MBTree: Detecting Encryption RATs Communication Using Malicious Behavior Tree

Cong Dong, Zhigang Lu*, Zelin Cui, Baoxu Liu, Kai Chen

Abstract—Network trace signature matching is one reliable approach to detect active *Remote Control Trojan, (RAT)*. Compared to statistical-based detection of malicious network traces in the face of known RATs, the signature-based method can achieve more stable performance and thus more reliability. However, with the development of encrypted technologies and disguise tricks, current methods suffer inaccurate signature descriptions and inflexible matching mechanisms. In this paper, we propose to tackle above problems by presenting *MBTree*, an approach to detect encryption RATs Command and Control (C&C) communication based on host-level network trace behavior. MBTree first models the RAT network behaviors as the malicious set by automatically building the *multiple level tree, MLTree* from distinctive network traces of each sample. Then, MBTree employs a detection algorithm to detect malicious network traces that are *similar* to any MLTrees in the malicious set. To illustrate the effectiveness of our proposed method, we adopt theoretical analysis of MBTree from the probability perspective. In addition, we have implemented MBTree to evaluate it on five datasets which are reorganized in a sophisticated manner for comprehensive assessment. The experimental results demonstrate the accurate and robust of MBTree, especially in the face of new emerging benign applications.

Index Terms—Encrypted Traffic, Malicious Traffic, Trojan Detection, Signature, Network Behavior, Command and Control.

✦

1 INTRODUCTION

NOWADAYS, modern network attacks are accomplished with advanced automatic tools. One of them used in the post-penetration procedure is the RAT. RAT is essentially a type of software providing convenience for adversaries to complete post-penetration. Typical functions of RAT include process monitor, command execute, keyboard logger, file transfer and others. After break into the victim machine, adversies rely on the sophisticated tool for further steps. Hence, there is a need to specify the malicious activities of RAT to cut off the whole kill chain and prevent potential losses. In the early stage of the RAT development, signature-based methods can accurately detect malicious network activities with distinctive string patterns. However, since traffic encryption is widely used in different malware including RAT to acquire longer life cycles, the traditional network detection method is greatly challenged.

To tackle the problem brought by encryption technology, novel detection approaches are proposed. Generally, the art methods can be distributed into two main categories according to the detection technology. 1) Statistical-based [1], [2], [3], [4], [5], [6], [7], [8], [9], which relies on the machine learning or deep learning to learn the classification boundary among different traffic types. It follows the procedure that first produces traffic features like bytes count or interval gap means of packets, and then applies models to train and predict based on the distilled information. 2) Signature-based [10], [11], which relies on abstract behaviors

of the encrypted traffic used as identification signatures. It first formulates distinctive elements of the traffic as the signature set, and then check if the monitored traffic implies any appointed signature.

In recent years, the computer security community mainly follows the first technical route to utilize the powerful learning ability of machine learning models to tackle the problem. However, this approach is not perfect. First and most important, statistical models perform unstable in different environments. The learning strategy of most machine learning models requires not only malicious traffic but also benign traffic in training procedure [12]. Since the benign applications vary in different environments, the model will be confused by the unknown application network traces not appeared in the training procedure; thus the trained model will perform unstable when transferred into a different environment. Second, the method also requires sufficient labeled instances in the training procedure. To meet this requirement, a large amount of data should be collected in advance, which dramatically increases the inconvenience of applying the method in a real environment. Considering the shortage of statistical-based approaches, hence, we focus on signature-based routine. Compared to the former technic, the signature-based approaches can achieve robust performance through different environments and thus more reliable in a real application. Besides, the signature-based approach requires only a few malicious network traces, which can release the tedious work of data gathering.

1.1 Problems

Traditional signatures, like string patterns, can hardly adapt to encryption context because the payload is unknown during data transmission. Motivated by the observation that applications usually follow a fixed-code procedure to

---

Cong Dong, Zhigang Lu, Zelin Cui, Baoxu Liu and Kai Chen are with Institute of Information Engineering, Chinese Academy of Sciences, and School of Cyber Security, University of Chinese Academy of Sciences, e-mail: ({dongcong, luzhigang, cuizelin, liubaoxu, chenkai}@iie.ac.cn).

initiate or respond to requests, the snippets of *directed packet payload size (DirPiz)* sequence of a flow in arbitrary position is used by several arts as fingerprints to identify encrypted behaviors [11], [10]. However, these proposed methods are not able to detect RAT in an accurate manner. On one hand, false positives can occur due to the Dirpiz conflicts. Potentially, the snippets of flow-level DirPiz sequences generated from a benign application can be the same as that from RATs, thereby causing false alarms. In another word, only flow-level DirPiz snippets are not precise enough as malicious signatures. On the other hand, false negatives can occur because the DirPiz varies slightly from different environments due to *dynamically generated packets*. Generally, the size of packets carrying the instruction which asks the trapped machine to report the base information is the same, yet the size of corresponding callback packets is different. Unfortunately, existing arts can not properly handle the dynamic packets because all of them rely on the exact matching mechanism of the regex engine.

1.2 Our Work

In this paper, we present *Malicious traces Behavior Tree (MBTree)*, a novel signature-based approach for robust and accurate encrypted C&C identification based on network behavior. First, MBTree creates signatures to depict malicious behaviors by integrating flow-level DirPiz sequences as a synthesis of host-level *Multi-Level Tree (MLTree)* to improve the unique degree of malicious descriptions. In such a way, approximately 94% false alarms triggered by only a few coincidences DirPiz can be reduced. Second, MBTree relies on a flexible similarity-based matching mechanism expanding the coverage of each signature to facilitate robust detection. The proposed mechanism can cover reasonable deviations from the generated signatures. Besides, the alarm level of the detection can be adjusted by a predefined threshold.

To demonstrate the effectiveness of our approach, we conduct solid experiments on several datasets. These datasets are reorganized in a sophisticated manner to simulate the situation that the test set contains *unknown applications* in a real environment. To acquire stable results, we also integrate the 5-fold cross-validation strategy. The experiment results show that MBTree yields a more accurate performance than machine learning state-of-the-arts on the test sets. Individually, MBTree can achieve approximately 94% F1-score on validation set, and 91% F1-score on test set. Moreover, we also analyze the influence of different hyperparameters of them by tuning them and inspect the malicious behaviors by reviewing generated MLTree signatures.

1.3 Contributions

In summary, this paper makes the following contributions:

- First, a sharp network behavior representation method, MLTree, is proposed as the enhancement signature. It helps to reduce the probability of DirPiz conflicts by integrating multiple related flow-level DirPiz sequences; thus, it can depict encrypted malicious network traces in a more accurate and robust manner.

- Second, a corresponding detection mechanism based on the similarity comparison of MLTree is proposed. This strategy enhances the flexibility of detection by covering deviations slightly different from generated signatures. Compared to the exact matching strategy, it can handle the dynamic packets properly; thus, more robust detection can be achieved.

- Third, we demonstrate the effectiveness of our approach from both theoretical and experimental aspects. From theoretical perspective, we illustrate that it is in an extremely low probability that MBTree would misclassify different applications. From experimental perspective, persuasive experiments are conducted on different datasets to evaluate the real performance.

The remainder of this paper is structured as follows. Section II summarizes related works and their limitations. Section III gives an overview of the design of MBTree. Section IV elaborates the details of MLTree. Section V describes the similarity matching mechanism. Section VI provides the theoretical analysis of MBTree. Section VII covers the experiment setup description. Section VIII reports the experiment evaluation results and analysis. Section IX discusses potential evading strategies to attack MBTree. And section X provides the conclusion.

2 RELATED WORK

MBTree makes contributions to the problem of encryption RAT C&C traffic detection. A central idea behind MBTree is adopting the MLTree as signatures. Below, we discuss related works in the above areas from two perspectives.

2.1 Statistical-based

In recent decades, statistical-based methods for RAT traffic detection have been widely studied [1], [2], [3], [4], [5], [6], [7], [8], [9], [13], [14], [15], [16], [17], [18]. Generally, most of these methods follow such procedures that first extract features from the traffic, and then using statistical models to fit the data.

