FOSS: Towards Fine-Grained Unknown Class Detection Against the Open-Set Attack Spectrum With Variable Legitimate Traffic

Ziming Zhao, *Student Member, IEEE*, Zhaoxuan Li, *Student Member, IEEE*, Xiaofei Xie, Jiongchi Yu, Fan Zhang, *Member, IEEE*, Rui Zhang, Binbin Chen, *Member, IEEE*, Xiangyang Luo, Ming Hu, *Member, IEEE*, and Wenrui Ma

***Abstract*— Anomaly-based network intrusion detection systems (NIDSs) are essential for ensuring cybersecurity. However, the security communities realize some limitations when they put most existing proposals into practice. The challenges are mainly concerned with (i) fine-grained unknown attack detection and (ii) ever-changing legitimate traffic adaptation. To tackle these problem, we present three key design norms. The core idea is to construct a model to split the data distribution hyperplane and leverage the concept of isolation, as well as advance the incremental model update. We utilize the isolation tree as the backbone to design our model, named FOSS, to echo back three norms. By analyzing the popular dataset of network intrusion traces, we show that FOSS significantly outperforms the state-of-the-art methods. Further, we perform an initial deployment of FOSS by working with the Internet Service Provider (ISP) to detect distributed denial of service (DDoS) attacks. With real-world tests and manual analysis, we demonstrate the effectiveness of FOSS to identify previously-unseen attacks in a fine-grained manner.**

***Index Terms*— Intrusion detection system, fine-grained unknown class detection, isolation forest.**

Manuscript received 27 February 2023; revised 9 January 2024; accepted 8 May 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Zhang. This work was supported in part by the National Natural Science Foundation of China under Grant 62227805, Grant 62072398, and Grant 62172405; in part by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity Research and Development Programme under Grant NCRP25-P04-TAICeN; in part by Singapore University of Technology and Design-Zhejiang University Innovation, Design and Entrepreneurship Alliance (SUTD-ZJU IDEA) IDEA Grant for Visiting Professors under Grant SUTD-ZJUVP201901; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20220075; in part by the Fok Ying-Tung Education Foundation for Young Teachers in the Higher Education Institutions of China under Grant 20193218210004; in part by the State Key Laboratory of Mathematical Engineering and Advanced Computing; in part by the Open Foundation of Henan Key Laboratory of Cyberspace Situation Awareness under Grant HNTS2022001; and in part by the Key Research and Development Program of Zhejiang Province under Grant 2023C01039. *(Corresponding authors: Fan Zhang; Wenrui Ma.)*

Ziming Zhao is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: zhaoziming@zju.edu.cn).

Zhaoxuan Li and Rui Zhang are with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, and also with the School of Cyber Security, UCAS, Beijing 100049, China (e-mail: lizhaoxuan@iie.ac.cn; zhangrui@iie.ac.cn).

Xiaofei Xie and Jiongchi Yu are with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: xfxie@smu.edu.sg; jcyu.2022@phdcs.smu.edu.sg).

Fan Zhang is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China, and also with Zhengzhou Xinda Institute of Advanced Technology, Zhengzhou 450000, China (e-mail: fanzhang@zju.edu.cn).

Binbin Chen is with the Advanced Digital Sciences Center, Singapore 138632, and also with the Information Systems Technology and Design (ISTD) Pillar, Singapore University of Technology and Design, Singapore 487372 (e-mail: binbin_chen@sutd.edu.sg).

Xiangyang Luo is with Henan Province Key Laboratory of Cyberspace Situation Awareness, Zhengzhou 450001, China (e-mail: luoxy_ieu@sina.com).

Ming Hu is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: ecnu_hm@163.com).

Wenrui Ma is with the School of Computer Science and Technology, Zhejiang Gongshang University, Hangzhou 310018, China (e-mail: mawenrui@zjgsu.edu.cn).

Digital Object Identifier 10.1109/TNET.2024.3413789

I. INTRODUCTION

NETWORK intrusion detection systems (NIDSs) occupy a significant role in cybersecurity infrastructures. Over the past decades, the academic and industrial communities have invested a lot of research to advance NIDS. The proposed approaches gradually evolve from signature-based to anomaly-based detection in the NIDS landscape [1], [2]. The former aims to characterize the malicious behaviors and hence to depict the per-class attack fingerprint [3], [4], [5], [6]. Yet the adversaries could adopt previously-unseen strategies such as zero-day attacks. We term the unknown a priori and known attacks collectively as the *open-set attack spectrum*. To cope with the open-set attack spectrum, the anomaly-based solutions construct profiles of benign traffic to discover unforeseen attacks that deviate from legitimate samples [1], [7], [8], [9], [10]. Thus anomaly detection becomes an indispensable step for security in the real world. However, academic communities and industrial practitioners reveal a series of limitations when they put most existing anomaly-based proposals into practice [10], [11], [12], [13]. By summarizing those issues, we recognize the following two main challenges.

