Encrypted Malware Traffic Detection via Graph-based Network Analysis

Zhuoqun Fu[1], Mingxuan Liu[1], Yue Qin[2], Jia Zhang[1], Yuan Zou[1,3], Qilei Yin[1], Qi Li[1], Haixin Duan[1,4]
[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
[2] Indiana University Bloomington, Bloomington, USA
[3] GeekSec Security Group, Beijing, China [4] Qi An Xin Group Corp., Beijing, China
[1] {fzq20, liumx18}@mails.tsinghua.edu.cn, {yinqilei, qli01, duanhx}@tsinghua.edu.cn, zhangjia@cernet.edu.cn
[2] qinyue@iu.edu, [3] zouyuan@geeksec.com

ABSTRACT

Malicious activities on the Internet continue to grow in volume and damage, posing a serious risk to society. Malware with remote control capabilities is considered one of the most threatening malicious activities, as it can enable arbitrary types of cyber-attacks. As a countermeasure, many malware detection methods are proposed to identify malicious behaviours based on traffic characteristics. However, the emerging encryption and evasion techniques pose substantial barriers to the full exploitation of network information. This significantly impairs the effectiveness of existing malware detection methods relying on a singular type of characteristics. In this paper, we propose ST-Graph to resolve this issue. In addition to traditional stream attributes, ST-Graph explores spatial and temporal characteristics of network behaviours based on a graph representation learning algorithm and integrates all available information to boost the detection decision. To illustrate the effectiveness of ST-Graph, we evaluate it on two datasets. Experimental results demonstrate that ST-Graph outperforms state-of-the-art malware detection systems and also shows good performance in efficiency, generalizability, and robustness. Specifically, it achieves over 99% precision and recall, and its False Positive Rate is even two orders of magnitude lower than (nearly 0.02 times) that of baseline models. Meanwhile, the deployment of ST-Graph in two real network scenarios for around one year shows an outstanding efficiency with only 160 seconds time cost for 5-minute traffic in 1.7 Gbps bandwidth.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

Traffic-based malware detection, encryption traffic, graph representation

ACM Reference Format:
Zhuoqun Fu, Mingxuan Liu, Yue Qin, Jia Zhang, Yuan Zou, Qilei Yin, Qi Li, Haixin Duan, . 2022. Encrypted Malware Traffic Detection via Graph-based Network Analysis. In *25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2022), October 26–28, 2022, Limassol, Cyprus.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3545948.3545983

1 INTRODUCTION

Malware refers to intrusive software programs developed by cybercriminals with malicious intentions such as stealing data, corrupting computers, bringing down servers, and penetrating networks. The most damaging malware are those with remote control capabilities, which give attackers administrative control of the victim's computer to enable arbitrary types of cyber-attacks. For example, NOPEN used in a massive breach of top-secret data in 2017 has such remote control capabilities [7]. The infection pattern of this type of malware is shown in Figure 1, where the infected host communicates with the control server to perform further malicious acts. Due to the highly concealed and high-risk nature, the detection of malware has received considerable critical attention from both academia [28, 69, 71] and industry [8, 9, 13].

Real-world malware traffic detection requires efficient detection over complex network traffic data with high accuracy. However, the emergence of encryption strategies and various evasion technologies of adversaries poses huge barriers to effective malware traffic detection. More specifically, adversaries employ encryption protocols (i.e. TLS protocol [52]) in the process of malware communications to hide suspicious information. According to [18], in 2021, more than 46% of malware have encrypted their communications. The encryption on most traffic greatly curtails the accessible information (e.g., URL and HTTP headers in payload,) that indicate malicious network behaviour. This impairs the accuracy and consistency of traditional network-based malware detection, such as Deep Packet Inspection (DPI) [47, 54].

Many previous works have made efforts to resolve the challenges of encryption. For example, to enlarge the set of indicative information, [3, 19] summarise the fine-grained features of TLS traffic and use machine learning methods to identify encrypted traffic generated by *known* malicious samples. Further, [37, 42] introduce deep learning methods to reduce the dependence on features and human experience. However, such detection methods based on features of singular streams cannot be generalized to *unknown* samples and may induce a large number of false positives that cannot be interpreted. To overcome such issues, another trend of methods [34, 35] attempt to reveal the essential characteristics of malware behaviour through the accessing relations between hosts and domains. However, adopting such methods in the real-world network scenario is not trivial due to efficiency concerns. More seriously, adversaries have developed various evasion techniques by perturbing stream features, which further reduces the amount of usable information

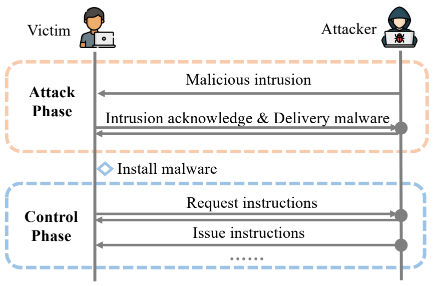


Figure 1: Infection pattern of remote control malware.

for malware identification. So far, the issue of lacking representative information for encrypted malware detection has not been well resolved. Previous works only use a single type of characteristics such as stream features or context information while the information loss and compression still exist.

In this paper, we propose ST-Graph, a multi-stream analysis framework that explores multiple features from spatial and temporal perspectives and integrates all available information for comprehensive malware traffic detection under encryption scenarios. More specifically, we design an attribute heterogeneous graph oriented to spatio-temporal traffic behaviour, which effectively captures the characteristics of associations between nodes in a large-scale network. Equipped with a graph representation learning method, our method extends the scope of information that can be utilized for recognizing encrypted malicious traffic, which boosts the detection to achieve higher accuracy and robustness. In the meantime, we carefully design our graph representation learning algorithm on the basis of random walk [49], which learns spatial and temporal features for network traffic with high efficiency. The learned host representations aggregate information from behavioural sequences in the network, which effectively highlights the differences in traffic between compromised hosts and benign samples with little information loss.

Our main contributions are summarised as follows:

- We propose ST-Graph, a real-time malicious traffic detection framework under an encryption scenario. By exploring and integrating multiple features, ST-Graph effectively reveals malicious behaviours within an encrypted network, which enables low false alarm rate detection.
- We design a heterogeneous attribute graph for encrypted traffic and propose a novel embedding method, namely interval-inclined random walk, for exploring and incorporating spatial and temporal characteristics of the traffic data.
- We evaluate our detection system in several real network scenarios for up to a year and observe good results. Compared to other works, our detection model results in higher accuracy (nearly 10 times that of baselines), and significantly lower false positives with a tolerable time cost.
- By real-world deployment, our detection system finds some malicious cases that cannot be discovered by other systems and reveals some emerging malicious traffic types.

2 BACKGROUND

In this section, we briefly overview the available information for malicious traffic detection under encryption and summarise the knowledge about graph representation learning.

Information in TLS handshake. Transport Layer Security (TLS) [52] is utilized to assure confidentiality and integrity between two communication entities. For example, HTTPS is the plain text HTTP protocol over TLS. In general, a TLS connection negotiates keys via a TLS handshake and then uses secure symmetric to encrypt communication payloads. Encryption of TLS could prevent any third parties from accessing plain communication payloads, which attracts an increasing number of applications to deploy TLS. As a coin has two sides, malicious adversaries also deploy TLS to encrypt their connections to evade detection by DPI systems [47, 54]. However, TLS handshake remains in plain-text in order to exchange the information necessary for encryption. During a TLS handshake, two communicating parties authenticate with each other, negotiate encryption algorithms, exchange encryption keys and finally agree on the encryption process based on previously exchanged information. More specifically, the client initially sends a ClientHello message, providing a list of cipher suites and a set of TLS extensions. The cipher suites are a set of cryptographic algorithms required by TLS. The extensions are the features supported by the TLS client, where the Server Name Indication (SNI) extension indicates the hostname of the target server to which the client is trying to connect. The server then responds with a ServerHello message, containing the selected algorithm and identity information. As the information exchanged during the handshake process is critical to the subsequent encrypted communication, it also provides vital information to detect malicious encrypted traffic [2].

Graph Representation Learning. Graph is a data structure for representing complex networks and modelling abstract concepts such as relations between entities. However, traditional graph construction algorithms (e.g. adjacency matrices [53], adjacency list [10]) and graph analysis algorithms (e.g. search algorithms [59]) have difficulty in solving the increasingly complex graph topology and can incur huge space and time overheads. At the same time, graph representation algorithms [24] allow for more flexible and efficient analysis on large-scale graphs, with the goal of optimizing a set of vectors that numerically represent node information as well as graph structure. Among such methods, graph embedding [21] is a representative approach, which transforms nodes on graphs into vectors. A graph could be approximated by several node lists, which is similar to the word sequences in a piece of text. Therefore, word2vec [39], a text embedding method, could be adopted for graph embedding effectively [49]. First, a list of nodes can be regarded as a "corpus" of a graph, which can be obtained by random walks [16]. Based on the node list, the vectorized representations of nodes can be optimized using a skip-gram network [40], by maximizing the likelihood probability of each node's context. In this paper, we extend this method to learn edge representations from the prioritized graph topology constructed from the complex, real-world network traffic data.

3 PROBLEM STATEMENT

Our goal is to detect infected hosts within an organization by observing host traffic behaviours. Generally, *gateway* refers to the network boundary to distinguish between internal and external networks, such as campus networks and enterprise networks. Due to the gateway's global perspective of the traffic through it, we position our detection system, ST-Graph, at the gateway to monitor the

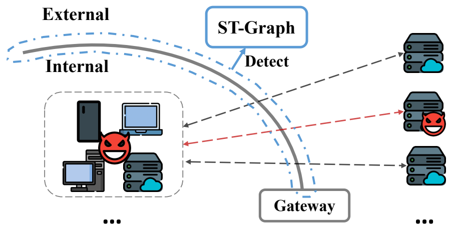


Figure 2: Threat model of ST-Graph.

whole traffic, as shown in Figure 2. Specifically, ST-Graph listens to encrypted network traffic generated by internal hosts accessing external servers at the gateway and detects infected hosts with suspicious communications in a real-time manner. In this work, we only focus on the standardized TLS protocol encrypted traffic. Since it's the mainstream encryption protocol with the most usage rate for malware, due to its ease of deployment. To be noticed, our detection system only captures the traffic without manipulating them and thus will not affect the benign forwarding traffic.