Recently, [2], [19], [5], [8] use side-channel information with Random Forest (RF) for encrypted malicious traffic detection. Specifically, these side-channel features range from packet length, packet interval time to payload ratio. The experiment results have shown the effectiveness of this combination of side-channel features and machine learning models. Besides, advanced deep learning models are also adopted in this area [4], [1], [20], [21], [22], [23]. Compared with traditional machine learning models, these models do not require the feature extraction procedure. The sophisticated models can automatically extract related features, and achieve end-to-end classification. For example, [4] proposes a deep learning method to detect HTTP malicious traffic on mobile networks. [1] proposes a sophisticated deep learning architecture to detect trojans with hierarchy spatiotemporal features. Generally, these traditional machine learning methods and advanced deep learning methods can achieve high performance in their experiments. However, they can hardly adapt to the situation of unseen applications beyond the training set due to the inconsistent statistical distribution.

TABLE 1
Comparison of different signature-based methods.

Method	Signature Generation	Main Element of Signature	Level	Type	Scene	Encrypted Detection	Detection Engine
Snort	Manual	String Pattern	packet/flow	Exact	IDS	No	Self
Suricata	Manual	String Pattern	packet/flow	Exact	IDS	No	Self
Zeek(Bro)	Manual	String Pattern	packet/flow	Exact	IDS	No	Self
	Automatic	String Pattern	packet/flow	Exact	Malware	No	Snort
	Semi-Automatic	String Pattern	flow	Exact	Malware	No	Snort
OTTer	Automatic	Packet Length	flow	Exact	Application	Yes	Self
PingPong	Automatic	Packet Length	packet pair	Exact	IOT Application	Yes	Self
MBTree	*Automatic*	*Packet Length*	*host*	*Similarity*	*Malware*	*Yes*	*Self*

Fig. 1. High-level overview of MBTree

Moreover, they both lack the interpretability to trace back the cause of alarms for further response.

2.2 Signature-based

Different from statistical-based methods, signature-based methods rely on previously defined elements of the traffic, mainly on specific content string patterns. Generally, most of them first create a signature set and then determine if the testing instances match one or more signatures in the set. Compared with machine learning methods, signature-based methods are more robust among different environments; thus, it is still applied in multiple security products, ranging from intrusion detection systems (IDS) to firewalls [24].

Despite the advance, there exist two apparent limitations of traditional signature methods. First, most methods require manual work to produce high-quality signatures even though given the cleaned malicious traffic. This manual work is tedious for security analysts. Second, traditional signatures like specific strings are seriously challenged by encrypted traffic [25], [26], [27]. Because the string patterns are invalid to ciphertext. Several studies focus on the automatic generation of network signatures to tackle the first problem [28], [29], [30], [31]. For example, [28] propose three automatically generated signatures ranging from conjunction signatures, token-subsequence signatures to Bayes signatures for polymorphic worm detection with traffic payload. [29] uses clustered malicious traces to produce network-level signatures from HTTP fields automatically. These network-level signatures are translated in a format compatible with Snort rules and can be used for detection. Apart from only focusing on the first problem, recent studies focus on tackling both problems by introducing novel behavior patterns. Instead of using specific strings as sig-

natures, these novel methods use other aspect information to identify specific behaviors, like packet size sequence. For example, [10] propose an encrypted traffic pattern language based on *packet payload size sequence* for scalable Over-The-Top (OTT) applications identification. They show that this signature is unique to identify applications or even application events. Similarly, [11] proposes a method based on packet length pairs to identify specific events of home IoT devices. To achieve the fine-grained event-level traffic detection, [11] adopt the DBSCAN algorithm to search frequent conversation pairs, and then concatenate the pairs into sequences as signatures.

Even though previous studies present the effectiveness of novel network signatures, they still lack the ability to detect encrypted RAT in an accurate and robust manner. This can be attributed to the inaccurate signature building and inflexible matching mechanism. Since several RATs can establish different connections to evade the detection [32], [33], only flow-level fingerprints can hardly cover the whole picture of the encryption RAT behaviors. Besides, although the exact matching strategy can cooperate with the finite state machine to improve efficiency, it lacks the flexibility to capture similar behaviors among different environments. To clearly show the path of research in this direction, we summarize the differences of each method in Table 1.

3 MBTREE OVERVIEW

In this section, we provide an overview of the proposed MBTree system. As shown in Fig. 1, our approach consists of two main procedures, *host behavior formulation* and *detection*. First, the raw traffic is formulated to MLTree representing host-level behaviors in three steps. 1). The preprocessing achieves traffic cleaning and session reassembling. 2). The

Quasar Traffic

[68, -228, 68, 68]
[68, -228, 68, 136]
[68, -228, 68, 136]

Head Sequence

(-68, 516, -68, -84)
(-68, 516, -68, 292)
(-68, 276, -68, -68)

Tail Sequence

[68, 3] → [-228, 3] → [68, 3] → [68, 1]
　　　　　　　　　　　　　　　[136, 2]

Head MLTree

(-68, 3) → (516, 2) / (276, 1) → (-68, 3) → (-84, 1) / (292, 1) / (-68, 1)

Tail MLTree

Testing Traffic

[68, -228, 68, 68]
[68, -228, 68, 182]
[3, -60, 40, 59]

Head Sequence

(-68, 516, -68, 116)
(-68, 84, 84, -68)
(176, 1440, 224, 1440)

Tail Sequence

[68, 2] → [-228, 2] → [68, 2] → [68, 1]
　　　　　　　　　　　　　　　[182, 1]
[-3, 1] → [-60, 1] → [40, 1] → [59, 1]

Head MLTree

(-68, 2) → (516, 1) → (-68, 1) → (116, 1)
　　　　　(84, 1) → (84, 1) → (-68, 1)
(176, 1) → (1440, 1) → (224, 1) → (1440, 1)

Tail MLTree

(a) DirPiz Sequence　　　　　　　　　　　(b) MLTree

Fig. 2. A brief example shows the simplified DirPiz sequences of each session with setting L as 4. The training set contains the malicious traffic generated from Quasar. The testing set contains the mixed traffic generated from Quasar and WhatsApp. Positive numbers denote the packet sent from the client, while negative numbers denote the packet sent from the server.

DirPiz sequences from each session as multiple independent fingerprints. 3). The sequences are correlated based on common hosts to construct MLTree as the host behavior. It is worth mentioning that malicious signatures and testing instances are produced following the same steps in this procedure. Second, the converted testing instances are compared to each signature to decide if they match any malicious behaviors in two steps. 1). The similarity between the testing instance and each signature is calculated to produce a similarity vector. 2) whether the instance belongs to malicious is predicted based on the similarity vector.

To show the workflow clearly, we also provide a simplified example throughout the following sections to describe the key design of MBTree. The example uses a part of pure Quasar traffic as malicious traffic, and a part of mixed Quasar traffic and WhatsApp traffic as testing traffic, as shown in Fig. 2.

4 HOST BEHAVIOR FORMULATION

In this section, we describe the details of the formulation procedure from raw traffic to the host-level signature MLTree. Section 4.1 introduces the cleaning and session reassemble strategy, Section 4.2 shows our consideration of DirPiz extraction, and Section 4.3 provides the definition of MLTree and corresponding construction process.

4.1 Preprocessing

Raw traffic is usually captured in pcap or pcapng format recording a lot of communication details. Extra meta information should be removed before building the signature. Hence, we first apply traffic cleaning to discard invalid packets, then we apply session reassembling to recover the whole communication.

Cleaning; Traffic cleaning aims at filtering out redundant information in raw traffic. As the requirements of cleaned traffic for signature and testing instance formulation are different, we apply different strategies for the two sets.

For the training set, we apply a 'whitelist' strategy to only reserve the packets containing C&C IPs. For other traffic in the malicious communication, we just discard them. Thus, only ensured communications between the victim and C&C are selected to produce valid signatures. For the testing set, we apply a 'blacklist' strategy to only discard packets that meet the following conditions, repeated packets, loop packets, non-transmission packets. With this strategy, we can reserve the main information of testing traffic and avoid potential malicious packets bypassing.

Session reassembling; After cleaned, the traffic is reassembled into sessions to recover all end-to-end communication. In this procedure, we mainly identify a transmission session based on 5-tuple. The 5-tuple includes source IP, destination IP, source port, destination port, and protocol. Besides, for TCP protocol, the flag fields representing the communication status are also used to identify different sessions using the same 5-tuple.

4.2 DirPiz Sequence Extraction

Given a reassembled session, we extract the DirPiz sequence of the session to fingerprint the automated procedure. As a matter of fact, multiple types of meta-information of the packet can be extracted as fingerprints, such as packet gap intervals, and packet payload hash. However, they either lack stable performance in different network environments or do not adapt to variable encrypted content because of the dynamic encryption key negotiation mechanism. As a result, we choose the DirPiz as the meta information for its robust performance. Typically, a DirPiz sequence consists of the payload size of each packet with the direction in a connection. The direction means that the packet is either request from the client to the server or a response from the server to the client. Here we provide an example of extracted DirPiz sequences in Fig. 2(a). The Quasar traffic is used as a training set. While the mixed traffic of Quasar and WhatsApp is used as a testing set.