(i) *Fine-grained unknown attack detection.* The anomaly-based methods can identify unknown attacks, while previous proposals are usually binary classification models[1] [7], [8]. That is to say, they could only infer whether the sample is "benign" or "abnormal", but cannot recognize that the anomaly is "unknown attack 1", "unknown attack 2", or "unknown

[1]Some existing multi-class detection methods have strong assumptions, and we summarize them in baselines (§ V-A).

attack 3", etc. Yet these fine-grained labels are the key for defenders to deploy effective countermeasures against the attacks [10], [12]. For example, the victims can count the per-flow protocol flag to mitigate flood-based DDoS [14], [15]. And they can defend the reflection-based attacks by source verification [16]. In other words, it exists a semantic gap between the model identification results and the actionable reports for network operators [11]. If the proposed scheme can automatically distinguish different unknown attack classes based on the network traffic characterization, it could facilitate understanding attack details and implementing countermeasures. Consequently, the first challenge is to detect the unknown (and known as well) attack in a fine-grained manner.

(ii) *Ever-changing legitimate traffic adaptation.* Another crucial problem is that legitimate traffic could be ever-changing. In most anomaly-based detection, they advocate only using benign traffic to train the classifier, also known as "zero-positive" learning [17], [18], then those samples deviating from legitimate traffic will be considered malicious. However, it could appear a large number of false positives when the legitimate traffic manifested as different from priors properties. Intuitively, an anomaly detector trained on benign HTTP traffic would identify normal FTP traces as malicious given that these two protocols are indeed different. Therefore, a pending issue is to adapt to the changing legitimate traffic as the system service variation or scenario alteration.

In this paper, we aim to enable fine-grained unknown attack detection, as well as consider ever-changing legitimate traffic. To this end, we present FOSS (Fine-Grained Unknown Class Detection against the Open-Set Attack Spectrum, for short), a novel tree-based model to handle the above challenges. At the high level, FOSS is designed to abide by three norms. *Norm* ①: Splitting the data distribution hyperplane instead of dividing the samples. *Norm* ②: Leveraging the concept of isolation different from distance or density. *Norm* ③: Advancing the incremental model update to cope with variable legitimate traffic. We will provide detailed elaboration on these norms in § III. To satisfy them, we design FOSS including: feature extraction, model construction (*Norm* ①), outlier detection and classification (*Norm* ②), and model update (*Norm* ③).

In summary, this paper makes two key contributions.

- To address the above two issues faced in current anomaly-based NIDSs, we present FOSS, a novel tree-based architecture that satisfies the proposed three norms. Specifically, FOSS designs a model building scheme for hyperplane partition with the Monte Carlo method to echo back *Norm* ①. Then, it leverages the isolation-based detection following *Norm* ② to improve the local anomaly perception and assign fine-grained labels. Finally, FOSS implements incremental model updates through growing and retiring mechanisms to cater for *Norm* ③.
- We fully implement a prototype of FOSS and evaluate it substantially on our testbed. The results demonstrate that FOSS significantly outperforms previous methods. Also, it can achieve robust detection against the train-time and test-time adversarial attacks. Meanwhile, we explain how FOSS implements fine-grained unknown class detection from the feature perception, and the real-world DDoS evaluation for FOSS reflects its practicality.

II. Assumptions and Problem Formulation

A. Motivating Scenarios

Consider security practitioners in the Internet Service Provider (ISP) data center, where they need to analyze traffic and report fine-grained labels for each session, *e.g.,* "Benign", "Attack 1", "Attack 2", *etc.* In the open world, traffic instances to be analyzed involve unknown attacks (previously unseen in the training set), such as zero-day attacks. Naturally, practitioners thought of leveraging anomaly-based models to detect unknown attacks. However, it has been found in practice that existing anomaly-based detection solutions mainly report "benign" or "malicious" and cannot support fine-grained attack outcomes, which means that practitioners need to analyze all malicious samples to mark fine-grained labels (is labor intensive). Alternatively, unsupervised algorithms such as clustering are considered. The problem faced by this solution is that it does not use the prior knowledge of known attacks, resulting in misclassifications for some known attack samples. And practitioners do not know how many unknown attacks are included in the real-world samples, which may affect the clustering hyperparameters settings (such as K-means [19]). More details about the disadvantages of common clustering algorithms are discussed in § III.