However, such detection demand poses huge challenges mainly from two perspectives: 1) curtailed available information limits the *effectiveness* of the detection, and 2) the highly comprehensive network connections hinder the *efficiency* of the detection. More specifically, the first concern results from recent encryption and adversaries' evasion technologies, which greatly curtail the amount of information that can be utilized for detection. In encryption scenarios, the invisibility of payloads of communication prevents traditional detection methods, e.g. DPI methods [47, 54], from making accurate predictions. More seriously, adversaries have evolved several techniques to evade existing single-stream-based detection methods [17, 61, 65]. In obfuscation-based evasion, adversaries change frequency of accesses from a host to command and control (C&C) servers [26], which blurs the suspicious property of the host to evade detection based on packet length and time interval [3, 19, 45]. In disguising-based evasion, malware takes the form of benign software connections, disguises that its traffic is generated by benign software [17], visits benign third-party websites initially as a front [68], or reduces the number of websites visited [38] to avoid context-based detection [50, 73]. The suspicious behaviours hide behind millions of legitimate requests, which poses substantial barriers to effective detection and introduces a high level of false positives [11, 17]. Therefore, encryption and evasion behaviours prompt us to explore more available information for effective detection.

As for the second concern, the detection over network interactions confronts a large scale of network throughput, resulting from the increasing number of network users and the complex network connectivity. Meanwhile, the demand of exploring additional, multi-faceted features for effective detection also intensifies the computational complexity. Therefore, we need to design new detection technologies that can explore and process more network features over comprehensive network interactions with tolerable computational complexity.

4 SYSTEM OVERVIEW

In this paper, we propose ST-Graph, a multi-stream analysis framework equipped with a novel graph embedding algorithm. In this section, we illustrate the key observations of malware infections, explain the design of the system, and describe the workflow of ST-Graph.

4.1 Key Observations

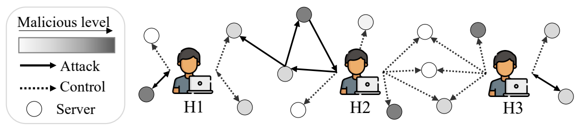


Figure 3: Schematic representation of host connectivity in a network environment with malware infections. Colour of circle represents the maliciousness of the destination server. For example, white circles represent normal destination servers, like www.google.com, grey circles represent middle-malicious servers, like phishing websites, and black circles represent malicious servers, like C&C servers.

Since remote control malware has the advantage of low attack cost and the capability of widespread, it has been widely taken by adversaries in more than half of malware attacks [30]. To detect remote control malware, we first perform an empirical study to condense the key features. Based on previous work that found that some malware has similarities in attack targets and attack methods [29, 46, 57], we perform a manual analysis of remote control malware from two aspects: code and generated traffic. First, we randomly sample 30 malware as our ground truth and run them on several operating systems under our control. Note that the entire experiment environment is enclosed and will not affect any third parties. From the code aspect, we reverse their code and analyse the code logic and hard-coded content. We find that malware in the same family prefers to share a similar software framework, which is the root cause of the similar traffic behaviour. The reason for code sharing and reuse, as we speculate, is likely to be saving costs. And malware is generally not up to date and still accepts low version TLS connections. Besides, developers of malware often use default fixed parameters, especially the time interval. For example, we find that some malware usually set the time interval to connect to the control servers to 60 seconds.

Further, to explore the traffic behaviour, we try to decrypt the traffic generated by ground truth malware. With the master key extracted from controlled operating systems, we could analyse the plain communication between malware and servers. The similarity of code within the same family leads to a concentration of behavioural characteristics in their traffic behaviour. Due to a clear, joint attack purpose, i.e. remote control of the infected hosts, there exists a relatively fixed traffic pattern in remote control attack. As shown in Figure 1, the communication process can be summarised into two main phases: the *attack phase* (solid lines) and the *control phase* (dashed lines). Initially, the attack phase is to deliver malware to victim's machine by spoofing websites (grey circles) or exploiting vulnerabilities to install malware, which can help adversaries gain control of the host. For example, for *H2* in Figure 3, the attacker commits a fraudulent activity by sending phishing emails in the attack phase and lures victim to click the embedded link, which may cause the redirection from phishing website (grey circle) to

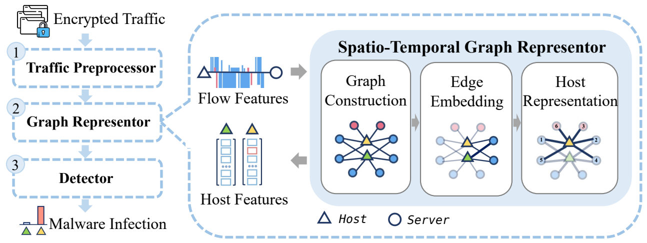


Figure 4: Overview of ST-Graph.

malware download site (black circle). Afterwards, in the control phase, infected hosts would connect to the control server of adversaries to receive and then execute instructions [46]. We find that most malware prefers to connect to different destination servers in different phases, which construct a connection order. We speculate the reason may be to evade the detection. Interestingly, the forged innocent network connectivity test of malware is common, which is preparation before connecting to control servers, e.g., querying the host's public IPs or downloading servers' certificates. For example, in Figure 3, dashed lines connected to white circles represent the network connectivity test behaviour. Specifically, the infected hosts may actively connect to external servers, such as third-party servers for network inspection.

Above, we summarise the two main phases of remote control malware from code and traffic analysis, i.e., *attack phase* and *control phase*. Besides, we find an exclusive observation, i.e., the forged innocent network connection test, which gives us a unique perspective on detection. Our observations not only extend existing understanding of malware behaviour but also deliver distinctive findings. Below, we summarise the common characteristics of malware infections from the following aspects:

- *Spatial Feature* refers to the concentration property between hosts and target servers, especially within families. As mentioned before, due to framework reuse, some malware within the same family exhibits high similarity, reflected by the destination servers they try to connect with. For example in Figure 3, *H2* and *H3* are connected to the same destination servers in the control phase.
- *Temporal Feature* of the connection. The connection order of a host over a period of time can help us rebuild the infection process of malware, which can clearly show the change with phase. Besides, due to the setting of fixed parameters, communications between infected hosts and control servers are regular, especially the packet length and packet interval time [26].

4.2 System Design

To achieve effectiveness and efficiency, the key point of our detection system, ST-Graph, is to retrieve as much information as possible with tolerable computation complexity. According to the observations of our empirical study (See §4.1), we summarise two categories of general and distinguishable features. First, *spatial feature* reflects the property of network connection relationships for each host. In addition, *temporal feature* shows the side-channel information of communications. These two kinds of features jointly depict the network communication behaviour of hosts in a comprehensive way.

Based on the observed features, we design ST-Graph, which incorporates the spatial (i.e., accesses between hosts and servers) and temporal (i.e., connection order and time-related information) characteristics of hosts' network behaviour into host representations to boost the effectiveness of detection decisions. To achieve higher efficiency, we improve the algorithm of graph representation by only optimising edge representation with iterative updates, while the optimal node representations are derived from closed-form solutions. This significantly reduces the computational complexity of graph representation learning.

4.3 Workflow

As shown in Figure 4, our detection system consists of three components: data preprocessor, graph representor, and detector. Below we elaborate on the functionality of each component.

Traffic Preprocessor. Raw traffic is usually captured as packets in the Pcap format. In this original data format, many communication details are presented in a fragmented manner, making it difficult to be utilized for detection. Therefore, we construct a *Traffic Preprocessor* to recover the complete communication while removing the extra meta-information from the traffic data. To begin with, we apply stream reassembly to recover the entire end-to-end communication. To do this, we extract a 5-tuple (i.e., <source IP, source port, destination IP, destination port, protocol>) from each original packet, and integrate the packets associated with the same 5-tuple as a stream. Afterwards, we reserve the traffic of the TLS protocol with a complete TLS Handshake and extract the key information for each stream. More specifically, we extract the contents of the TLS handshake in each reserved TLS stream, including the TLS version and the supported cipher suites, and calculate some statistical information such as the number of packets and the bytes within a stream. Such statistical information extracted from the data stream remains an important element of malware traffic detection and is used later in the computation of the graph representation.

Graph Representor. We build a heterogeneous graph to represent the end-to-end communications among hosts and servers by their temporal and spatial features. Specifically, the temporal features are the traffic characteristics we extract from the streams, which are embedded in the stream representations; and the spatial features are reflected by the graph structure. Based on this, we design a node embedding algorithm to transfer the spatio-temporal features into host representations, which reflect the similarity of the host's access behaviour. We will elaborate on this module in §5.

Detector. Given the embedding vector of each host that numerically quantifies the host's behavioural characteristics, we apply a machine learning approach to calculate the likelihood of the host being infected. Considering the ability to resist overfitting [12] and the better performance in comparative experiments (see Appendix A), we finally employ the random forest (RF) regression algorithm [12] as our detector. It is an ensemble learning method that makes predictions by averaging the output of multiple decision trees. With the predicted infection value of each host, the detector outputs a list of suspicious hosts along with their access information.

5 ST-GRAPH: SPATIO-TEMPORAL GRAPH

In this section, we present the design details of ST-Graph, elaborate on the process of graph construction and the optimization of edge representation, and explain how we propagate spatial and temporal information in a lossless way to represent hosts for malware traffic detection. The key of the approach is to exploit the similarities in the network behaviour of different applications. To do this, we first build a heterogeneous graph to correlate all network connections between hosts and servers. Based on the graph structure, we apply Random Walk [16] to generate a list of corresponding connections for each stream and use a probabilistic model to optimize the edge representations. Each stream is represented by its edge representation and traffic characteristics. Finally, we represent each host by its all accesses generated in a sequential order, which integrates the traffic characteristics and network structure associated with the host. Compared with Graph Neural Network-based methods that train thousands of parameters [72], and Knowledge Graph Embedding-based methods that jointly optimize the representation of all nodes and edges [36, 66], ST-Graph learns edge embeddings based on random-walk with a small number of iterations and optimizes host embeddings with a closed-form solution. This highly reduces the computational complexity to meet the need for real-time detection.

5.1 Graph Construction

Graph Topology. We design a host-server bipartite graph $G = (H, D, E, S, I)$ to capture the interactions between internal hosts and external nodes. We denote the internal hosts as a vertex set H, each represented by its IP as the unique identifier; Similarly, the external server destinations are represented by a vertex set D, each initialized by the server's domain name or its IP address. The domain can be obtained from SNI in ClientHello, and the server's IP address is used as a substitute when the domain is not available. We use an edge set $E = \{e|e = (h_e, d_e), h_e \in H, d_e \in D\}$ to denote all connections (i.e., streams) between hosts and servers. If a host h has a TLS handshake with a server d, then an edge $e = (h, d)$ connecting these two nodes will be added to E. $I = \{i_e|e \in E\}$ denotes the temporal feature of each edge, where i_e represents the order of e connected by its host h_e. $S = \{s_e|e \in E\}$ denotes the attributes extracted from the streams as described below.