Then we provide the specific procedures to produce a DirPiz sequence. First, different IP packets are reassembled if the packet is fragmented. Since the communication content can be scattered in multiple IP packets due to the limitation of MTU or potential IP fragment attack, reassembling these fragmented IP packets can restore the real payload in a communication. Second, based on these reassembled IP packets, the payload sizes are counted according to the upper-level protocols to form a sequence. In this step, the influence of low-level protocol details can be reduced, e.g., TCP handshake or SSL/TLS negotiation. Different strategies are applied according to different protocols. For UDP protocol, the length of the transport payload is used directly. For TCP protocol, only the length of the transport payload after the connection is established is used. For SSL/TLS protocol, only the length of the payload in the 'Application Data' packet is used. Third, we append the direction sign to the elements of payload size sequences. The request information from a client to the server is formulated as a positive number, and the response information from a server to the client is formulated as a negative number. Fourth, all sequences are aligned to the same length to keep consistent. Only L *DirPiz* are reserved in the sequence. Besides, for sequences less than L in length, 0 is used as padding value. For example, Fig. 2(a) shows the extracted DirPiz sequences with setting L as four.

4.3 MLTree Construction

Algorithm 1 MLTree Initialization

Require: DirPiz Sequence Set P, Max Level L
Ensure: MLTree T
 1: Let $T = (N, E, C_N, C_E)$
 2: Let $N = \varnothing, E = \varnothing, C_N = \varnothing, C_E = \varnothing$
 3: **for all** level $l \in L$ **do**
 4:    $N^l = \varnothing, C_N^l = \varnothing, E^l = \varnothing, E_N^l = \varnothing$
 5:    **for all** DirPiz sequence $p \in P$ **do**
 6:       $N^l = N^l \cup \{p[l]\}, C_N^l[p[l]]+ = 1$
 7:       **if** l eq 0 **then**
 8:          $e = (0, p[l]), E^l = E^l \cup \{e\}, C_E^l[e]+ = 1$
 9:       **else**
10:          $e = (p[l-1], p[l]), E^l = E^l \cup \{e\}, C_E^l[e]+ = 1$
11:       **end if**
12:    **end for**
13:    Append N^l, C_N^l, E^l, C_E^l to N, C, C_N, C_E
14: **end for**
15: **return** T

In this section, we integrate multiple diverse DirPiz sequences into MLTree as the host signature. Corresponde to head sequences and tail sequences, two MLTrees are used to represent the host signature, specifically, a head MLTree and a tail MLTree. As a central structure of our approach, MLTree is defined as follows,

Definition 1. **MLTree** MLTree T is a Weighted Directed Acyclic Graph (WDAG), $T = (N, E, C_N, C_E)$, where N, E, C_N, C_E represent the node set, edge set, node weight set, edge weight set respectively. The node set N is used to represent unique DirPiz grouped by level. The statistical set C_N is used to record the aggregated

occurrence of each unique DirPiz organized in level. The edge set E is used to represent two co-occurrence DirPiz grouped by two adjacent levels. The statistical set C_E is used to record the co-occurrences of each two adjacent DirPiz. With different to normal WDAG, MLTree is organized in hierarchy structure, every weighted node n_i^l and weighted edge e_i^l belongs to a specific level sub-set, $n_i^l \in N^l, e_i^l \in E^l$. And all these level sub-sets N^l, E^l, C_N^l, C_E^l consist of the corresponding elements, $N = \{N^l\}_{l=0..L}, E = \{E^l\}_{l=0..L}, C_N = \{C_N^l\}_{l=0..L}, C_E = \{C_E^l\}_{l=0..L}$.

Algorithm 2 MLTree Merging

Require: MLTree Set M
Ensure: Merged MLTree T
 1: Let $T = (N, E, C_N, C_E)$
 2: Let $N = \varnothing, E = \varnothing, C_N = \varnothing, C_E = \varnothing$
 3: **for all** level $l \in L$ **do**
 4:    $N^l = \varnothing, C_N^l = \varnothing, E^l = \varnothing, E_N^l = \varnothing$
 5:    **for all** MLTree $m \in M$ **do**
 6:       $N^l = N^l \cup m.N^l, E^l = E^l \cup m.E^l$
 7:       **for all** node $n \in N^l$, edge $e \in E^l$ **do**
 8:          $C_N^l[n]+ = m.C_N^l[n], C_E^l[e]+ = m.C_E^l[e]$
 9:       **end for**
10:    **end for**
11:    Append N^l, C_N^l, E^l, C_E^l to N, C, C_N, C_E
12: **end for**
13: **return** T

The specific MLTree construction process is as follows. First, the DirPiz sequences are processed level by level as the initial MLTree. In each level of processing, the unique nodes, edges, and their corresponding statistics are specified. The specific initialization schema is shown in Algorithm 1. Second, merging operations are taken to enhance the coverage as well as reduce the storage cost of repeated patterns. The specific merging algorithm is shown in Algorithm 2. Intuitively, it contains *polluting risk* when integrating a large number of individual MLTrees into a huge MLTree. Because the DirPiz coverage of the large tree will increase dramatically resulting in high-level false alarms. We divide this kind of situation into two categories for discussion. 1) All initial MLTrees are from the same RAT. In this situation, all these individual MLTrees denote similar DirPiz coverage, thus they can be merged without worrying about polluting risk. 2) Initial MLTrees are from different RAT. In this situation, there is a high polluting risk in merging these individual MLTrees because their DirPiz coverage is different. To avoid this, we limit the merging operation to integrate the DirPiz sequences generated from the same malicious sample. Thus, similar behaviors can be merged regardless of the number of DirPiz sequences, and the scale of the tree can be limited to a proper extent. Example MLTrees built from Fig. 2(a). is shown in Fig 2(b). It is apparent that MLTree is hierarchically organized.

Briefly, MLTree provides several advantages to depict malicious behaviors. First, MLTree ensures flexible merging to represent signatures efficiently. In our design, different MLTrees derived from the same malicious sample can be merged into a single MLTree. This feature can enhance

the signature update ability as well as reduce the cost of storing repeated signatures. Second, the hierarchy design can represent the host behavior in an intuitive and reasonable manner. Since the frequent DirPiz is with a higher probability of being generated automatically, the automated handshake behavior can be quantified by differentiating between frequent DirPizs and infrequent DirPizs hierarchically. Third, MLTree can be automatically constructed given the cleaned malicious traffic. Both the MLTree construction and merging can be completed with a systematic script. In such a manner, the construction requires no interactions with security experts and prior knowledge. Hence the labor costs can be reduced.

5 DETECTION

Unlike traditional signature-based detection, our detection relies on similarity matching strategy instead of exact matching strategy to decide if the testing instance should be regarded as malicious. Brief steps of detection are as follows. First, the similarity vector of the testing instance and each signature is calculated as $v_m = [s_0, s_1, ..., s_n]$. The element in the vector represents the similarity score of the testing instance and the corresponding signature. The specific similarity score calculation method is elaborated in Section 5.1. Second, we make a prediction based on the similarity vector to decide if the testing instance contains malicious behavior. The details of this step are shown in Section 5.2.

5.1 Similarity Calculation

The first step towards malicious prediction is to produce the similarity vector of the testing instance with pre-produced MLTree signatures. In this step, the core is to calculate the similarity of two MLTrees. Roughly, the MLTree similarity is measured from two aspects, *path similarity* and *node similarity*. The path similarity measures the similarity of continuous edges in corresponding hierarchies, while the node similarity measures the similarity of nodes in corresponding hierarchies. Specifically, a similarity score S of the testing instance and a signature is formulated as

$$S = \beta[\alpha S_{fp} + (1-\alpha)S_{fn}] + (1-\beta)[\alpha S_{lp} + (1-\alpha)S_{ln}] \quad (1)$$

where S_{fp} and S_{fn} represent the path similarity and the node similarity of corresponding head MLTree respectively, S_{lp} and S_{ln} represent the path similarity and the node similarity of corresponding tail MLTree respectively, α represents the path score ratio parameter that determines the balance of the path score and node score, and β represents the head ratio parameter that determines the balance of the head score and tail score. In addition, special DirPiz that appear frequently in both benign and malicious MLTrees can be removed in calculating the similarity score. Thus, identical DirPizs can have a more important influence in similarity calculation.