One piece of good news is that a new network intrusion detection system (*i.e.,* FOSS) is provided, which can support fine-grained identification of unknown attacks. Specifically, the proposed NIDS is able to report "Benign", "Known Attack 1", "Known Attack 2", "Unknown Attack 1", "Unknown Attack 2", and so on. Benign and known attack labels correspond to the prior knowledge in the label library, which is readily available for result reports. For unknown attacks, the proposed NIDS can divide these unseen attack samples into fine-grained categories based on traffic characteristics (ideally, each category corresponds to one attack method/vulnerability). With these preliminary identification results, practitioners only need to verify a small number of samples for each unknown attack type, and could report specific attack characteristics, for subsequent cross-verifying with the threat intelligence and expanding the attack knowledge base. Such a new NIDS is practical and feasible for practitioners. Furthermore, an additional function of the proposed NIDS is to support incremental model updates, which can be used to adapt to new benign/legitimate traffic, and can also be used to update/expand the known attack library (as unknown attacks are identified).

B. Threat Model and Assumptions

Adversary Model. We consider unknown intrusions such as zero-day attacks that exist in real-world scenarios. In other words, strong adversaries will adopt the emerging attack strategies that are previously unseen by victims, including variants of the existing attacks or brand new ones. Therefore, it is hard to have any prior data about these unforeseen attacks (*i.e.,* not included in the training set). In addition, we mainly focus on *encrypted traffic analysis* in this paper, since the growing prevalence of encryption protocols in network transmissions, such as SSL/TLS and SSH. Concretely, we tend to characterize traffic behavior by portraying packet field distribution rather than analyzing transmission content, *e.g.,* TCP Payload.

TABLE I
THE SYMBOL DESCRIPTION FOR PROBLEM FORMULATION

Notation	Definition
S_{tra}	The training set
S_{tes}	The testing set
B	Benign traffic (*i.e.,* all legitimate traffic)
A_k^i	The *i-th* known attack
n	The type number of known attacks
A_u^i	The *i-th* unknown attack
m	The type number of unknown attacks
L_o^i	The *i-th* original legitimate traffic
s	The type number of original legitimate traffic
L_n^i	The *i-th* new legitimate traffic
q	The type number of new legitimate traffic
M	A trained-well model
M'	The updated model

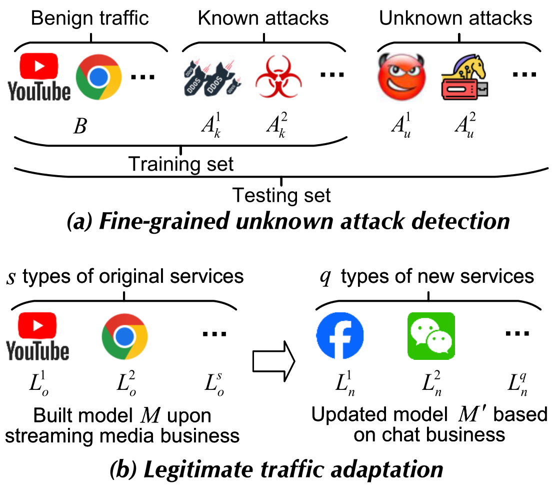

Benign traffic
Known attacks
Unknown attacks
YouTube
B
A_k^1
A_k^2
A_u^1
A_u^2
Training set
Testing set
(a) Fine-grained unknown attack detection
s types of original services
q types of new services
YouTube
L_o^1
L_o^2
L_o^s
L_n^1
L_n^2
L_n^q
Built model M upon streaming media business
Updated model M' based on chat business
(b) Legitimate traffic adaptation

Fig. 1. Illustrative explanation of two challenges. Subfigure (a) represents the proposed model that could identify known/unknown attacks in a fine-grained manner. Subfigure (b) refers to the model that can be incrementally updated to adapt to new legitimate service traffic.