Stream Attributes. The stream attributes S characterize the features of every single stream. To explore more detailed features, we extend previous works [3, 55] with TLS handshake features such as orders of cipher suits. Specifically, we exploit stream attributes (details in Appendix E) from three aspects: 1) the TLS handshake features extracted from internal hosts, 2) the domain name features extracted from external nodes, and 3) the side channel statistics features of the stream.

For the first type of attributes, we observe that malware and C&C servers are not updated promptly, so they usually accept lower TLS versions and maintain support capabilities for weak encryption algorithms. The underlying reasons are that attackers are relatively less concerned about whether communications will be decrypted and that vulnerabilities in lower versions of operating systems are easier to exploit. According to this observation, we extract features from the TLS ClientHello message, including the TLS version, the list of offered cipher suites and extensions as TLS handshake features. Such features can provide information about the encryption algorithm supported by the client. For the second type of attributes, our feature extraction is mainly based on the fact that many malicious domains are computed by Domain Generation Algorithms (DGA). Hence, we extract features such as the number, the character ratio, and the vowel or consonant letter ratio for identification. We also extract domain name length features to cope with lexicon-based DGA, which selects words from proprietary dictionaries for combination to reduce randomness. In addition, we include the side channel information as our third type of attributes for they provide indirect signals to our detection. Although the packet length and the arrival time do not provide insight into the content of the connection, they can help to infer network behaviour. For example, the large number of heartbeat packets used to maintain a connection can result in small packets per stream [3]. So we calculate some statistical information such as the number, the length, or the time interval of packets sent and received within a stream.

5.2 Edge Embedding

Edge embeddings are numerical representations of streams. In this paper, we integrate two categories of features into edge embeddings: 1) stream attributes S for preserving original traffic information and 2) spatio-temporal embeddings for reflecting comprehensive and dynamic network behaviours. We consider the network interactions as the *spatial features* generated from graph topology, while the *temporal features* refer to the sequential order of the servers being visited by the host through the stream. To integrate spatial and temporal features, we propose an *Interval-inclined Random Walk* $R : e \rightarrow r_e$ that takes into account both network interactions and access orders to generate Spatio-temporal embeddings. Our model extends the traditional random walk method [49] which traverses the graph along the edges and maximizes the similarity of all edges within several traverses. The main reason for this design choice is that other graph-based models for edge representation learning, such as Knowledge Graph Embedding methods [36, 66] and Graph Neural Networks [25, 63], suffer from high computational complexity and cannot serve our demand for efficient, real-time detection. In this paper, we incorporate the temporal feature, i.e., the access order, into the context generating process proposed by node2vec [22], where the random walk strategy balances the breadth-first sampling (BFS) and depth-first sampling (DFS). In general, from a starting edge o, we perform random walks with at most P_L steps for P_N times to collect a set of edges $\mathcal{N}_o$. Further, we optimize the Spatio-temporal embedding of o by maximizing the similarity of edges in $\mathcal{N}_o$. Below we elaborate on our augmented random-walk strategy, which decides the edge to take for the next step based on previous wanderings.

For an edge e, we define its neighbours as a collection of edges that share the same host or the server with e, formally $C_e = \{u : (h_u, d_u)|u \in E, h_e = h_u \vee d_e = d_u\}$. Let $w = [w_0, w_1, ..., w_{P_L}]$ be a certain random walk. We define the *connection order distance* d between two edges u, v as $d(u, v) = |i_u - i_v|$, where i_e is the order of edge e being connected by its host h_e (See §5.1). Assume the random walker starts from edge o, we define the probability of selecting a certain edge $x \in C_o$ from the neighbours of o for the next walk as

$$\Pr(w_1 = x|w_0 = o) = \frac{\frac{1}{d(o,x)}}{\sum_{y \in C_o} \frac{1}{d(y,o)}}.$$

Here, our random walker inclines to select the edge having the minimum connection order distance with the current walk as the next step. By doing this, streams that are generated in a small time interval are tended to be consecutive steps in the random walk.

For the following walks, each selection is based on the two previous walks. Assume the last step is at edge u and the current step takes edge v, the walker selects the next step from v's neighbours $x \in C_v$ with

$$\Pr(w_{i+1} = x | w_i = v, w_{i-1} = u) = \frac{\alpha_{u,x} \cdot \frac{1}{d(v,x)}}{\sum_{t \in C_v} \alpha_{u,t} \cdot \frac{1}{d(v,t)}},$$

where the value of $\alpha_{u,x}$ is determined by the relation between x and u. When $x = u$, $\alpha_{u,x}$ takes the value of $\frac{1}{p}$; when x is one of the neighbours of u, $\alpha_{u,x}$ is set as 1; otherwise $\alpha_{u,x} = \frac{1}{q}$. Here, p and q are hyper-parameters. When edge x is overlapped with edge u, we use a constant p to control the probability of returning from v to the starting point u. Otherwise, if there are one or more edges between edge x and edge u, we apply a constant q to control the probability of v going to a new node. If $q < 1$, the walker tends to visit global nodes (DFS); if $q > 1$, the walker tends to visit local nodes (BFS), which enhances the coverage of the surrounding neighbours.

According to this strategy, for each edge e, we generate a *network neighbourhood* $\mathcal{N}_e$ by P_N times of random walks. The edges in $\mathcal{N}_e$ are highly correlated with edge e from both spatial and temporal perspectives. Hence, to incorporate the spatial and temporal features of the graph into edge embeddings, we set up a vector r_e for each edge, and optimize the vector by the proximity within its network neighbourhood $\mathcal{N}_e$. More specifically, we model the plausibility that an edge n is correlated with edge u as

$$\Pr(n|u) = \frac{\exp(r_u \cdot r_n)}{\sum_{v \in E} \exp(r_u \cdot r_v)}. \tag{1}$$

We assume the neighbourhood relation between different edge pairs is independent, and further define the neighbourhood likelihood of edge u as

$$\Pr(\mathcal{N}_u|u) = \prod_{n \in \mathcal{N}_u} \Pr(n|u). \tag{2}$$

We optimize the neighbourhood likelihood of all edges to pursuing the global proximity:

$$\begin{aligned} &\max_{r_u} \prod_{u \in E} \Pr(\mathcal{N}_u|u) \\ &= \max_{r_u} \sum_{u \in E} \sum_{n \in \mathcal{N}_u} \log \frac{\exp(r_u \cdot r_n)}{\sum_{v \in E} \exp(r_v \cdot r_u)} \\ &= \max_{r_u} \sum_{u \in E} \left[-|\mathcal{N}_u| \cdot \log Z_u + \sum_{n \in \mathcal{N}_u} r_u \cdot r_n \right] \end{aligned} \tag{3}$$

where $Z_u = \sum_{v \in E} \exp(r_v \cdot r_u)$ is a normalization factor.

We use negative sampling to obtain an approximation of Z_u and optimize the objective function (3) using a stochastic gradient ascent algorithm to update the Spatio-temporal embeddings of edges. Finally, we represent edge e by concatenating its stream feature s_e and its spatio-temporal embedding r_e: $f_e = \phi([s_e||r_e])$, where $\phi(x) = \frac{x}{|x|}$ is a normalization function.

5.3 Host Representation

To detect infected hosts through traffic behaviours, we propagate the information of the streams associated with each host to numerically represent the hosts. Formally, for host h, we have its representation g_h as $g_h = \textsc{Propagate}(\{r_e | e \in E \wedge h_e = h\})$ to represent the host's sequential access behaviour in a short period. Below we explain the details of the information propagation from streams to hosts.

For a target host t, assume u is a server being visited by t through the stream $e = (t, u)$. Among network traffics, each host may access many public services (e.g., windows update service, etc.), some of which are not informative for characterizing the host's behaviour while taking a large proportion of traffic data. Hence, we rank the streams associated with the host according to the importance of the server visited through the stream before the information propagation. Such importance is with respect to 1) the frequency of the server being visited by all hosts H and 2) the order ($i_e : e \in E, h_e = t \wedge d_e = u$) of the host t visiting the server u. Intuitively, if a server u is visited by most hosts, it is likely to be visited by the target host t, too. Also, if t visits u at a very early stage, then u is an important access destination of t, compared with the other servers. Hence, we define the importance of edge e to its host t as

$$\rho(e) = \frac{|\{e' | e' \in E \wedge d_{e'} = u\}|}{i_e \cdot |E|},$$

where $|\{e'\}|$ is the number of edges ending at u, $|E|$ is the total number of edges, and i_e is the order of edge e being connected by t. Further, we model the correlation score between host t and edge e as a weighted sum over the importance of e to t and the similarity between the host representation g_t and edge embedding f_e:

$$\text{corr}(t, e) = \lambda \rho(e) + (1 - \lambda) \frac{\exp(g_t \cdot f_e)}{Z},$$

where λ is a scalar hyperparameter and $Z = \sum_{i \in E} \exp(g_t \cdot f_i)$ is a normalization factor. Let $\mathcal{L}_t = \{e | e \in E \wedge h_e = t\}$ denote all edges proceeding from host t. The joint correlation score of all edges in $\mathcal{L}_t$ is defined as:

$$\textsc{Corr}(\mathcal{L}_t) = \prod_{e \in \mathcal{L}_t} \text{corr}(t, e) = \prod_{e \in \mathcal{L}_t} \left[\lambda \rho(e) + (1 - \lambda) \frac{\exp(g_t \cdot f_e)}{Z} \right],$$

and its logarithmic form can be approximated by the first-order Taylor Expansion as

$$\begin{aligned} \log\left[\textsc{Corr}(\mathcal{L}_t)\right] &= \sum_{e \in \mathcal{L}_t} \log \left[\lambda \rho(e) + (1 - \lambda) \frac{\exp(g_t \cdot f_e)}{Z} \right] \\ &\approx \left[\sum_{e \in \mathcal{L}_t} \left(\log\left(\lambda [\rho(e) + \xi]\right) + \frac{\xi}{\rho(e) + \xi} \right) \cdot f_e \right] \cdot g_t, \end{aligned}$$

where $\xi = \frac{1-\lambda}{\lambda Z}$ is a constant. We restrict the host representation g_t as a unit vector. Let $M_t = \sum_{e \in \mathcal{L}_t} f_e \cdot \left(\log\left(\lambda [\rho(e) + \xi]\right) + \frac{\xi}{\rho(e)+\xi} \right)$. To maximize $\log[\textsc{Corr}(\mathcal{L}_t)]$, g_t should be a vector of length 1 having the same direction with M_t. Therefore, we have the optimized host representation as $g_t^* = \frac{M_t}{|M_t|}$.