5.1.1 Path Similarity

Path similarity is used to measure the common continuous paths of two MLTrees. Towards achieving this measurement, we first define *common weighted path (CWP)* as follows,

Definition 2. **Common Weighted Path** Given two MLTrees $bt = \{N_t, E_t, C_{Nt}, C_{Et}\}$, $bm = \{N_m, E_m, C_{Nm}, C_{Em}\}$, a CWP P_C is defined as the intersection of continuous weighted edges of two MBTrees. Specifically, $P_C = \{E_C, C_{Ec}\}$, the edge set E_I is the continuous intersection of the two edge sets E_t and E_m. This means $\forall 1 < l < L$, if $(n_i^l, n_j^{l+1}) \in E_I^l$, then $(n_i^l, n_j^{l+1}) \in (E_t \cap E_m)$, and there exists at least one edge (n_p^{l-1}, n_i^l) that $(n_p^{l-1}, n_i^l) \in E_I^{l-1}$. Besides, the statistics set C_{Ec} based on E_C is also the intersection of the two statistics sets according to different levels.

Generally, CWP aims at capturing the common successive sequential information of two MLTrees. Thus, it is used as the middleware to measure the continuous similarity. The algorithm to generate CWP of two MLTrees is shown in Algorithm 3.

Algorithm 3 CWP Generation

Require: Testing MLTree $bt = \{N_t, E_t, C_{Nt}, C_{Et}\}$, and signature MLTree $bm = \{N_m, E_m, C_{Nm}, C_{Em}\}$, Max Level L.
Ensure: Common weighted path P_C
1: Let $P_c = \{E_c = \varnothing, C_{Ec} = \varnothing\}$
2: **for all** level $l \in L$ **do**
3:    **if** l is not 0 and E_c^{l-1} is $\varnothing$ **then**
4:       **return** P_c
5:    **end if**
6:    Let $E_c^l = \varnothing, C_{Ec}^l = \varnothing, N_{tmp}^l = N_t^l \cap N_m^l$
7:    **if** l is not 0 **then**
8:       $E_c^l = E_t^l \cap E_m^l \cap (N_{tmp}^l \times N_{tmp}^{l-1})$
9:    **else**
10:      $E_c^l = E_t^l \cap E_m^l$
11:    **end if**
12:    **for all** edge $e \in E_c^l$ **do**
13:      $C_{Ec}^l = C_{Em}^l[e]$
14:    **end for**
15: **end for**
16: **return** P_c

Then we provide several observations of CWP. 1) Edge statistics of CWP can reflect the similar level of two MLTrees. 2) Range of edge statistics at the same level in different CWP is inconsistent, because the testing instance may contain the traffic generated from a different period than signature traffic. 3) Range of edge statistics at different levels in CWP is inconsistent because the edge statistics are generated independently. 4) It is of low probability to produce a long CWP between two unrelated MLTree. As a matter of fact, two unrelated MLTree may contain the same DirPiz at a level; however, they can hardly contain the same edges or even continuous edges through different levels. Thus, a long CWP can indicate a higher level of similarity than a short CWP.

Next, we synthesize the observations of CWP and provide several considerations to design the formula for path similarity. 1) A higher similarity score should indicate more similar the two MLTrees are. To achieve this, we design a weighted product mechanism to ensure that the score increases monotonically with the increase of the number of edges and levels. 2) Edge statistics should be normalized to

Common Weighted Path

Head	Tail
{[68, -228], 2}, {[-228, 68], 2}, {[68, 68], 1}	*{(-68, 516), 1}, {(516, -68),1}, {(68, 68), 1}*

Common Nodes

Head	Tail
{[68], 2}, {[-228], 2}, {[68], 2}, {[68], 1}	*{(-68), 2}, {(516),1}, {(68), 1}, {(68), 1}*

Fig. 3. CWP and Common Nodes of corresponding MLTrees in Fig. 2(b).

be used as a base factor of the score. Specifically, we use the edge statistics of signatures at the corresponding level to normalize that of testing instance; thus, the impact of the difference among levels can be eliminated. Besides, we also introduce a time ratio to balance the period difference between the testing instance and the signature. 3) Hierarchy level can be used as a weight factor for edge statistics. To leverage the continuous property of CWP, we assign different weights to statistics at different levels. Thus a longer CWP can correspond to higher similarity. 4) the statistics should be normalized to reduce the influence of different time lengths.

Based on these considerations, we tried several schemes and performed experiments on a small training set. Finally, we found the best result is achieved by the formula as follows,

$$S_p = 2^{L' + \sum\limits_{l=1}^{L'} \left[\frac{F(E_C^l, C_{Em}^l)}{F(E_m^l, C_{Em}^l)} \times \frac{l^2}{L'} \times R_t \right]} \quad (2)$$

where L' denotes the max level of CWP, $\frac{l^2}{L'}$ denotes the *level important factor*, the $\frac{F(E_C^l, C_{Ec}^l)}{F(E_m^l, C_{Em}^l)}$ denotes the *normalized level path similarity factor*, $F(a, b)$ represents *counting the total occurrence of all elements of a in set b*, $R_t = \frac{T_m}{T_t}$ denotes *the time ratio for normalization*, the T_m represents the capturing time of the signature traffic and the T_t represents the capturing time of the testing traffic. It is worth to mention that we add level L' in exponential part as an score normalization factor to differentiate the minum value with different max level L.

5.1.2 Node Similarity

Node similarity is used to measure the common nodes of two MLTrees in corresponding hierarchies. As preliminary, we define the *Common Nodes* as follows,

Definition 3. Common Nodes The common nodes N_I are defined as the intersection nodes and their corresponding minimal occurrence at each level, formally, $I_N = \{N_I, C_{NI}\}, \forall l < L \rightarrow N_I^l \subseteq (N_t^l \cap N_m^l), C_{NI}^l \subseteq (C_{Nt}^l \cap C_{Nm}^l)$.

Totally, the node similarity is calculated based on the *common nodes*. Unlike CWP, common nodes are generated independently through different levels. Thus, the nodes are treated equally to contribute to the diverse similarity of two MLTrees. Specifically, correspond to produce path similarity, the formula to calculate node similarity is as follows,

$$S_n = 2^{L + \sum\limits_{l=1}^{L} \left[\frac{F(N^l, C_N^l)}{F(N_m^l, C_{Nm}^l)} \times R_t \right]} \quad (3)$$

Similarity Vector v_m

I_m	Quasar	71.653	S_m
	koadic	32	
	Metasploit	16	
	pupy	16	

Testing Traffic ⟹   ⟹ Quasar

• • •

Fig. 4. Prediction logic. θ is set as 32.

where L denotes the max level of Common Nodes, $\frac{F(N^l, C_N^l)}{F(N_m^l, C_{Nm}^l)}$ denotes the *normalized level node similarity factor*.

Here we provide the consideration to introduce the node similarity. Unlike the path similarity measurement used to accurately depict the malicious behavior, the node similarity is considered to facilitate robust detection ability. Although the path similarity can precisely measure a similar path, it is not flexible enough to capture rough similar patterns, especially the dynamic packets. Generally, dynamic packets also exist in the handshake procedure with automated packets, like the inspection results of the victim machine transferred by the RAT client to the server. These dynamic packets can truncate the CWP for their random DirPiz. Hence, we propose the node similarity as a supplementary to path similarity to handle the dynamic packets problem.

Following the example in previous section, we provide CWP and Common Nodes of former MLTree instances in Fig. 3. Based on these, the similarity score of the example can be calculated, $S_{fp} = 69.13$, $S_{fn} = 80.63$, $S_{tp} = 34.56$, $S_{tn} = 69.12$, when setting $\alpha = 0.3$, and $\beta = 0.7$, $S = 71.65$.

5.2 Prediction

After similarity calculation, we acquire the similarity vector v_m denoting the similarity of the testing instance with each signature. Hence, we can further predict if the testing instance is malicious based on the max value S_m of the vector v_m. First, the S_m is specified. Then, if S_m exceeds a predefined threshold θ, the testing instance is regarded as containing malicious behaviors. Otherwise, the testing instance is regarded as benign. Fig. 4 shows the prediction logic with setting θ as 32. In addition, in the situation that the testing instance is regarded as malicious, the specific type of malicious behavior can also be predicted based

on the index of the malicious value I_m. It is worth to mention that the specific encryption RAT type will only be decided when S_m exceeds the threshold θ. Otherwise, it is meaningless to predict the specific RAT type since the traffic is regarded as benign.