Assumptions. We assume there is no prior knowledge when suffering unprecedented attacks in practice. We are also not aware of how many types of unknown attacks exist in the collected traffic samples in advance. Meanwhile, we do not assume additional collaborations from other Internet entities, such as IP blacklists provided by security vendors. Moreover, the victim server needs to support variable legitimate traffic as the scenario and business change. This means that benign traffic is not set in stone. When the server business changes, we suppose the victim can provide some legitimate traffic samples for model adaptation.

C. Problem Formulation

In this section, we provide the precise definition of two critical challenges concerned in our work, with the involved notations summarized in Tabel I.

Fine-Grained Unknown Attack Detection. Given a prior dataset S_{tra}, consisting of samples from benign traffic B and known cyber attacks $\{A_k^1, A_k^2 \cdots A_k^n\}$ ("k" represents "known"), where n refers to the number of known classes. And we use S_{tra} as the training set to fit the model M. When deploying M in practice, it will encounter the open-world testset S_{tes} include: (i) samples with the ground-truth labels from $\{B, A_k^1, A_k^2 \cdots A_k^n\}$; (ii) instances of emerging classes $\{A_u^1, A_u^2 \cdots A_u^m\}$ ("u" represents "unknown"), where m denotes the number of unknown classes, and it is unknown to us in advance.

As shown in Figure 1(a), the *fine-grained unknown attack detection* refers to: M can identify the specific-attack labels of test samples, *i.e.,* the sample prediction result is B, A_k^1, A_k^2, A_u^1, A_u^2, or others.[2]

Ever-Changing Legitimate Traffic Adaptation. In Figure 1(b), suppose a server will generate legitimate traffic $\{L_o^1, L_o^2 \cdots L_o^s\}$ ("o" represents "original") in the original business (*e.g.,* Streaming media: YouTube and Chrome as examples), while the server could also support legitimate traffic $\{L_n^1, L_n^2 \cdots L_n^q\}$ ("n" represents "new") after the business scenario changes (*e.g.,* Chat: Facebook and WeChat as examples). Among them, s and q represent the type number of legitimate traffic from original services and new businesses, respectively.

Given a trained model M fitted from benign traffic B and attack samples A, where B denotes a series of original legitimate traffic. When the server adjusts the business scenario, the *ever-changing legitimate traffic adaptation* refers to: M can directly perform model growing based on increased legitimate traffic $\{L_n^1, L_n^2 \cdots L_n^q\}$ without retraining the existing structure. After growing, the updated model M' can recognize all the original and increased legitimate traffic to realize business adaptation. On the contrary, if the original legitimate traffic no longer requires to be supported, the model will be automatically or manually triggered to the retiring mechanism to forget the outdated samples.

Essentially, the second challenge requires that the model be able to support incremental updates. Therefore, this capability of incremental update also applies to expand the knowledge base of known attacks. When the instance of any fine-grained unknown attack reaches an adequate number, the model will automatically update to enable learning for this attack. Subsequently, the unknown attacks that have been identified will become known attacks in the new model. Furthermore, readers may be concerned about whether an adversary could stealthily inject attack traffic into the updated legitimate traffic. On the one hand, performing expert analysis for part of the samples can facilitate improving data quality in practice. On the other hand, we also consider this train-time data pollution in § V-E (evaluation results show that the impacts of poisoned samples could be alleviated due to the training sampling ψ in our design § IV-A), and the proposed retiring mechanism (§ IV-C) could trigger the instance forgetting and node deletion to rectify the wrong branches.

III. Proposed Design Norms for Fine-Grained Unknown Class Detection

We elaborate here on the three key norms for fine-grained unknown class detection with variable legitimate traffic.

Norm 1: *Splitting the data distribution hyperplane instead of dividing the samples.* So-called methods that divide the samples refer to the current supervised-based and anomaly-based

[2]Note that the model marks the attack with a series of fine-grained codenames (*e.g.,* "new attack 1", "new attack 2"), rather than naming each attack (*e.g.,* "Heartbleed"). We aim to automatically detect unknown attacks and distinguish their types from each other in this work, which benefits network operators to analyze and further deploy countermeasures. The specific attack names can be given by the security communities, just like we could also call "Heartbleed" as "Buffer over-read."