6 EXPERIMENTAL EVALUATION

In this section, we present experimental evaluations of our detection system. We evaluate ST-Graph on two tasks: *malware detection* and *malware family classification*. The first task distinguishes hosts with malicious network behaviour from benign ones, while the second task further specifies the family of malware. A malware family is a group of associated programs with similar attack techniques, some of which have "code overlap" [15] to a large extent. Grouping them as a family broadens the scope of a single piece of malware as it alters over time while reserving distinct family traits.

6.1 Experimental Setup

Implementation. We present the tools used in each component of our detection system (See §4.3). For traffic preprocessor, we employ TShark (version 3.2.5) [32] to parse the fragmented packets and then recover the complete communication. To implement the graph representor, we use NetworkX (version 2.5.1) [23] to build the heterogeneous graph and initialize nodes with text representation by Gensim (version 4.1.2) [51]. Finally, we use scikit-learn (version 0.23.2) [48] for both singular value decomposition in host representation and the classification algorithms.

Baselines. We compare our ST-Graph with two state-of-the-art malicious traffic detection methods (ETA and FS-Net), which both perform detection in stream-level detection. The computational complexity of Graph Neural Network-based methods [36, 66] is too heavy to meet efficient detection needs at all. Therefore, we exclude these methods as our comparative methods. The details of the baselines are as follows.

• *ETA* [2] adopts the random forest model to detect malware traffic. Specifically, the ETA utilizes the stream features, including TLS handshake metadata, DNS contextual streams linked to the encrypted stream, and the HTTP headers of HTTP contextual streams from the same source IP address within a 5-minute window.

• *FS-Net* [37] is an end-to-end deep learning model, which takes the multi-layer encoder-decoder structure to mine the potential sequential characteristics of streams. In summary, it learns representative features from raw streams and then classifies them.

We set hyperparameters of baselines to either the values adopted by their authors or default values. Specifically, for FS-Net, we set the dimension of hidden state as 128, layer number as 2 and dimension of length embedding as 16. For the other parameters involved, such as the parameters in random forest, we apply the default settings in scikit-learn (version 0.23.2).

Environment and Parameters. We conduct the methods on a Supermicro server with two Intel Xeon E5-2690 CPUs (2× 14 cores), Centos 7.9.2009, 345G memory. For hyper-parameters, we set the dimension of the host vectors D_h as 256, the length of random walk path P_L as 100 and the number of paths per edge P_N as 10. And we set the parameter p controlling the probability of returning as 1 and the parameter q controlling the probability of exploring new nodes as 2 for better coverage of the surrounding neighbours.

Metrics. We use the following metrics to evaluate the detection performance: (i) precision, (ii) recall and (iii) false-positive rates (FPR). (See Appendix B)

6.2 Dataset

We conduct experiments on one public dataset (CICInvesAndMal-2019, as `AndMal2019` [58]) and one dataset collected by ourselves (`EncMal2021`). **`AndMal2019`** dataset includes traffic and device logs generated by 5065 benign and 426 malicious Android apps on real smart devices. And the malicious apps can be divided into 39 families. Our collected dataset **`EncMal2021`** is constructed by capturing the traffic generated by the example and the campus network traffic. It contains 108,847 hosts with 5,202,093 streams, 4.5% of which are marked as malicious, and the others are marked as benign. Below we elaborate on the data collection process of EncMal2021.

Data Collection. EncMal2021 consists of malicious and benign traffic data.

• *Malicious Traffic:* We use malware analysis sandboxes to collect the data. The sandboxes, including Windows 7 and Windows 10 operating systems, allow users to submit malicious executable examples and control the runtime based on the execution of the executable files. Malicious samples come from malware analysis website VirusTotal and large security companies we work with, with millions of sample updates per day. Table1 shows the names of malware families and the number of examples we sample from each malware family. Each example runs for 5 minutes on average, during which all generated traffic is recorded and saved. Since most of the behaviours of the samples in the sandbox can be observed in the first two minutes [31], we consider five minutes to be a sufficient time interval to observe almost all valid behaviours. In total, we end up capturing 239,007 streams from 53,111 samples.

Table 1: Malware Families

Family Name	# Samples	Family Name	# Samples
Minerd	5717	Unwanted	1610
Cryxos	4652	Faceliker	1564
PhishingSite	3080	Trojandownloader	1528
Wacatac	2949	Brocoiner	1482
hidelink	2860	Sality	1046
Kryptik	2403	Zbot	1035
Redirector	2190	RelevantKnowledge	771
Generickdz	1974	Scrinject	756
Installcore	1807	Ramnit	719
Iframe	1731	Others	13237

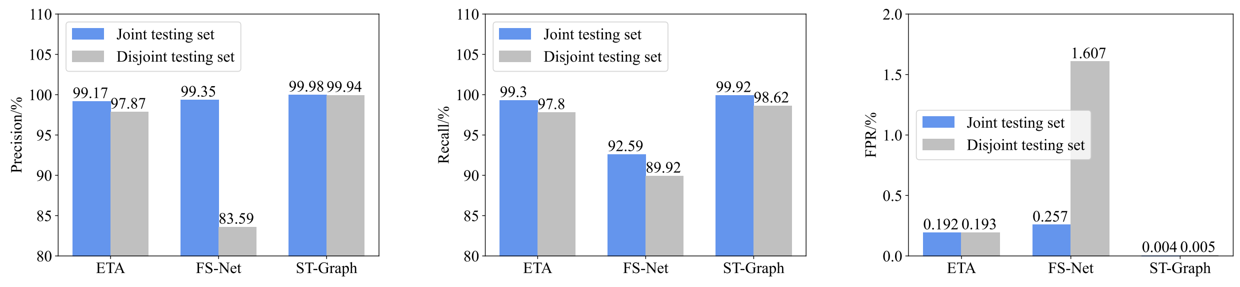
• *Benign Traffic:* We construct benign traffic from two data sources, one is real traffic captured in a campus network and the other is the traffic generated by benign samples running in the sandbox. Most samples in the benign dataset are collected from a large campus network with nearly 10,000 active hosts. We passively monitor all inbound and outbound encrypted traffic at different time points for 5 consecutive months from January 2021 to April 2021 and save the raw pcap packets. Due to the possibility of malware-infected hosts in the campus network, we cannot directly use all the traffic as benign traffic. We chose Alexa ranking as a filtering criterion, keeping domains with the top 1 million traffic rankings. The reason for this choice is that websites with more than 1 million rankings on Alexa can be considered as "Long Tail", thus this filtering guarantees the diversity of the sample and also ensures the traffic is benign to a certain extent. Since the crawled traffic may contain some private information, we also need to anonymize the IPs. The same host is treated as the same fake IP to ensure the integrity of the network relationship. On the other hand, in order to avoid large bias in the domain distribution between benign and malicious traffic due to the different collection methods, we also collect the traffic generated by benign samples running in the same sandbox environment. Such samples are partly collected from the top 100 of the Microsoft Store's top free list and partly collected from the programs flagged as benign by the malware analysis site. Among all the domains of the traffic generated by the benign samples, 24% are *not* in the Alexa Top 1 million rankings. We believe this mitigates the bias associated with filtering traffic while keeping the traffic benign. Finally, we capture 4,940,593 streams in the existing network by 53,281 hosts and 22,493 streams in the sandbox using 2,455 samples.

Dataset statement. In `EncMal2021`, the malicious samples are run in sandboxes, which makes the distribution of malicious traffic not exactly consistent with the actual situation. However, we compensate for this by enriching the malware types and the sandbox environment, so that the constructed dataset is comprehensive as possible.

Train-test Split. For the first task, i.e., malware detection, we perform different train-test split strategies on the two datasets, for the capacities of these datasets are different.

Table 2: Detection Performance of ST-Graph and Baselines. In EncMal2021, use *joint testing set* as the test set.

Dataset	Method	Malware Detection					Malware Family				
		Performance			Time Taken		Performance			Time Taken	
		Precision(%)	Recall(%)	FPR(%)	Train(s)	Test(s)	Precision(%)	Recall(%)	FPR(%)	Train(s)	Test(s)
EncMal2021	ETA	99.1726	99.3013	0.1915	729.26	138.24	79.9473	76.5882	1.2410	306.49	76.66
	FS-Net	99.3515	92.5990	0.2565	45324.10	17741.15	73.0697	72.6062	1.5218	41048.30	1890.69
	ST-Graph	99.9805	99.9221	**0.0045**	5956.45	150.31	**93.3669**	91.2105	0.4016	1489.41	127.14
AndMal2019	ETA	77.8451	74.7129	19.0409	65.31	22.48	28.8382	27.5170	1.8134	139.05	39.19
	FS-Net	72.8997	72.5344	21.4712	23275.86	5112.93	19.4906	16.7309	2.0863	23055.60	774.21
	ST-Graph	99.2973	99.6444	**0.0170**	2113.13	27.19	**53.7568**	53.4014	1.0180	520.36	66.85



(a) Precision comparison (higher is better) **(b) Recall comparison (higher is better)** **(c) FPR comparison (lower is better)**

Figure 5: Comparison between *joint testing set* and *disjoint testing set* on EncMal2021.

• For EncMal2021: We divide the collected dataset into three parts: 1) *training set*, 2) *joint testing set* and 3) *disjoint testing set*, with a ratio of 6:2:2. Here, the joint testing set shares the same distribution of malware families with the training set; while the malware families in the disjoint testing set do not intersect with those in the training set. And we allocate benign traffic randomly and keep the ratio of benign traffic to malicious traffic at 10:1.

• For AndMal2019: We use 60% samples for training and 40% for testing. Here, we do not further divide the testing set into a joint or a disjoint one since AndMal2019 only includes a limited number of malicious families.

As for the second task, i.e., malware family classification, we use 8:2 as the ratio of train-test split for *both* two datasets. The labels for this task are the name of families in Table 1.

Ethical Concerns. We also consider ethical concerns when collecting our dataset. As aforementioned, a portion of our benign traffic comes from border traffic on the real campus network. To avoid burdening normal network access, we limited our collection to passive listening. To protect privacy, we only saved TLS traffic when capturing traffic and anonymized the address of the communication. In the detection stages, we focus on the destination of the traffic (domains) and do not look into the payload of any TLS traffic. Also, the traffic is stored in the physical servers to which only privileged administrators have access.