6 THEORETICAL ANALYSIS

To demonstrate the effectiveness of our proposed methods, we provide theoretical analysis in this section. First, since the principle of MBTree is based on the uniqueness of the DirPiz sequence, we demonstrate the extremely low probability that two different applications would produce the same DirPiz sequences. Assume that the DirPiz X at a level are random variables, $P(X)$ represent the probability dense function of X, and a DirPiz sequence S is consists of several random variable $S^m = [X_0, X_1, ..., X_m]$, where $X_0, X_1, ..., X_m$ are independent. Due to the limit of MTU on the Internet, X can be only chosen from [-1500, 1500] in most cases. According to [34] and [35], X on the Internet generally obey to the β distribution. However, after a small amount of frequent DirPiz are removed, X roughly obey to the uniform distribution. That is

$$X \sim U(-1500, 1500).$$

Hence, when give two independent DirPiz sequences S^m and S'^m, the probability of their collision in n levels $P(S_n^m)$ is $C(m,n) * (3*10^{-3})^n$. Suppose that we take m as 10, the corresponding $P(S_n^{10})$ shown in Table 2. As can be seen from the table, the probability of S^{10} and S'^{10} being completely the same is only 10^{-26} in the case of $m = 10$, which is extremely low. It is worth to mention that, previous studies uses DirPiz snippets at arbitary position rather than corresponding hierarchy to identify application behavior, which increases the collision probability.

TABLE 2
The probability of different n collision in two DirPiz sequences.

n	$P(S_n^{10})$	n	$P(S_n^{10})$
1	3×10^{-2}	6	1×10^{-13}
2	4×10^{-4}	7	2×10^{-16}
3	3×10^{-6}	8	3×10^{-19}
4	1×10^{-8}	9	2×10^{-22}
5	6×10^{-11}	10	6×10^{-26}

Second, based on the collision probability, we discuss the theoretical best threshold interval. Generally, the interval is decide based on number of unique applications N_A. When N_A is ensured, the tolerable level of collisions should determined to calculate the max threshold θ. Specifically, $\theta = 2^{L+n}$, where n should satisfy the following conditions

$$\begin{cases} N_A \times P(S_n^m) \leq 1 \\ \max(n) \end{cases}.$$

For example, when $N_A = 100$, the optimal value of n is 1, and the corresponding threshold should be 2^{L+1}.

7 EVALUATION FRAMEWORK

7.1 Evaluation Data

In our experiment, three malicious parts and two benign parts of traffic are used for evaluation. The first malicious part is the open-source RATs traffic collected by ourselves, the second part is the wild Trojan traffic selected from the Stratosphere project [36], and the last part is a public open-source dataset CTU-13 [37]. While the two parts benign applications traffic used are from two open datasets, ISCX VPN2016 [15] and USTC-TFC2016 [38]. Each part is described below.

Open Source Encryption RAT (OSER); In order to hide the real identity, adopting customized OSER for attack is popular in recent years [39], [40]. Thus, the open-version RAT traffic is studied in this paper. Based on popularity, stability, and maintenance on Github, 7 OSERs are selected to generate this part of the traffic. Specifically, in the traffic generation procedure, to evaluate whether a RAT follows the same procedure for communication in different environments, we collect the traffic of two hosts, which install different systems but are infected by the same sample. The traffic generation mechanism is shown in Fig. 5. Besides, to simulate the practice usage of samples, 5 randomly chosen commands are executed on the comprised machine for each malicious session. The details of the collected traffic are shown in Table 6 in Appendix.

Wild Trojan (WT); Apart from the OSER traffic, wild trojans (from 2015 to 2018) are also selected from [36]. Compared to OSER traffic, the WT traffic contains more number sessions. However, since the communications between the victim and Trojan C&C are not controlled as detailed as OSER, it may also contain noise traffic generated by the machine automatically. Nevertheless, this part of the traffic is also a fair test to evaluate the WT detection ability of MBTree. More details of this part of the traffic are shown in Table 7 in Appendix.

CTU-13; The public open-source dataset is widely used in malware traffic detection research [41], [42]. It contains malicious traffic from 7 different types of Trojans. Apart from C&C traffic, it also collects attack traffic in other procedures, like distributed denial of service (DDoS) traffic and spam traffic. Hence, this traffic is cleaned to only reserve C&C traffic according to the dataset description. Due to the lack of valid traces for 5-fold validation, two types of Trojan (Rbot, NSIS.ay) are not used in our experiment. More details of cleaned traffic are shown in Table 8 in Appendix.

ISCX VPN 2016; In recent studies, the ISCX VPN2016 is widely used for encrypted traffic classification. In this paper, we also adopt this set as a part of benign traffic to evaluate the false alarm levels. Totally, the set organizes the 27G raw traffic generated from 17 typical applications in 150 pcap or pcapng files. Compared to former malicious parts, this benign application set is larger.

USTC-TFC 2016; Apart from ISCX VPN2016, we also use the benign part of USTC-TFC2016 as another part of benign traffic, because it contains different applications traffic from the ISCX VPN2016. Actually, this part of traffic consists of both benign and malicious traffic. Since the traffic contained in the malicious part of USTC-TFC2016 is covered by former datasets, they are not taken as malicious parts. Thus, only the benign part of the set is used for evaluation. Totally, the benign part of the USTC-TFC2016 organizes the 3.71GB traffic generated from 10 applications in 14 pcap files.

TABLE 3
This table shows the partitioning details in each dataset. The ● and ⋆ denotes the OSER traffic generated from Debian and CentOS respectively; ▷ denotes the WT traffic; ⊗ denotes the CTU-13 traffic; □ and ○ denote different applications traffic from the combined set of ISCX VPN2016 and USTC-TFC2016.

	Dataset I			Dataset II			Dataset III		
	Train	Validation	Test	Train	Validation	Test	Train	Validation	Test
Malicious	●	●	⋆	▷	▷	▷	⊗	⊗	⊗
Benign	□	□	○	□	□	○	□	□	○

Fig. 5. Traffic collection mechanism.

7.2 Data Organization

We organize the five parts of the collected traffic into three datasets for evaluation. Each dataset consists of malicious traffic and benign traffic. The OSER is used as malicious traffic in dataset I, the WT is used in dataset II, and CTU-13 is used in dataset III. The benign traffic shared by two datasets is the combination of ISCX VPN 2016 and USTC-TFC 2016. Total N_A of these two datasets is around 100.

For persuasive evaluation, we use the 5-fold cross-evaluation strategy to acquire a stable performance. Moreover, in each fold, the dataset is divided into three parts, train, validation, and test. Basically, the three parts in a fold are divided following the ratio of 0.49:0.21:0.3. It is noteworthy that since OSER is generated from two machines, we use the traffic generated from one machine as the train and validation sets respectively, and from the other machine as the test set. Besides, the benign applications that appeared in the train set do not appear in the test set. This partitioning strategy aims at simulating the situation that unknown applications that emerged in the test environment, which scenario is common in reality. The details of partition are shown in Table 3.

7.3 Baselines

In correspondence to the related studies in Section 2, six baselines are covered in this paper as comparisons. The first baseline is a machine learning state-of-the-art [19] using side-channel features and CART represented as *CART*. The second baseline is another machine learning-based state-of-the-art [2] using different features from [19] with random forest for Trickbot Trojan detection as *RF*. The third baseline is the extended features generated by CICFlowmeter [43] with GradientBoosting implemented by ourselves as *GBDT-CIC*. The fourth baseline is a deep learning method using one dimensional CNN from [21] as *CNN*. The fifth baseline is another deep learning method using a stacked autoencoder mechanism as *SAE* [21]. Besides, we also implement

the flow-level similar matching method using the cosine distance and threshold of 0.99 represented as *DirPiz-Seq* to illustrate the advantage of the host-level signatures. Moreover, we try to compare our method with ping-pong [11], which is a signature-based state-of-the-art for encrypted IOT traffic event identification. However, we find that ping-pong can hardly extract packet-level signatures of malicious traffic. Because the conversation pairs of packet-level signatures are too diverse to be clustered by the DBSCAN algorithm even though we try to tune corresponding parameters.