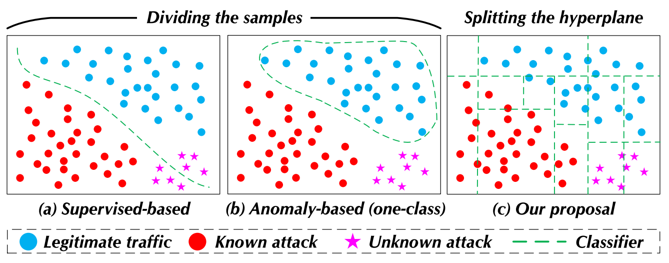


Fig. 2. The norm 1: hyperplane partition.

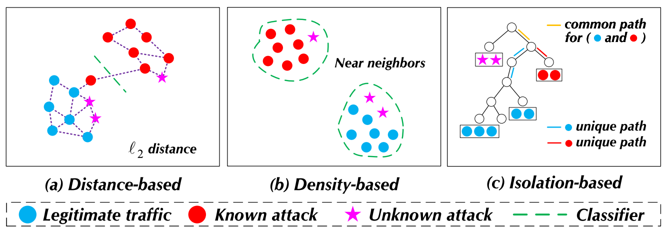


Fig. 3. The norm 2: leverage the isolation concept.

(one-class) detections. As Figure 2(a) shows, the supervised-based [20], [21], [22] approach generally trains a classifier to distinguish benign traffic and known attacks. This classifier is not suitable for unknown attacks as lacking corresponding prior knowledge in the training set, thus it tends to random guess against new attacks. A typical anomaly-based (also known as "zero-positive" learning [17]) model is depicted in Figure 2(b), it uses solely the legitimate samples to fit the detector, which works well in identifying the instances that deviate from normal [1], [7], [23]. However, it usually reports normal or abnormal rather than specific attack categories, and it could not be very sensitive to subtle outliers (also known as "local anomalies are difficult to detect" [24]).

We collectively refer to Figure 2(a) and (b) as the methods that divide the sample. In contrast, our proposal is splitting the data distribution hyperplane, it iteratively selects one dimension feature and separates the region into two subspaces, as shown in Figure 2(c). Given that dividing the sample boundary is prone to failure in an open-set environment, splitting the hyperplane is more suitable for the scenario with variable samples. Note that the distribution hyperplane partition range does not consider all regions at once, the specific split process will be adjusted according to the situation. For instance, a classifier could select the split threshold $T_s \in [D_{min}, D_{max}]$ in sample subset Set_1, while using $T'_s \in [D'_{min}, D'_{max}]$ in sample subset Set_2 based on a specific dimension D or D', where D_{min}/D'_{min} and D_{max}/D'_{max} refer to the minimum and maximum values of the corresponding dimension features. To be clarified, this process aims to split subspace without utilizing any labels, the minimum and maximum values are only used to help reduce invalid partitions in the current situation. Readers may be confused about whether this statement is similar to the unsupervised clustering algorithms, we immediately elucidate this problem in Norm 2.

Norm 2: *Leveraging the concept of isolation different from distance or density.* Compared to common unsupervised clustering algorithms, we tend to leverage the concept of isolation. Figure 3(a) and (b) illustrate distance-based (*e.g.,* K-means [19]) and density-based (*e.g.,* DBSCAN [25]) methods respectively, as two typical unsupervised clustering algorithms. They have some disclosed drawbacks in open-set detection,

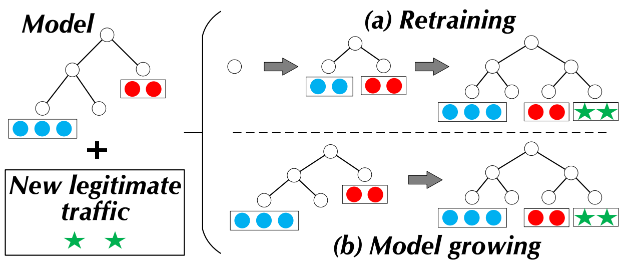


Fig. 4. The norm 3: incremental model update.

such as K-means requires the number of input clusters, and DBSCAN does not handle uneven densities well. A more crucial problem is they could be insensitive to subtle-feature abnormal deviation since their distance and radius calculations require based on a fixed-dimensional (*e.g.,* feature dimension) vector. This will cause subtle anomalies in one dimension or several dimensions to be diluted by the overall vector during computation. Given two feature vectors $\mathcal{V} =< v_1, v_2 \cdots v_d >$, $\mathcal{V}' =< v'_1, v'_2 \cdots v'_d >$, if only $v_j \neq v'_j$ and $v_i = v'_i$ for $i \in C_{[1,d]}(j)$, the dilution refers to $\mathcal{D}(\mathcal{V}, \mathcal{V}') \ll \mathcal{D}(v_j, v'_j)$ where $\mathcal{D}$ represents ℓ_p distance. It means that these methods are hard to perceive subtle-feature anomalies, thereby missing potential attacks.