6.3 Detection Performance

We compare ST-Graph with other baselines on EncMal2021 and AndMal2019, respectively.

6.3.1 Evaluation on EncMal2021. We use EncMal2021 to verify the model has good malware detection abilities in a large-scale dataset, i.e. it can achieve good binary and multi-classification results.

Malware Detection. The purpose of this experiment is to evaluate the detection effectiveness of ST-Graph and to test whether it can identify unknown malware families, i.e., generalization. Generalization is to describe a model's ability to react to new data. In malicious traffic detection, it is important to know how well a trained model will generalize to unseen data. We train all models using *training set* and test them with *joint testing set* and *disjoint testing set*, respectively.

The results of the *joint testing set* are shown in Table 2 (1st row). From the results, we observe ST-Graph approaches with high precision and recall over of 99.99%, and the FPR is even two orders of magnitude lower than the other two models. In terms of computation cost, compared with ETA, a simple feature engineering-based model, ST-Graph takes more time for training and testing, while this is much less than the deep learning-based model FS-Net.

Figures 5a, 5b and 5c show the detection results of the three models for the *disjoint testing set*, compared with the results on the *joint testing set*. In terms of precision, only our model does not degrade on unseen samples. This is due to the fact that although its recall for malicious communication traffic drops a little, it avoids more false positives. As for the recall score, each model decreases to some extent, while our model is still the most effective one. FS-Net can detect most malware traffic, but it also introduces a high level of false positives even when the distribution of benign traffic data changes slightly. This may be due to its overfitting, where small changes can lead to changes in the results of the model, making it unrealistic to be applied to security analysis in practice.

Malware Family Classification. We use only malicious data for multi-classification to understand ST-Graph's effectiveness for malicious family classification. This task is valuable because when malicious communications are detected, we need to analyse their binaries to confirm the alarm, while the family labels of the alarms offered by the detection model itself greatly reduce analysis time.

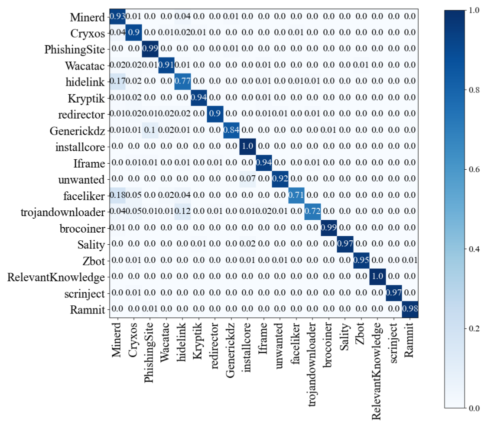


Figure 6: Malware family classification result of ST-Graph.

Figure 6 shows the normalized confusion matrix of multi-class classification for ST-Graph. And the results of the other two models show in Appendix D. We observe that ETA model and FS-Net model can only identify a small part of malware families while having poor performance in the majority of them. For example, ETA performs a good classification for RelevantKnowledge. After analyzing the raw traffic data generated by all samples in RelevantKnowledge family, we found that all samples have the same cipher suites, signature algorithms and other characteristics in the TLS handshake phase. These features also differ from the traffic generated by the samples in other families, forming a unique fingerprint. Beyond that, the distinction between malware may not be obvious, and the features extracted by humans are designed to detect anomalies rather than to distinguish between malware families. On the other hand, FS-Net prefers to group malware into large malware families because large malware families have a greater variety of packet length sequences. Hence, it is more difficult for FS-Net to distinguish between malware with similar packet length sequences.

Our model can achieve 93% precision in classifying malware families. Figure 11 (in Appendix D) visualizes the results of embedding the traffic graphs into six of these families. It shows that malware families can be distinguished based on the clustering results, and each family can form a large cluster. In addition to some intuitive TLS handshake features, our model can capture these same-family network relationships and so achieve better classification results. However, our model still cannot classify malware families completely correctly. On the one hand, this is because malware families are classified by security analysis tools based on the software binary, and the labels themselves may not be completely accurate; on the other hand, there is still a portion of malware that may not present complete communication in the sandbox. These two reasons can cause our model to misclassify malware that is not well differentiated. In general, our model can perform fine-grained classification.

6.3.2 Evaluation on `AndMal2019`*.* We also test our model on a publicly available dataset to verify that it is applicable to a wide range of data with good detection results.

Malware Detection. We first evaluate the detection effectiveness on this dataset, i.e. the binary classification performance. Table 2 (2nd row) illustrates the results and Figure 7 shows the feature

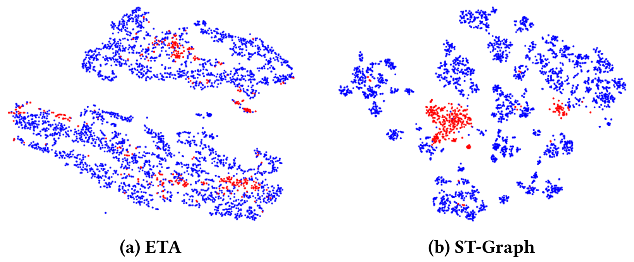


Figure 7: Visual distinction of features where red dots are malware.

vector visualization of the ETA and ST-Graph. The figure clearly shows that our approach enables a greater aggregation of malware compared to ETA, which enables a better classification result.

Malware Family Classification. The results in Table 2 show that it is possible to detect malware using only network traffic, but single-stream methods are not sufficient to characterize the malware families. Our approach makes it easier to find correlations between malware. By combining the spatial-temporal characteristics of hosts accessing the network, our detection results can be improved considerably.

6.4 Generalization of ST-Graph

In addition to examining the model's generalization to new infections (§6.3.1), we also conduct ablation studies to explore the role of stream attributes and spatio-temporal embeddings.

6.4.1 Features. We reduce the variety of features of edges to confirm that ST-Graph can detect malware in less informative and more critical situations. TLS 1.3 [52] has been standardized in 2018 and the adoption rate reaches 48% on major websites by January 2021 [27, 33]. The main change in this version is it enhances security by also encrypting the handshake process, while this results in less information being available in ClientHello. And the proposed Encrypted-SNI (ESNI) extension will prevent others from fetching the server name [6, 14]. This directly affects the acquisition of TLS handshake features and domain features for Spatio-temporal Graph. Even so, our method still works under the stringent conditions of TLS 1.3 and remains efficient and effective. We run ST-Graph on `EncMal2021` without stream attributes that are blocked by TLS 1.3, and the results are shown in Table 3 (2nd row), where the stream attributes are removed one by one from top to bottom.

According to the results, we conclude that as the feature types are reduced, there is a slight weakening in the precision and a slight increase in the FPR of our method. Without using any stream features, ST-Graph is still able to achieve satisfactory results and even causes a lower FPR compared to baseline models using all stream attributes (See Table 2). This suggests that the spatio-temporal features extracted by our method play a decisive role. Therefore, our approach is still competitive in detecting malicious traffic after the full deployment of TLS 1.3.

6.4.2 Embedding. We modify the graph embedding algorithm in Graph Representor to standard techniques (such as DeepWalk [49] and node2vec [22]), leaving the other modules unchanged for testing. This experiment is also tested on `EncMal2021` and the results are recorded in Table 3 (3rd row). The results show that our system has good detection and generalisation capabilities and that our graph embedding algorithm does perform better in our scenario compared to other graph embedding algorithms.

Table 3: Ablation study's results of ST-Graph.

Settings		Precision(%)	Recall(%)	FPR(%)
Original model		99.9805	99.9221	0.0045
Features	No domain feat.	99.9784	99.8983	0.005
	No handshake feat.	99.8522	99.5578	0.0161
	No statistical feat.	99.5848	99.0915	0.0955
Embedding	DeepWalk	99.7376	99.4148	0.0184
	node2vec	99.8891	99.5947	0.0121

6.5 Robustness of ST-Graph

We then design experiments to analyse the robustness of ST-Graph about noise labels and the sensitivity to parameters of ST-Graph.

6.5.1 Noise Labels. We evaluate how noisy data affect the proposed model in this experiment. This is worth concerning for samples running in a sandbox may not produce truly malicious behaviour in a short period of time, while such behaviours are still labelled as malicious in our dataset as long as they are generated by malware. To explore the impact of this problem, we randomly select a portion of benign hosts and mark them as malicious ones and investigate whether this changes the decision of the model.

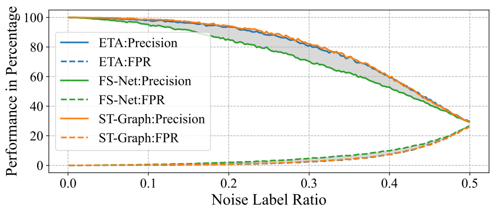


Figure 8: Detection performance with noise labels.

In the experiment, the ratio of benign to malicious hosts is 10:1. Figure 8 shows the relationship between the proportion of benign hosts mislabeled as malicious ones and the detection effectiveness. The solid line represents the precision of the detection, while the dotted line is the FPR. From the results, we observe that ST-Graph and EAT show better robustness than FS-Net. More specifically, with 15% mislabelling, ST-Graph still achieves 97% precision and lower than 1% FPR. We observe that labelling benign hosts as malicious does not change the detection rate of truly malicious traffic. Even if the samples do not generate malicious traffic when running in the sandbox and are consequently mislabeled, the impact on the model prediction is not serious by only introducing a small number of false positives.

6.5.2 Parameter Sensitivity. We evaluate ST-Graph under different parameter settings. Specifically, we keep all parameters at their default values and adjust the values of three parameters D_h (i.e., dimension of host representation), P_L (i.e., length of random walk) and P_N (i.e., times of random walk starting from each edge) in turn, respectively. The experimental results are shown in Figure 12 in Appendix D. The results show that changing the parameters within a certain range does not have a significant impact on the detection performance of the model. The system is able to consistently maintain a high detection accuracy and a low false alarm rate. For the parameters P_N, the detection performance tends to get better and then worse as the number of random walk paths per edge increases and the number of contexts per edge increases.

Table 4: Precision(%) of Real-world evaluation results.