7.4 Tasks & Metrics

Based on the aforementioned data, we propose the detection task for experiments. The goal of the task is to distinguish whether the traffic is benign or malicious. Specifically, the malicious traffic is labeled as the specific type when is predicted. While for benign traffic, they are all regarded equally as general benign without further classifying the specific application type. In addition, we also record the prediction time of each method. As evaluation, four well-known metrics are adopted: *False Positive Ratio (also known as False Alarms Ration, FPR), False Negative Ratio (FNR)* [44], *Accuracy (Acc)*, and *Macro F1 (F1)*. It should be noted that when calculating FPR and FNR, the multi-class labels are masked to binary labels as only simple malicious or benign. While calculating Acc and F1, the multi-class labels are directly taken into consideration. We believe this mechanism can measure the performance of different models more comprehensively.

8 EXPERIMENTS

In this section, we provide a thorough evaluation of MB-Tree from five perspectives based on the aforementioned dataset. First, we perform the evaluation on the detection task to show the effectiveness of the proposed MBTree by comparing it to several machine learning-based state-of-the-arts. Second, we provide a case study to clearly show the significance of MBTree. Third, we compare the efficiency of each method. Fourth, we conduct experiments to analyze the influence of the hyperparameters and their corresponding optimal choice. Further, we show the analysis of the generated signatures to reveal the network behavior differences among different samples. Except for the fourth experiment, the hyperparameters of MBTree are set preliminary as *max level L* of 10, *path similarity ratio* α of 0.3, *head signature ratio* β of 0.7, and *threshold* θ as 2048.

8.1 Malicious Detection

In this section, we focus on presenting the overall effectiveness of MBTree. Table 4 shows the performance of different approaches on each dataset. And Fig. 6 shows the

TABLE 4
Experiment results. Highlighted values: **overall best method**, second best method (†), third best method (‡).

Dataset	Methods	Validation				Test			
		FPR	FNR	Acc	F1	FPR	FNR	Acc	F1
I	MBTree	0.21±0.21 †	0.23±0.35	99.82±0.23 ‡	99.27±0.12 †	**0.1±0.09**	1.41±1.55 †	**99.59±0.25**	**99.42±0.15**
	CART	2.49±5.05	0.03±0.07 †	99.56±2.68	95.71±11.54	26.77±6.99 ‡	1.73±2.1 ‡	74.2±31.43 ‡	66.17±8.43 ‡
	RF	24.79±19.45	0.08±0.07 ‡	99.36±0.93	74.59±20.34	33.06±16.54	3.2±1.28	57.44±21.63	57.72±21.94
	GBDT-CIC	49.25±22.02	1.59±3.11	79.2±44.25	47.37±25.22	67.16±13.82	3.85±2.2	54.02±21.56	25.39±11.93
	CNN	**0.03±0**	**0.01±0**	**99.98±0**	**99.95±0.06**	70.6±1.14	2.4±0.43	65.93±7.53	30.77±0.79
	SAE	1.37±2.98 ‡	**0.01±0**	99.97±0.03 †	98.57±3.16 ‡	67.85±3.08	2.32±0.55	68.94±9.44	33.65±3.86
	DirPiz-Seq	18.49±2.29	1.45±0.66	80.28±9.77	83.14±1.95	18.1±2.42 †	**0.99±0.2**	84.52±3.09 †	80.66±1.83 †
II	MBTree	**0.57±1.14**	**0.03±0.05**	**99.96±0.06**	92.43±1.33 ‡	**0.62±1.25**	0.84±0.91 ‡	**99.08±0.94**	**87.71±1.71**
	CART	6.05±2.18 †	0.81±0.26	95.72±1.05	93.14±1.51 †	52.66±13.2	5.43±2.23	68.06±17.96	44.17±16.88
	RF	10.32±13.01	1.4±1.5	91.62±10.19	88.01±17.97	38.67±9.98	11.77±12.91	67.68±13.00	51.93±10.99
	GBDT-CIC	3.56±1.63 †	0.57±0.51 †	96.94±0.74 †	**96.33±1.32**	38.78±11.82	6.49±3.21	65.00±15.69	52.54±9.44
	CNN	15.67±1.99	0.66±0.33 ‡	95.75±2.16 †	83.43±2.28	19.79±7.08 ‡	**0.58±0.25**	95.56±1.88 †	74.4±7.20 ‡
	SAE	15.74±2.6	0.78±0.51	95.16±3.28	83.05±3.14	19.13±6.78 †	0.59±0.41 †	95.46±3.1 ‡	75.71±9.35 †
	DirPiz-Seq	40.27±4.77	7.55±1.69	58.20±9.39	56.33±3.70	41.96±5.65	7.59±0.41	52.84±2.15	52.00±2.28
III	MBTree	**0.0±0.0**	**0.0±0.0**	99.99±0.01 †	91.11±0.0	**0.0±0.0**	0.12±0.13 ‡	**99.88±0.13**	**86.66±6.67**
	CART	0.17±0.24 †	0.22±0.35 †	99.61±0.27	93.59±4.11	0.03±0.05 †	12.29±14.53	88.95±13.39 †	79.58±11.73 †
	RF	**0.0±0.0**	**0.0±0.0**	99.92±0.06	97.6±0.67 ‡	**0.0±0.0**	79.42±13.2	21.65±12.9	65.4±8.2
	GBDT-CIC	**0.0±0.0**	**0.0±0.0**	99.93±0.05 ‡	98.35±1.1 †	0.03±0.07 †	75.82±20.42	25.17±20.07	66.49±9.1
	CNN	1.36±2.71 ‡	**0.0±0.0**	99.2±1.5	93.5±7.97	82.84±15.09	0.01±0.02 †	81.12±15.04 ‡	70.95±8.02 ‡
	SAE	**0.0±0.0**	**0.0±0.0**	**100.00±0.0**	**100.00±0.0**	87.96±8.12	**0.0±0.0**	72.11±21.1	64.46±15.05
	DirPiz-Seq	20.85±2.27	21.6±38.72 ‡	66.69±17.73	52.63±8.78	14.92±0.31	49.0±31.39	61.32±5.18	54.04±5.8

(a) Confusion matrix on Dataset I.　　　(b) Confusion matrix on Dataset II.　　　(c) Confusion matrix on Dataset III.

Fig. 6. Confusion matrices of MBTree test predictions on different datasets.

classification results of MBTree on the test part of different datasets. First, as a high-level performance comparison, although MBTree is not achieving the best performance on the validation set, it outperforms all the considered elements of contrast in the test set in most metrics. This indicates the robust ability of our proposed signature-based approach. Besides, considering differences in performance on validation and test set, it can be concluded that the signature-based methods perform more stable on different sets, including both MBTree and DirPiz-Seq. From another view, although the statistical distribution of malicious part is consistent in validation and test of dataset II and III, the statistical-based methods also perform quite differently. This illustrates that these statistical-based methods are more susceptible to the change of the environment. Hence, the robust detection ability of signature-based methods through different environments is demonstrated.

Second, It should be noted that the F1 of MBTree on dataset II and III is not reaching the level that on the dataset I. According to 6(b) and 6(c), it can be observed that most of the misclassified instances are false negatives. With a detailed analysis of these instances, we found that they consist of shorter communication sequences, which

means there are only one or two valid payloads exchanged between the client and C&C. Thus, without sufficient evidence, MBTree tends to classify them as benign. An interesting phenomenon is that other metrics are consistent in all datasets, which indicates that though there exist misclassified instances, the number of them is still at a low level. This can be attributed to that the host-level detection reduces the number of instances. Hence, even only a few misclassified instances will lead to a significant change in F1.

Third, it can be noticed that the flow-level DirPiz-Seq performs worse than MBTree. This phenomenon indicates that only utilizing the flow-level communication sequences as signatures is not precise enough. Integrating flow fingerprints to host signatures can improve the performance dramatically.

8.2 Case Study

To illustrate the advantage of MBTree, we provide a case study of *koadic* RAT. The most interesting behavior of koadic is that it simulates the browser traffic by changing source port to disguise the C&C traffic. Hence, session-based detection including most of the statistical-based methods and DirPiz-Seq can hardly capture the malicious patterns of

(a) Path Score Ratio α

(b) Head Score Ratio β

(c) Max Level L

Fig. 7. Different parameter tuning results.

(a) Max Level $L = 5$

(b) Max Level $L = 10$

(c) Max Level $L = 15$

(d) Max Level $L = 20$

Fig. 8. The tuning results of threshold θ with different Max Level L. For each L, 10 points of θ are sampled between $[2^L, 2^{L+2}]$
.