We propose to leverage isolation-based technology as a distinct method from distance/density-based. The concept of isolation refers to randomly selecting a dimension to isolate the two sets for per-node split, initially to maximize the diversity of the tree [26], [27]. Since the per-node split is based on one dimension, when emerging subtle-feature anomalies, the model is able to isolate these anomalies from other samples. Readers could concern that when handling the high-dimension feature vector, completely random selection cannot guarantee that each feature will be selected, and some non-robust features could affect the splitting results. We realize this problem and carefully design delicate search algorithms to alleviate these impacts, more details will be explained in § IV-A. Furthermore, isolation-based schemes could facilitate local anomaly detection, as we stated in § IV-B.

Norm 3: *Advancing the incremental model update to cope with variable legitimate traffic.* In the problem space of unknown class detection, we have to consider that legitimate traffic may also ever change due to real-world business scenario adjustments. Although we could not predict what future legitimate traffic will look like, we still hope that the model can flexibly adapt to upcoming scenarios when servers provide some new benign traffic. Therefore, the third norm is advancing the model growing in an incremental manner to cope with variable legitimate traffic. Figure 4 provides an illustrative explanation for this idea. Existing attack detectors generally need to be retrained to suit the new dataset, just like in Figure 4(a), it retrains from the initialized model state.

To fulfill the model growing process of Figure 4(b), we choose the tree as the backbone architecture of FOSS. When the increased legitimate traffic is input to the original model, it will fall on some leaf nodes. If the leaf nodes meet the splitting conditions, the model will directly execute the growing process without having to train the existing structure. This growing process is still applicable for attack samples, which means that the detected fine-grained unknown attacks could be used to update the model and expand the

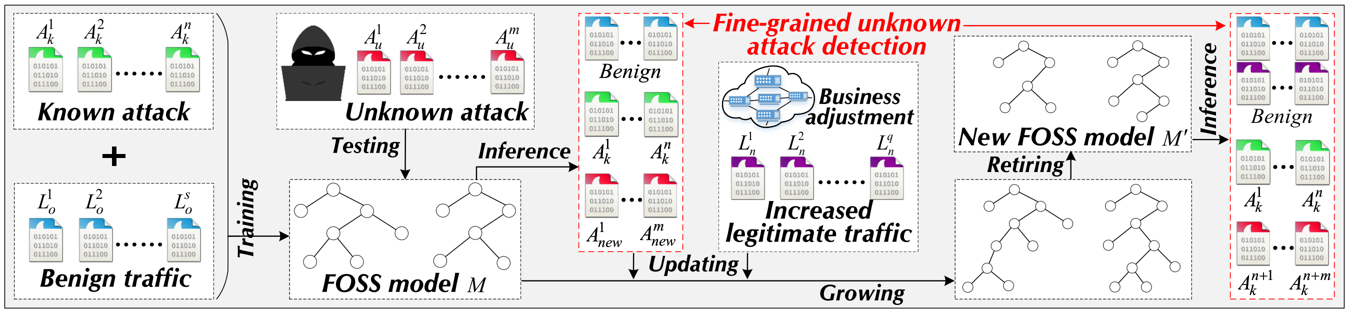


Fig. 5. The overview of FOSS, including model construction, outlier detection/classification, and model update.

TABLE II
THE 106-D FEATURE SET. "TTL": "TIME TO LIVE"

Feature				Dim
Protocol				1
Direction	Sum			33
	Frame	IP	TCP	
Forward Backward Bi-dir	Packet_num, Duration	Flags_df, Flags_mf, Frag_offset	ACK, URG, PUSH,RESET, SYN, FIN	
Direction	Statistic			72
	Max Min Mean Std			
Forward Backward Bi-dir	Total Length, Time_delta, Calculated Window Size, Scale_factor, Window, TTL			

attack library. However, endless growth could lead to model oversized and path explosion. So we config model retiring mechanism for FOSS (§ IV-C) to merge unnecessary branches and reduce the model scale, as well as alleviate the risk from exploitation of outdated samples.