Stage	ETA	FS-Net	ST-Graph
Campus-04/15/2021	8.0	4.0	86.0
Campus-02/25/2022	6.0	2.0	64.0
Enterprise-04/15/2021	1.0	0.5	80.0
Enterprise-02/25/2022	0.8	0.5	68.0

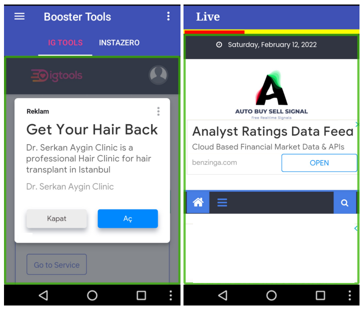
7 REAL-WORLD EVALUATION

To assess the performance of ST-Graph more comprehensively, we deploy it in two real-world scenarios, i.e., a campus and an enterprise gateway, for almost a year-long (from April 2021 to April 2022) operation and evaluation. These two network scenarios both have thousands of active users inside. Due to the significant number of users, the network throughput is large enough to prove the efficiency of ST-Graph, reaching the traffic bandwidth of 3.6 Gbps on the campus and 1.7 Gbps in the enterprise. Due to its best performance, we deploy the model trained in §6.3.1 in this real-world evaluation and compare its performance to existing detection systems (ETA [2] and FS-Net [37]).

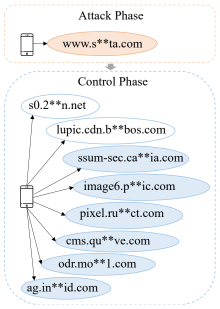
7.1 Real-word Result

Manual Analysis of Precision. To evaluate the precision of alarms, we operate a manual analysis. It is impossible to manually analyse all alarms, with nearly tens of thousands of alerts per day from these 3 models. As such, we randomly select 50 alarms for each model and analyse them manually by an expert researcher, with the help of information from threat intelligence [56]. First, we judge an alert to be correct when an identifier (domain or IP address) of the destination server is flagged malicious by threat intelligence. Then, for alarms which threat intelligence cannot cover, we manually visit this destination server and judge with response contents. If the response content compared to the contextual traffic information is abnormal, e.g., response is a phishing website, we consider this alarm is correct. To notice, if there is no correct alarm in these 50 samples, we resample 50 alarms from unsampled data space until we find at least one correct alarm, and the precision rate is calculated on all sampled alarms. Due to the random nature of sampling, we believe that our results on the sampled dataset can be representative of the global results.

Results of Campus. To illustrate the variation in detection performance over time, we choose results for the start date and end date to analyse and compare, which are shown in Table 4. At the start (April 2021), ST-Graph had the lowest average daily alarm rate of 0.237% compared to the other two detection systems (3.034% for ETA and 3.665% for FS-Net). From the precision of alarms by manual analysis on 50 random sampled alarms in Table 4, we find that the precision of ST-Graph(86%) significantly surpasses the other two detection systems. As a result, ST-Graph is accurate and meanwhile with a low false positive, which can significantly reduce the cost of manual auditing. In addition, ST-Graph even can detect a large number of false negatives that cannot be detected by other


Booster Tools
IG TOOLS
INSTAZERO
Reklam
Get Your Hair Back
Dr. Serkan Aygin Clinic is a professional Hair Clinic for hair transplant in Istanbul
Kapat
Aç
Go to Service
Live
Saturday, February 12, 2022
AUTO BUY SELL SIGNAL
Analyst Ratings Data Feed
Cloud Based Financial Market Data & APIs
OPEN

(a) Ewind examples



(b) Host' access sequence

Figure 9: Activities of Ewind.

two detection models. In order to evaluate the ageing issue over time, we compare the results of these 3 models after nearly one year (February 2022). Although there is a slight degradation in the performance of ST-Graph over time, it is still far superior to the other two models.

Results of Enterprise. As the training set for the detection model comes from the campus, the distribution is different from that of the enterprise network. Affected by the inconsistent distribution of training set, the average daily alarm rate for all models is higher than the results on campus. However, ST-Graph still has the lowest alarm rate at 0.249%, compared to 22.313% for ETA and 26.482% for FS-Net. Similar to the results of campus, ST-Graph achieves the highest precision and its performance is less affected by the time ageing issue.

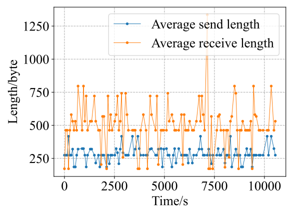
It's worth noting that our detection is efficient in both scenarios. In enterprise, ST-Graph only costs 160 seconds for 5-minute traffic with 1.7 Gbps bandwidth. Even in the more complex network of campus, ST-Graph still cost less time, with 200 seconds for 5-minute traffic with 3.6 Gbps bandwidth. Overall, our model is effective and efficient in detecting malware-infected hosts, subject only to slight time ageing issues.

7.2 Case Studies

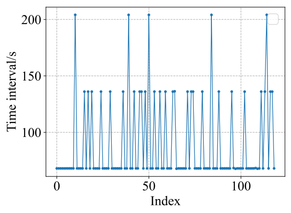
As mentioned before, ST-Graph could detect false negatives from other detection models, i.e., malicious behaviours that are not detected by other models. To understand the differences in performance of different detection models for these "unknown" malware, we analyse two representative cases: *EWind* and *Miner.*

EWind. As shown in Figure 9a, Ewind is a sophisticated, long-standing class of adware [62], profiting through displaying advertisements on the victim's device. More seriously, Ewind also includes functionality such as collecting device data and forwarding SMS messages to attackers, which poses a serious security risk.

From the results of enterprise, we identify a host infected with malware belonging to a variant of the Ewind [70] family, without being detected by the other two detection models. As we cannot access the source code of Ewind, we only analyse the traffic behaviour of these infected hosts. By extracting the connection list of this infected host, we depict the infection procedure of Ewind in Figure 9b. First, it visited a free software download site, which is peculated be the *attack phase.* Afterwards, we observe a series of access requests to multiple websites at very short intervals (almost simultaneously). By manually checking these websites, we find



(a) Average length per stream



(b) Time interval between streams

Figure 10: Statistical information on mining traffic.

that most of these websites are advertising websites (blue circles), while two websites are benign websites with promotional content (white circles). Therefore, we think that the connection sequence is the root cause for the detection of ST-Graph, which reflects the whole procedure of this malware. Apart from the anomalous observation of the connection relationship, this malware didn't exhibit any other malicious features. Specifically, Ewind works on the Android platform that uses a system interface to encrypt traffic [4]. Therefore, the encryption-related information in TLS is not different from benign applications, which leads to the failure of ETA, the TLS-information-based detection system. Besides, in the control phases, Ewind has no subsequent interaction after TLS handshake. As such, FS-Net, based on packet-based side-channel information, has no capability to detect this malware.

Miner. Malicious coin miners are an emerging attack since 2018, whose activity is similar to ransomware [43]. By unauthorized use of victim's device for cryptomining, it can consume valuable computing resources, which poses new threats to the Internet.

From the results of campus, we detect a host infected by a malicious miner. Without SNI or any other domain information, this host only connects one target server only with IP address while with hundreds of connections. Based on the response by manually connecting to this IP address, we verify that this IP address hosts an online mining pool. By analysis of the communication traffic between this host and server, we find highly regular activities, especially side-channel temporal features. As shown in Figure 10, the average length of streams between this host and server and the time interval between each connection are regular and within a fixed range. From this regular behaviour, we speculate that this infected host sends "heartbeat" messages to indicate its alive status and mining results to mining pool (control server) regularly. Although the connection sequence is not very informative, ST-Graph is still able to detect this kind of malware based on this regular activity.

From the analysis of these "unknown" malware, we have further proven the effectiveness of ST-Graph.

8 DISCUSSION

In this section, we discuss the defence ability of ST-Graph against existing attacks and the limitations of ST-Graph.

8.1 Defence Ability against Attacks

Several existing works have proposed methodologies to mislead traffic detection systems, including *disguising attacks* and *obfuscation attacks.*[1] We have already discussed in §6.4 that the extracted

[1] In this work, we don't compare attacks target to learning-based models [5, 64].

spatio-temporal features in our approach play a decisive role in enhancing the model's generalizability. On this basis, we next discuss the impact of each attack on ST-Graph.

Disguising Attack. Goal of this attack is to disguise malicious traffic as benign applications. Frolov *et al.* proposed a tool to modify TLS information to mimic other popular TLS implementations by changing the fingerprints extracted from ClientHello and ServerHello [17]. However, this attack only modifies the TLS-stream information without changing the network connection relationships. Therefore, ST-Graph can still detect infected hosts even with this kind of attack.

Obfuscation Attack. This attack is more discreet, which tends to confuse side-channel information-based detection systems. Wang *et al.* [65] change the timestamp, direction and packet size of packets to confuse detection based on packet length and time interval. Similarly, even changing the side-channel information, ST-Graph retains the connection relationships, which allows it to still effectively detect. Since obfuscation attacks will cause a highly dispersed distribution of packet lengths and time intervals, which is reflected in higher entropy. As such, it's simple to defend this attack, i.e., calculating the entropy of side-channel information.

To conclude, existing attacks have a limited impact on ST-Graph.

8.2 Limitations

Scale of network. We acknowledge that the scale of internal network has an impact on ST-Graph. The increase of internal hosts will cause more nodes and edges in our graph structure. A larger number of edges in the graph brings a heavier time cost for the detection, and therefore an infinite number of hosts cannot be handled at the gateway. However, our real-world evaluation is conducted on two large representative networks, with bandwidth up to 3.6 Gbps and covering over 10,000 hosts. This huge-scale experiment could prove the efficiency of ST-Graph in reality.

In future work, we would explore more efficient traffic feature representation methods and ease the practical deployment limitations by deploying clusters or separate inspection by network segment for large enterprise network environments with high bandwidth. In addition, the features we used rely on the TLS protocol. Although the model can still achieve 99.85% precision (Table 3) after removing protocol-related features, it is still necessary to explore the detection methods for generic encrypted traffic in the future.

9 RELATED WORK

Traditional network-based malware detection methods such as DPI [47, 54] and HTTP-based methods [41, 45], perform keyword matching on the plain-text payload of each packet. However, traffic encryption makes these methods no longer effective. Besides, limited available information under encryption makes network-based detection more challenging. We divide existing encrypted-network-based detection methods into *single-stream-based detection* and *context-based detection.*

Single-stream-based detection. Single-stream-based detection strives to dig any available plain-text information for detection in encrypted traffic. Two main features are utilized by single-stream-based detection: *features of TLS stream* and *features of side-channel.*

• *Features of TLS streams* utilize the information in the TLS Handshake process, which is the only plain-text process of TLS interactions. Anderson *et al.* extract information as representative features for detection, including TLS version, the cipher suites provided by the client, the TLS extensions used, the server certificate, and the result of negotiation between the two parties [2, 3]. Althouse *et al.* first stitched information from TLS handshake to compute JA3/JA3S fingerprints and detect malware by matching the client/server fingerprints [1]. However, methods with only these features will be ineffective under TLS 1.3, with fewer plain-text available information for detection.