(a) Max Level $L = 5$

(b) Max Level $L = 10$

(c) Max Level $L = 15$

(d) Max Level $L = 20$

Fig. 9. DET curves with different Max Level L and threshold θ.

koadic since the content is transferred separately. However, MBTree can well handle the disguised traffic by regarding all session traffic of a host as a whole part. And the similarity-based matching strategy can accurately identify the infected hosts by synthesizing the path similarity and node similarity.

8.3 Efficiency Evaluation

TABLE 5
Prediction time of different methods for per instance.

Methods	MBTree	CART	RF	GBDT-CIC
Time (s)	10^{-5}	10^{-7}	10^{-5}	10^{-3}
Methods	CNN	SAE	DirPiz-Seq	
Time (s)	10^{-2}	10^{-3}	10^{-1}	

As a comprehensive evaluation, we also record the prediction time of each method as an efficiency comparison. By analyzing the workflow of the matching mechanism, we find that the main cost of our approach comes from the intersection operation in calculating the similarity score. Hence, we design a mechanism to realize parallel computing of similarity scores in our implementation. Specifically, for each signature, we start a process to calculate the corresponding similarity score and then collect all similarity scores for

further steps. With this mechanism, the detection time is two order of magnitudes lower than that of the ordinary. The optimized results are shown in Table 5. Apparently, MBTree has the same or even better performance than most statistical-based methods.

8.4 Parameter Tuning

In this section, we take experiments on the hyperparameters of MBTree to analyze their influences. Totally, there are four parameters, Max Level L, Scores Ratio α, β, and the Threshold θ. As adopted in previous experiments, we continue to use the default settings as start, L = 10, alpha = 0.3, beta = 0.7 and threshold = 2048. Results are shown in Fig. 7-9.

Path Score Ratio α; The score ratio α determines the balance of the path score and node score. Specifically, α represents the ratio of the node score. The results are shown in 7(a). It can be noticed that the performance rise with α increasing at the initial on all datasets. This can be mainly attributed to the existence of the dynamic DirPiz in the communication sequence, which truncates the CWP. Hence, we suggest that the α should be set lower than 0.75 under normal conditions to facilitate robust detection.

Head Score Ratio β; The parameter β determines the balance of the head score and tail score. Specifically, β represents the ratio of the head score. The results are shown

(a) Unique degree of head nodes in signatures of dataset I

(b) Unique degree of tail nodes in signatures of dataset I

(c) Unique degree of head nodes in signatures of dataset II

(d) Unique degree of tail nodes in signatures of dataset II

(e) Unique degree of head nodes in signatures of dataset III

(f) Unique degree of tail nodes in signatures of dataset III

Fig. 10. Number of unique values of nodes and edges organized hierarchically in the generated MLTree signatures.

in 7(b). Slightly, the performances rise with β increasing. This can be attributed to that the patterns of the handshake process are rather obvious. With the analysis of their traffic, it is obvious that the handshake process is sophisticated. The C&C server sends fixed instructions (e.g., initialization) to the client after establishing the TCP connection. While in the handwave process, it is rather simple. The server only sends one or two packets to notify the victim that the connection is closing, and then the rest work is handed over to the TCP level. Hence, we suggest that the β should be set greater than 0.5 without extra prior knowledge about samples.

Max Level θ; The max level θ determines how deep should two MLTree be compared. Ideally, L is supposed to be set as exactly the length of the automated handshake procedure. However, since the handshake process implementation varies from each other, it requires experiments on L to determine the most appropriate value. Besides, for each L, we choose the corresponding theoretical best θ. The results are shown in 7(c). Obviously, based on the trend of the curves, the most appropriate value of L should be lower than 15 to properly capture the automated behavior in most cases.

Threshold θ; The parameter θ determines the alarm level. In this experiment, we observe the performance of different θ by setting different max levels L. Recall that N_A is around 100. For each L, we sample 10 threshold points in the interval $[2^L, 2^{L+2}]$. The Fig. 8 shows the F1 curve, and Fig. 9 shows the Detection Error Tradeoff (DET) curve. It can be observed from Fig. 8 that the best performance is achieved around the points at [1,5], which covers the theoretical best threshold; thus this demonstrates the correctness of our theoretical analysis.

8.5 Generated Signatures

In this section, we analyze the generated signatures to inspect the behavior differences among different samples by counting the number of unique values at each level. The statistics are shown in Fig. 10. In this experiment, the max level L is set as 30, which is a relatively deeper value for the handshake procedure.

First, it is apparent that the unique number of most signatures is less than 50 through different levels. However, pupy-obfs3 shows a different trend from most of the others. The DirPizs unique degree of this RAT is relatively high at the start of the communication because obfs3 achieves the traffic obfuscation in the handshake procedure by changing the packet size distribution [45]. However, even though obfuscation technology is adopted to randomize the packet size, MBTree can also effectively identify the traffic. This can be attributed to the limited range of DirPiz after randomization. Since the randomize strategy of obfs3 is applied as *using random length of bytes to pad the rest of the packet*, the length of the padded DirPiz can be only in the interval $[raw\_content\_length, MTU]$ according to [46], [47]. Thus our designed node similarity can still cover the padded DirPiz sequences. Second, comparing the head nodes with tail nodes, it is apparent that most samples use the same packets to complete the handshake process. Thus, it can be deduced that the head patterns are more identical. This conclusion is also in accord with what we acquired in the experiment of β. Third, rough automatic handshake length can be deduced based on the change points of the curves. For example, pupy-obfs3, msf-1, Dridex, TrickBot, and neris accomplish the handshake process through 5 message exchanges.

9 DISCUSSION

In the previous section, the experiment results illustrate that even though the content is transferred through encrypted transport, MBTree still identifies malicious C&C traffic. However, sophisticated attack strategies can still be taken by adversaries to paralyze MBTree. Here we discuss the strategies that can be used against MBTree.

Disguising Attack; A potential attack strategy for MBTree is disguising malicious traffic as benign applications. When adopting this strategy, though the malicious traces can still be identified, it will *pollute* MLTree signatures and result in a large number of false alarms to cripple the MBTree. However, in order to implement such a strategy, it requires the adversaries (i) to acquire which benign application is running on the victim machine; (ii) to keep on the update of the benign applications' behaviors, which is hard to achieve in practice.

Malformed Packet; Since MBTree relies on successful payload identification, the evading techniques exploiting the protocol stack parsing procedure can be used to evade MBTree[48], [49]. For example, transfer content through RST packets. When adopting such a technique, the payload can be incorrectly reassembled by the man-in-the-middle MBTree, and the malicious patterns cannot be identified. However, implementing this strategy also requires extra protocol stack control to deal with the malformed packets correctly. Besides, traditional firewall or IDS can identified these malformed packets easily.

Obfuscation Attack; Although our experiment proves that MBTree can resist the obfuscation strategy to some extent, it will still lead to the invalidity of MBTree in the face of a highly targeted attack strategy. A potential valid evading strategy is radically splitting the padded content into several packets with random packet size. In the case of such a strategy, not only will MBTree be polluted, resulting in high-level false alarms, but the sample can also evade the detection of MBTree with unseen DirPiz sequences. However, current RATs rarely attempt to hide their DirPiz identifications to our best known. Besides, to against this radical strategy, we suggest that the entropy analysis of the DirPiz sequences can be used. Since benign applications usually follow a specific procedure to complete the handshake, the entropy of their DirPiz sequences are relatively low; thus, it is abnormal if the entropy is exceedingly high in the case of only running a few applications on the machine.

10 CONCLUSION

In this paper, we present the MBTree, a novel signature-based approach that integrates DirPiz sequences as MLTree signatures with the similarity matching mechanism to detect encrypted RAT traffic. We evaluate MBTree against several C&C traffic with comprehensive benign applications' traffic as background. The results show that MBTree can detect different malicious traffic in different environments with high-level accuracy. Briefly, there are two directions in our future work to improve MBTree. First, we plan to improve the similarity score calculation, so as to detect malicious behaviors through different hierarchies. Second, inspired by [50], we plan to improve the parallel computing ability by integrating the famous MapReduce framework [51].

APPENDIX

REFERENCES

[1] Jiang Xie, Shuhao Li, Yongzheng Zhang, Xiaochun Yun, and Jia Li. A method based on hierarchical spatiotemporal features for trojan traffic detection. In *International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2019.

TABLE 6
Selected open source RATs for generating malicious traffic. N1 denotes the session number in victim 1 as the training set. N2 denotes the session number in victim 2 as the testing set.