IV. DESIGN DETAILS OF FOSS

In this section, we first elaborate on the design overview and feature extraction of FOSS, followed by the technical details (§ IV-A∼§ IV-C). Furthermore, we conduct a theoretical analysis for its effectiveness in terms of the critical feature perception and the complexity of time/space overhead, which are summarized in § IV-D.

Overview. We start by introducing an overview of FOSS, including feature extraction, model construction (§ IV-A), outlier detection and classification (§ IV-B), and model update (§ IV-C). As Figure 5 depicts the workflow of FOSS, we first design a model building scheme for hyperplane partition with the Monte Carlo method to realize Norm 1. Then we leverage isolation-based anomaly following Norm 2 to improve the local-abnormal perception and assign the fine-grained label. Finally, to echo back Norm 3, we propose the model update including the growing mechanism and retiring one.

Feature Extraction. Our proposal on feature extraction is based on the traffic session with the 5-tuple index, *i.e.,* $\{Source\ IP, Source\ Port, Destination\ IP, Destination\ Port, Protocol\}$. Table II shows the feature generated for each bi-directional flow, it characterizes the traffic in terms of temporal, volumetric, and header-field distributions. Specifically, the feature vector roughly includes three parts. (i) the protocol coding with one dimension. (ii) the count of a series of flags from layers 1-4 (such as IP Fragment, TCP Flags) in three situations (forward, backward, and bidirectional), which is 33 dimensions. (iii) the 72-dimensional statistical results (*i.e.,* max, min, $mean$, and std) for several transport-functional fields (*i.e.,* TTL, window size) in three situations about direction. In general, all features are either int or $float$ types.

A. Model Construction (#Norm 1)

Single-Tree Building. Building a single tree in FOSS is essentially an iterative binary partitioning process. Given a training set $X_n \in \mathbb{R}^d$ (d refers to the feature dimension), the subset S drawn from X_n, the single tree is recursively built according to the Algorithms 1. During the tree construction, each node splits solely based on one dimension feature q, with the randomly generated threshold $p \in [\min(S(q)), \max(S(q))]$. The recursion terminates until meeting either of the following conditions: (i) the tree reaches a height limit h_{max}, (ii) the subsample size $|S| \leqslant S_{min}$, or (iii) all data in S have the same values. After generating, each node in the *FOSSTrees* has exactly zero or two child nodes.

In the dimension selection for each node splitting, the previous methods are either completely random [26], [27], [28] or determined processes [8]. Those schemes have some drawbacks that could not cater to our demands. For one thing, the former could not handle high-dimensional data well. If the characteristics of the attack are subtle and most of the dimensions are not so significant, completely random methods will struggle under a large fraction of the inefficient features. This will increase the model complexity, more importantly, it may lose the accuracy of known attacks due to underfitting. For another, the latter inevitably limits unknown attack perception given its deterministic tree construction process based on a prior dataset. Empirically, the unknown attacks may have quite different traffic characterizations than known attacks, which means the effective features in detecting existing attacks could be unsuitable for future attacks.

To cope with the above problems, we propose a design between completely random and deterministic for FOSS to reconcile this seeming contradiction by introducing the Monte Carlo method. Specifically, the Monte Carlo method here refers to mimicking multiple times of feature selection and determining the one we think is the more effective. As described in Algorithm 2, we randomly generate a subset $\mathcal{F}_{\text{sam}}$ which consists n_{sam} candidate dimensions to assess. This process can directly increase the probability that each feature is considered, thereby alleviating the negative impact induced by high-dimensional data. For multiple candidate dimensions in each Monte Carlo process, we prefer the one whose data has scattered values (far from the mean) and presents no cluttered distribution (the small entropy value).

Algorithm 1 Function *FOSSTree*(S, h, h_{max})

```
Require: The subset $S \in X_n$, current node depth $h$, maximal depth limit $h_{max}$, minimum sample for node splitting $S_{min}$
Ensure: FOSSTree
1: if ($h \geqslant h_{max}$) or ($|S| \leqslant S_{min}$) or ($s \in S$ are the same) then
2:     return leafNode($S$)
3: else
4:     Select a dimension $q \leftarrow$ get_dimension($S$) $\in \mathcal{F}$
5:     Randomly select a split value $p \in [\min(S(q)), \max(S(q))]$
6:     $S_l \leftarrow$ filter($S(q) < p$)
7:     $S_r \leftarrow$ filter($S(q) \geqslant p$)
8:     return inNode(Left $\leftarrow$ FOSSTree($S_l, h+1, h_{max}$),
                     Right $\leftarrow$ FOSSTree($S_r, h+1, h_{max}$),
                     splitDim $\leftarrow q$, splitVal $\leftarrow p$)
9: end if
```

In this way, for the discrete data distribution, the hyperplane division can separate them with fewer times of paths. Also if the data in a specific dimension is cluttered, it indicates this dimension tends to fewer-useful for detection, and is susceptible to random noises [8]. Therefore, we define a tailor-made weighted entropy and choose one dimension with the smallest value.