• *Features of side-channel* exploits information, such as packet length, packet interval time and packet length frequency. For example, several works utilized packet length statistics information to detect malware at the TCP/IP layer [19, 60, 67]. Furthermore, Liao *et al.* utilized deep learning techniques to improve the performance of the detection models based on packet length sequences.

Above methods usually consider a single feature, which is not robust and can be easily escaped by using evasion strategies.

Context-based detection. Encryption results in limited available plain-text information for TCP connections, context-based detection methods try to use information from multiple protocols [3, 20] or multi-stream connections to extend the perspective. In fact, information from DNS could help to construct the relation graph of multiple malware servers [35, 44, 50]. To notice, Oprea *et al.* detect malware and APT infections within an organization [46], while their method only works on available seeds of known malware.

Note that relations between multiple malware and hosts may change over time, so capturing the dynamic changes is essential in the detection. However, previous graph-based methods are mainly based on static relationship graphs and usually ignore the temporal characteristics, which limits the effectiveness of detection with less information. In our work, we propose ST-Graphto detect malicious traffics based on a spatial-temporal graph.

10 CONCLUSION

In this paper, we propose ST-Graph, an encrypted malicious traffic detection system equipped with a well-designed, novel graph representation learning algorithm. By exploring additional, informative network attributes and effectively integrating multiple features, ST-Graph achieves high detection accuracy with a significantly lower false alarm rate and tolerable computational complexity. Experimental results on both self-collected dataset and benchmark dataset demonstrate the effectiveness and the efficiency of ST-Graph, compared with state-of-the-art malware traffic detection systems. In addition, ST-Graph shows good performance in both generalization and robustness perspectives and reveals outstanding efficiency by real-world deployment.

ACKNOWLEDGMENTS

We are grateful to anonymous reviewers for their constructive comments on this work. This work is supported by National Natural Science Foundation of China (Grant No. U1836213, U19B2034), the Huawei Technologies Co., Ltd under Grant No. TC20200917004. Jia Zhang is the corresponding author (zhangjia@cernet.edu.cn).

REFERENCES

[1] John Althouse. 2019. TLS Fingerprinting with JA3 and JA3S. https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.
[2] Blake Anderson and David McGrew. 2016. Identifying Encrypted Malware Traffic with Contextual Flow Data. In *AISec@CCS*. 35–46.
[3] Blake Anderson and David McGrew. 2017. Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity. In *SIGKDD*. 1723–1732.
[4] AndroidDev. 2021. Developer Guides-SSLSocket. https://developer.android.com/reference/javax/net/ssl/SSLSocket Accessed January 22, 2022.
[5] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *ICML*, Vol. 97. PMLR, 695–704.
[6] Zimo Chai, Amirhossein Ghafari, and Amir Houmansadr. 2019. On the Importance of Encrypted-SNI (ESNI) to Censorship Circumvention. In *FOCI@USENIX Security Symposium*. USENIX Association.
[7] Catalin Cimpanu. 2016. NOPEN Is the Equation Group's Backdoor for Unix Systems. https://news.softpedia.com/news/nopen-is-the-equation-group-s-backdoor-for-unix-systems-508257.shtml Accessed March 10, 2022.
[8] Cisco. 2018. Cisco Advanced Malware Protection Solution Overview. https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/advanced-malware-protection/solution-overview-c22-734228.html.
[9] Cloudflare. 2021. Cloudflare Gateway. https://www.cloudflare.com/products/zero-trust/gateway/ Accessed March 10, 2022.
[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to Algorithms*. MIT press.
[11] Neustar International Security Council. 2020. NISC Survey Results. https://www.nisc.neustar/nisc-survey-results/ Accessed March 20, 2022.
[12] Antonio Criminisi, Jamie Shotton, Ender Konukoglu, et al. 2012. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and trends® in computer graphics and vision* 7, 2–3 (2012), 81–227.
[13] Azure documentation. 2022. Microsoft Antimalware for Azure Cloud Services and Virtual Machines. https://docs.microsoft.com/en-us/azure/defender-for-iot/organizations/how-to-control-what-traffic-is-monitored.
[14] David Fifield. 2018. Anticipating a world of encrypted SNI: risks, opportunities, how to win big. https://www.bamsoftware.com/sec/esni.html.
[15] FireEye. 2020. Definition of Malware Family. https://vision.fireeye.com/editions/06/06-m-trends-fireeye-mandiant.html.
[16] Francois Fouss, Alain Pirotte, Jean-Michel Renders, et al. 2007. Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation. *IEEE Trans. Knowl. Data Eng.* 19, 3 (2007), 355–369.
[17] Sergey Frolov and Eric Wustrow. 2019. The Use of TLS in Censorship Circumvention. In *NDSS*. The Internet Society.
[18] Sean Gallagher. 2021. Nearly half of malware now use TLS to conceal communications. https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/ Accessed November 20, 2021.
[19] Ali Gezer, Gary Warner, Clifford Wilson, et al. 2019. A Flow-Based Approach For Trickbot Banking Trojan Detection. *Comput. Secur.* 84 (2019), 179–192.
[20] Paul Giura and Wei Wang. 2012. A Context-Based Detection Framework for Advanced Persistent Threats. In *CyberSecurity*. IEEE Computer Society, 69–74.
[21] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowl. Based Syst.* 151 (2018), 78–94.
[22] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.
[23] Aric Hagberg and Drew Conway. 2020. NetworkX: Network Analysis with Python. *URL: https://networkx. github. io* (2020).
[24] William L Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14, 3 (2020), 1–159.
[25] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*. 1025–1035.
[26] Cobaltstrike Helpsystem. 2021. Beacon Covert C2 Payload. https://www.cobaltstrike.com/help-beacon Accessed November 20, 2021.
[27] Ralph Holz, Jens Hiller, Johanna Amann, et al. 2020. Tracking the deployment of TLS 1.3 on the Web: A story of experimentation and centralization. *SIGCOMM* 50, 3 (2020), 3–15.
[28] Dan Jiang and Kazumasa Omote. 2015. An Approach to Detect Remote Access Trojan in the Early Stage of Communication. In *AINA*. IEEE Computer Society, 706–713.
[29] İlker Kara and Murat Aydos. 2019. The ghost in the system: technical analysis of remote access trojan. *International Journal on Information Technologies & Security* 11, 1 (2019), 73–84.
[30] Catherine Knowles. 2021. End of 2021 marks drop in cyber attacks, and increase in remote access malware. https://securitybrief.asia/story/end-of-2021-marks-drop-in-cyber-attacks-and-increase-in-remote-access-malware.
[31] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. 2021. Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes.. In *NDSS*.
[32] Ulf Lamping and Ed Warnicke. 2004. Wireshark user's guide. *Interface* 4, 6 (2004), 1.
[33] Hyunwoo Lee, Doowon Kim, and Yonghwi Kwon. 2021. TLS 1.3 in Practice: How TLS 1.3 Contributes to the Internet. In *Proceedings of the Web Conference 2021*. 70–79.
[34] Jehyun Lee and Heejo Lee. 2014. GMAD: Graph-based Malware Activity Detection by DNS traffic analysis. *Computer Communications* 49 (2014), 33–47.
[35] Kai Lei, Qiuai Fu, Jiake Ni, et al. 2019. Detecting malicious domains with behavioral modeling and graph embedding. In *ICDCS*. IEEE, 601–611.
[36] Yankai Lin, Zhiyuan Liu, Maosong Sun, et al. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*.
[37] Chang Liu, Longtao He, Gang Xiong, et al. 2019. Fs-net: A flow sequence network for encrypted traffic classification. In *INFOCOM*. IEEE, 1171–1179.
[38] Dan Mcwhrter. 2014. APT1: Exposing One of China's Cyber Espionage Units. https://www.mandiant.com/resources/apt1-exposing-one-of-chinas-cyber-espionage-units.
[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).
[41] Mamoru Mimura, Yuhei Otsubo, Hidehiko Tanaka, and Hidema Tanaka. 2017. A practical experiment of the HTTP-based RAT detection method in proxy server logs. In *AsiaJCIS*. IEEE, 31–37.
[42] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).
[43] Michael Nadeau. 2021. Cryptojacking explained: How to prevent, detect, and recover from it. https://www.csoonline.com/article/3253572/what-is-cryptojacking-how-to-prevent-detect-and-recover-from-it.html.
[44] Pejman Najafi, Andrey Sapegin, Feng Cheng, and Christoph Meinel. 2017. Guilt-by-association: detecting malicious entities via graph mining. In *International Conference on Security and Privacy in Communication Systems*. Springer, 88–107.
[45] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. 2013. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *USENIX Security Symposium*. 589–604.
[46] Alina Oprea, Zhou Li, Ting-Fang Yen, et al. 2015. Detection of early-stage enterprise infection by mining large-scale log data. In *DSN*. IEEE, 45–56.
[47] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.
[48] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.
[49] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.
[50] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. 2015. Segugio: Efficient behavior-based tracking of malware-control domains in large ISP networks. In *DSN*. IEEE, 403–414.
[51] Radim Rehuurek, Petr Sojka, et al. 2011. Gensim—statistical semantics in python. *Retrieved from genism. org* (2011).
[52] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446
[53] Ronald L Rivest and Jean Vuillemin. 1976. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science* 3, 3 (1976), 371–384.
[54] Martin Roesch et al. 1999. Snort: Lightweight intrusion detection for networks.. In *Lisa*, Vol. 99. 229–238.
[55] Samuel Schüppen, Dominik Teubert, Patrick Herrmann, and Ulrike Meyer. 2018. FANCI: Feature-based Automated NXDomain Classification and Intelligence. In *USENIX Security*. 1165–1181.
[56] Gaurav Sood. 2021. *virustotal: R Client for the virustotal API*. R package version 0.2.2.
[57] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. 2013. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *CCS*. 133–144.
[58] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. 2019. Extensible android malware detection and family classification using network-flows and API-calls. In *ICCST*. IEEE, 1–8.
[59] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* 1, 2 (1972), 146–160.
[60] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. 2012. Botfinder: Finding bots in network traffic without deep packet inspection. In *CoNEXT*. 349–360.
[61] Michael Carl Tschantz, Sadia Afroz, Vern Paxson, et al. 2016. Sok: Towards grounding censorship circumvention in empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 914–933.

[62] Tobias Urban, Dennis Tatang, Thorsten Holz, and Norbert Pohlmann. 2018. Towards understanding privacy implications of adware and potentially unwanted programs. In *ESORICS*. Springer, 449–469.

