as follows:

$$Neighbor^-\left(v\right) = \{z \mid e_{z,v} \in E\} \tag{7}$$

$$Neighbor^+\left(v\right) = \{z \mid e_{v,z} \in E\} \tag{8}$$

Clearly, each element $c_{ij}$ of the cost matrix indicates the bijective mapping of the i-th vertex to the j-th vertex that can be defined as follows:

$$c_{ij} = \delta\left(i, j\right) + \delta^-\left(i, j\right) + \delta^+\left(i, j\right) \tag{9}$$

The Munkres algorithm [7] [8] is a known algorithm that solves the bipartite matching problem in polynomial time. After obtaining the conceptual matched matrix, we implement the Munkres algorithm to find the optimal permutation that minimizes the cost as the similarity distance between the

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 |

two Android malicious samples' graphs $g_1$ and $g_2$. In Algorithm **??**, the first three steps determine whether the cost matrix has the specific scenario which means the optimization bipartite match from source sample's vertex to target sample's vertex already exists. Then, we can decide whether the scope of the cost matrix needs to be reduced. Furthermore, we acquire a conceptual match matrix $C'$ from the function $f_{gm}(C)$, which can effectively eliminate the side-effects of each wide variability.

## 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 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.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

### C. Effectiveness Analysis

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

### D. Limitation and Future Work

## V. Conclusion

## Acknowledgment

## References

[1] 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.

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

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

[4] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding up graph edit distance computation through fast bipartite matching," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2011, pp. 102–111.

[5] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision computing*, vol. 27, no. 7, pp. 950–959, 2009.

[6] F. Serratosa, "Fast computation of bipartite graph matching," *Pattern Recognition Letters*, vol. 45, pp. 244–250, 2014.

[7] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[8] H. W. Kuhn, "Variants of the hungarian method for assignment problems," *Naval Research Logistics (NRL)*, vol. 3, no. 4, pp. 253–258, 1956.