Platfrom	Name	Transport	N1/N2
Linux	pupy	cleartext	500/500
		obfs3	300/500
		http	100/100
		SSL	100/100
		ECM	100/100
		RSA	500/500
	Metasploit	EC4+ECPV+RC4	100/100
		websock+RSA+AES	100/100
		staged meterpreter	200/500
		stageless meterpreter	400/500
Windows	koadic	TLS	150/300
	Covenant	EKE+SSL	150/300
	Stitch	AES	250/200
	QuasarRAT	TLS	100/100
	Lime-RAT	AES	100/100

TABLE 7
An overview of the WT traffic.

Name	Time	Session Num	Host Num
TrickBot	2015.3-2018.4	152680	7
Emotet	2017.6	344850	9
Dridex	2017.3-2017.4	106427	13
Trickster	2017.6-2018.1	67301	3
Upatre	2015.10-2016.5	114921	2

[2] Ali Gezer, Gary Warner, Clifford Wilson, and Prakash Shrestha. A flow-based approach for trickbot banking trojan detection. *Computers & Security*, 84:179–192, 2019.

[3] Yixin Wu, Yuqiang Sun, Cheng Huang, Peng Jia, and Luping Liu. Session-based webshell detection using machine learning in web logs. *Security and Communication Networks*, 2019, 2019.

[4] Jia Li, Xiaochun Yun, Mao Tian, Jiang Xie, Shuhao Li, Yongzheng Zhang, and Yu Zhou. A method of http malicious traffic detection on mobile networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–8. IEEE, 2019.

[5] George Stergiopoulos, Georgia Chronopoulou, Evangelos Bitsikas, Nikolaos Tsalis, and Dimitris Gritzalis. Using side channel tcp features for real-time detection of malware connections. *Journal of Computer Security*, 27(5):507–520, 2019.

[6] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309. IEEE, 2017.

[7] Shanshan Wang, Zhenxiang Chen, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. A mobile malware detection method using behavior features in network traffic. *Journal of Network and Computer Applications*, 133:15–25, 2019.

[8] Jiyeon Kim, Yulim Shin, Eunjung Choi, et al. An intrusion detection model based on a convolutional neural network. *Journal of Multimedia Information System*, 6(4):165–172, 2019.

[9] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. Scalable test generation for trojan detection using side channel analysis. *IEEE Transactions on Information Forensics and Security (TIFS)*, 13(11):2746–2760, 2018.

[10] Eva Papadogiannaki, Constantinos Halevidis, Periklis Akritidis, and Lazaros Koromilas. Otter: A scalable high-resolution encrypted traffic identification engine. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 315–334. Springer, 2018.

[11] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-level signatures for smart home devices. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[12] Huichen Li, Xiaojun Xu, Chang Liu, Teng Ren, Kun Wu, Xuezhi Cao, Weinan Zhang, Yong Yu, and Dawn Song. A machine learning approach to prevent malicious calls over telephony networks.

TABLE 8
An overview of cleaned CTU-13 traffic.

Name	Session Num	Host Num
Neris	2771	4
Murlo	1070	2
Menti	198	2
Virut	58	2
Sogou	40	3

In *IEEE Symposium on Security and Privacy (S&P)*, pages 53–69. IEEE, 2018.

[13] Riyad Alshammari and A Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, ip addresses and payload inspection? *Computer Networks*, 55(6):1326–1350, 2011.

[14] Guang-Lu Sun, Yibo Xue, Yingfei Dong, Dongsheng Wang, and Chenglong Li. An novel hybrid method for effectively classifying encrypted traffic. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–5. IEEE, 2010.

[15] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[16] Fran Casino, Kim-Kwang Raymond Choo, and Constantinos Patsakis. Hedge: efficient traffic classification of encrypted and compressed packets. *IEEE Transactions on Information Forensics and Security (TIFS)*, 14(11):2916–2926, 2019.

[17] Meng Shen, Mingwei Wei, Liehuang Zhu, and Mingzhong Wang. Classification of encrypted traffic with second-order markov chains and application attribute bigrams. *IEEE Transactions on Information Forensics and Security (TIFS)*, 12(8):1830–1843, 2017.

[18] Bum Jun Kwon, Virinchi Srinivas, Amol Deshpande, and Tudor Dumitraş. Catching worms, trojan horses and pups: Unsupervised detection of silent delivery campaigns. *arXiv preprint arXiv:1611.02787*, 2016.

[19] George Stergiopoulos, Alexander Talavari, Evangelos Bitsikas, and Dimitris Gritzalis. Automatic detection of various malicious traffic using side channel features on tcp packets. In *European Symposium on Research in Computer Security (ESORICS)*, pages 346–362. Springer, 2018.

[20] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. *Computer Networks*, 165:106944, 2019.

[21] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020.

[22] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management (TNSM)*, 2019.

[23] Cong Dong, Chen Zhang, Zhigang Lu, Baoxu Liu, and Bo Jiang. Cetanalytics: Comprehensive effective traffic information analytics for encrypted traffic classification. *Computer Networks*, page 107258, 2020.

[24] Alessandro D'Alconzo, Idilio Drago, Andrea Morichetta, Marco Mellia, and Pedro Casas. A survey on big data for network traffic monitoring and analysis. *IEEE Transactions on Network and Service Management (TNSM)*, 16(3):800–813, 2019.

[25] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[26] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[27] Open Information Security Foundation. Suricata. https://suricata-ids.org/. Accessed Feburary 8, 2020.

[28] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy (S&P)*, pages 226–241. IEEE, 2005.

[29] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, volume 10, page 14, 2010.

[30] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 349–360, 2012.

[31] Kyu-Seok Shim, Sung-Ho Yoon, Su-Kang Lee, and Myung-Sup Kim. Sigbox: Automatic signature generation method for finegrained traffic identification. *Journal of Information Science and Engineering*, 33(2):537–569, 2017.

[32] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX)*, pages 1093–1110, 2017.

[33] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. A lustrum of malware network communication: Evolution and insights. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pages 788–804. IEEE, 2017.

[34] Ewerton RS Castro, Marcelo S Alencar, and Iguatemi E Fonseca. Probability density functions of the packet length for computer networks with bimodal traffic. *International Journal of Computer Networks & Communications*, 5(3):17, 2013.

[35] Sean McCreary et al. Trends in wide area ip traffic patterns-a view from ames internet exchange. In *ITC specialist seminar*, 2000.

[36] Stratosphere. Stratosphere laboratory datasets. https://www.stratosphereips.org/datasets-overview. Retrieved November 21, 2019.

[37] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[38] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *International Conference on Information Networking (ICOIN)*, pages 712–717. IEEE, 2017.

[39] Yoshihiro Ishikawa. Open source as fuel of recent apt. https://hitcon.org/2017/pacific/0composition/pdf/Day1/R1/R1-4.12.7.pdf. Accessed January 8, 2020.

[40] Candid Wueest and Doherty Stephen. The increased use of powershell in attacks. *CA: Symantec Corporation World Headquarters*, pages 1–18, 2016.

[41] Jing Wang and Ioannis Ch Paschalidis. Botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, 4(2):392–404, 2016.

[42] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019.

[43] *cicf*lowmeter. http://netflowmeter.ca. Retrieved November 9, 2019.

[44] Syed Ali Raza Shah and Biju Issac. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, 80:157–170, 2018.

[45] George Kadianakis and Nick Mathewson. obfs3 (the threebfuscator), jan. 2013. *URl: https://gitweb. torproject. org/pluggabletransports/obfsproxy. git/tree/doc/obfs3/obfs3-protocol-spec. txt (cit. on p. 14)*.

[46] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 57–69, 2015.

[47] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *Journal of the ACM (JACM)*, 65(6):1–37, 2018.

[48] Zhongjie Wang, Shitong Zhu, Yue Cao, Zhiyun Qian, Chengyu Song, Srikanth V Krishnamurthy, Kevin S Chan, and Tracy D Braun. Symtcp: Eluding stateful deep packet inspection with automated discrepancy discovery. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[49] Soo-Jin Moon, Jeffrey Helt, Yifei Yuan, Yves Bieri, Sujata Banerjee, Vyas Sekar, Wenfei Wu, Mihalis Yannakakis, and Ying Zhang. Alembic: automated model inference for stateful network functions. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 699–718, 2019.

[50] Giampaolo Bovenzi, Giuseppe Aceto, Domenico Ciuonzo, Valerio Persico, and Antonio Pescapé. A big data-enabled hierarchical framework for traffic classification. *IEEE Transactions on Network Science and Engineering*, 7(4):2608–2619, 2020.

[51] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.