[63] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[64] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking graph-based classification via manipulating the graph structure. In *CCS*. 2023–2040.

[65] Junnan Wang, Liu Qixu, Wu Di, Ying Dong, and Xiang Cui. 2021. Crafting Adversarial Example to Bypass Flow-&ML-based Botnet Detector via RL. In *24th International Symposium on Research in Attacks, Intrusions and Defenses*. 193–204.

[66] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, Vol. 28.

[67] Nigel Williams, Sebastian Zander, and Grenville Armitage. 2006. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM* 36, 5 (2006), 5–16.

[68] James Wyke. 2016. The ZeroAccess rootkit. https://nakedsecurity.sophos.com/zeroaccess4/ Accessed March 20, 2022.

[69] Zhixing Xu, Sayak Ray, Pramod Subramanyan, and Sharad Malik. 2017. Malware detection using machine learning based analysis of virtual memory access patterns. In *DATE, 2017*. IEEE, 169–174.

[70] Simon Conant Yaron Samuel. 2017. Ewind – Adware in Applications' Clothing. https://unit42.paloaltonetworks.com/unit42-ewind-adware-applications-clothing/ Accessed January 22, 2022.

[71] Yanfang Ye, Tao Li, Donald A. Adjeroh, and S. Sitharama Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* 50, 3 (2017), 41:1–41:40.

[72] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* 6, 1 (2019), 1–23.

[73] Futai Zou, Siyu Zhang, Weixiong Rao, and Ping Yi. 2015. Detecting malware based on DNS graph mining. *International Journal of Distributed Sensor Networks* 11, 10 (2015), 102687.

A MACHINE LEARNING ALGORITHMS

We compared seven common algorithms: Logistic Regression, Support Vector Machine (SVM), Naive Bayes, Artificial Neural Network (ANN), k-nearest neighbours (k-NN), Decision Tree and Random Forest ensemble. We used an implementation of Scikit-learn for all algorithms except ANN, which uses Keras. All also used the default hyperparameters. The experimental results are shown in Table 5.

Table 5: Classification Result of 7 algorithms

	Precision(%)	Recall(%)	FPR(%)
Logistic Regression	97.8118	98.0980	0.2417
SVM	97.4046	93.3431	0.2740
Naive Bayes	92.7736	98.6101	0.8461
ANN	95.9901	98.9759	0.1653
k-NN	99.8524	98.9759	0.0161
Decision Tree	99.3426	99.4879	0.0725
Random Forest	99.9805	99.9221	0.0045

B PERFORMANCE METRICS

We define True Positive (TP) as malicious connections predicted as malicious and False Positive (FP) as benign connections predicted as malicious. True Negative (TN) represents benign connections classified correctly and False Negative (FN) represents malicious connections classified incorrectly. We use the following metrics to evaluate the detection performance: (i) precision, (ii) recall and (iii) false-positive rates (FPR).

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

C SAMPLES FOR MANUAL ANALYSIS

Table 6 presents information on the 30 samples analysed manually.

Table 6: Families and Md5 of Malware Samples.

Family	Related md5
Adware	ab775c62c8d03b33f9d9b60e013d54f5
	3eeace60ad9f357dc8b77981465381c3
	3185657ca1707f2364f34fea46afc455
	75e3d279cc20419f2af57975f755614c
	04436941856e5a8c3296b55292c47636
Wacatac	fc92be6f976b598cf65a241c8520cfd6
	f18ac8264e592a4949c5fe09979234be
	e8766491e8c8a9fc79e86197b1a55a76
	d5bea7ed4ffc27366e218a99d5709987
	fe3bea4366ecc34ddbf90291762cba88
	d63d7bceed0da682db6170c24663b3b0
	df5ed0925bfb4e141a134bfdf4b2e0ce
	d32e7100988f924a0070418051de053f
Minerd	d31ca0d08a4bc600c51ecd6e891551eb
	331dfc88f7d056b2667875199eb2d504
CobaltStrike	332265a774e2ad113cbf4d05189d2ee0
	363eddcc28509a08c039833a9b6e2a04
	79a4c854d00928024f9ce3020a041451
Flystudio	67b4843d49d60e16372160cc10f80cf8
gamhak	c6752ffeb1ebc6bba468a71c36e69c85
krypyik	01372c76417280c2c7a524edd268d5a0
	ceb684549a97dae140df9e4cef10c308
hoax	464852e25233ece2e3ed769e727f3ef3
Rbot	e90fb38bd6c50f517d6d1fc00b445f91
avaddoncrypt	33874816e3eb31b874e1301fcf73bb72
lockscreen	4cf1bf8ebf10d596f7ecbb1c24258eef
graftor	71fb3ed4bf17e328c045a062fbf0895e
csdi	19435957f4a3d1380bfcd4c087e40a93
	cc07157af9a75f492748baed0d22a9fe
NetWorm	97acceb1b93ace58c250901ebd55aadd

D EVALUATION RESULTS

Figure 11 visualizes the results of embedding the traffic graphs into six of these families, Figure 12 shows the experiment results of parameter sensitivity, and Figure 13 shows malware family classification results of ETA and FS-Net in `EncMal2021`.

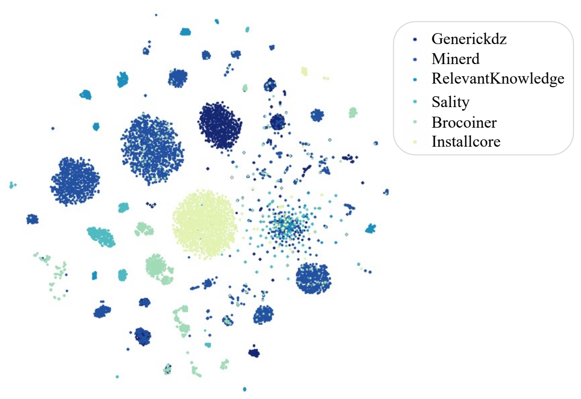
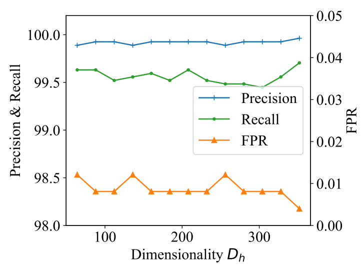
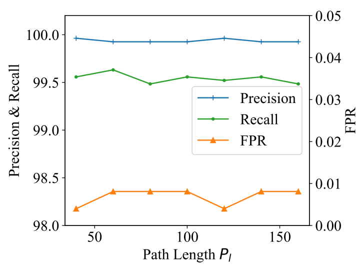


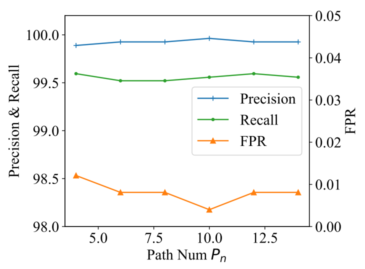
Figure 11: Visualization of the 6 malware families.



(a) Dimensionality of host vectors D_h

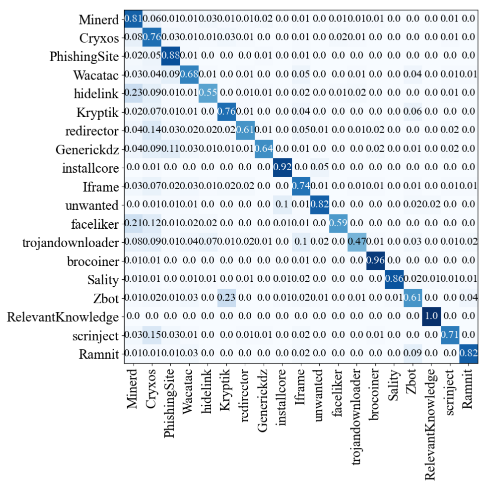


(b) Random walk path length P_L

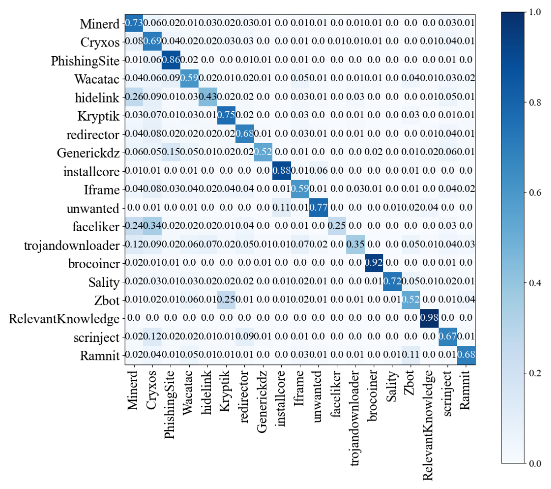


(c) Random walk path nums per edge P_N

Figure 12: Parameter sensitivity results.



(a) Results of ETA.



(b) Results of FS-Net.

Figure 13: Malware family classification result of ETA and FS-Net.

E THE STREAM ATTRIBUTES

Table 7 shows the single-stream attributes used in our model (See §5) and their descriptions.

Table 7: The Stream Attributes.

Attribute Type	Attribute	Description
Domain features	Domain Length	Total number of characters in the domain.
	Domain Level	Level of domain. Sub, second-level or top-level.
	Punctuation/Upperletter/ Lowerletter/Digit Ratio	The proportion of different character types to all characters.
	Vowel/Consonant Ratio	The proportion of vowel/consonant letters to all letters.
TLS handshake features	Client/Server TLS version	The TLS version selected by the client and server.
	Number of Client Ciphersuites	The number of the offered cipher suites by the client.
	Client Ciphersuite x	The xth in the list of cipher suites provided ($x = 1, \ldots, 10$).
	Server Chosen Ciphersuite	The cipher suite chosen by the server during the connection.
	Client/Server Compression Method	The compression method selected by the client and server.
	Number of Client Extensions	The number of the client extensions.
	Number of Signature Algorithms	The number of the signature algorithms offered by client.
	Signature Algorithm x	The xth in the list of signature algorithms provided ($x = 1, \ldots, 4$).
	Number of ec Point Formats	The number of point formats offered by the client.
	Number of Elliptic Curve	The number of elliptic curves offered by the client.
Statistical features	TLS Packet Length x	Length of the xth packet in the stream ($x = 1, \ldots, 25$).
	TLS Time Interval x	The xth time interval in the stream ($x = 1, \ldots, 25$).
	Number of Packets	The number of the total packets in the stream.
	Max/Min/Average Packet Length	The max/min/average length of total packets in the stream.