The calculation for weighted entropy as the following procedure: (i) Given a sample set S, $S(q)$ denotes the values of dimension q from S. Unify the value range with Max-Min normalization[3] in Eq. (1), where $x \in S(q)$.

$$x = \frac{x - \min(S(q))}{\max(S(q)) - \min(S(q))} \tag{1}$$

(ii) Generating the probability distribution of $S(q)$, to get the probability $\{p_1, \cdots, p_k\}$ corresponding to value $\{v_1, \cdots, v_k\}$, where k represents the number of values. Meanwhile, computing the mathematical expectation $\mathbb{E}(S(q)) = \sum_{i=1}^{k} v_i p_i$. (iii) The weighted entropy H_q of dimension q is expressed as the sum of all terms, and each term is calculated as the ratio of the information entropy to the distance from the expectation for each element, as Eq. (2) shows.

$$H_q = \sum\nolimits_{i=1}^{k} \frac{-p_i \log_2 p_i}{|x_i - \mathbb{E}(S(q))|} \tag{2}$$

Overall, small H_q corresponds to the feature dimension that presents a separable data manifold, and its probability distribution histogram is just like "several isolated peaks tend to at sides". We provide theoretical analysis for feature perception (§ IV-D) which explains why FOSS outperform existing approaches, and demonstrates the effectiveness experimentally in § V-D. Through the above process, we can get a *FOSSTree*. Note that the tree construction does not require any ground-truth label. In other words, the model building is not classification-oriented but partitions the data hyperplane space.

[3]If $\max(S(q)) = \min(S(q))$, skip these calculations and set H_q to infinity.

Algorithm 2 Function *get_dimension*(S)

```
Require: The sample set $S \in X_n$ of the current node and the Monte Carlo sampling number $n_{\text{sam}} \leqslant d$ for dimensions
Ensure: A split dimension $q \in \mathcal{F} = \{1, \cdots, d\}$ of the current node
1: Randomly select dimensions $\mathcal{F}_{\text{sam}} = \{d_1, \cdots, d_{n_{\text{sam}}}\} \subseteq \mathcal{F}$
2: for all $q \in \mathcal{F}_{\text{sam}}$ do
3:     Count $[x_i, c_i]_{i \in \{1,\cdots,k\}}$, where $c_i$ is the occurrence number of $x_i$ and $k$ represents the number of unduplicated values in $S(q)$
4:     Compute $\mathcal{D} = [x_i, p_i]_{i \in \{1,\cdots,k\}}$, where $p_i = c_i / \sum_{i=1}^{k} c_i$
5:     Normalize the values $[x_i \in [0,1]]_{i \in \{1,\cdots,k\}}$ (c.f., Eq. 1)
6:     $\mathbb{E}(S(q)) = \sum_{i=1}^{k} x_i p_i$
7:     $H_q = -\sum_{i=1}^{k} (p_i \log_2 p_i)/|x_i - \mathbb{E}(S(q))|$
8: end for
9: return $q \leftarrow \text{argmin}([H_q]_{q \in \mathcal{F}_{\text{sam}}})$
```

Algorithm 3 Function *FOSSForest*$(X_n, N_{tree}, \psi, h_{max})$

```
Require: The input data $X_n$, the number of trees $N_{tree}$, the subset size $\psi$, and the maximal depth limit $h_{max}$
Ensure: FOSSForest
1: Initialize: FOSSForest $\leftarrow \emptyset$
2: for all $i \in \{1, \cdots, N_{tree}\}$ do
3:     $S_i \leftarrow$ sample($X_n, \psi$)
4:     FOSSForest $\leftarrow$ FOSSForest $\cup$ FOSSTree($S_i, 1, h_{max}$)
5: end for
6: return FOSSForest
```

Multi-Tree Ensemble. To increase tree diversity and alleviate the impact of random errors, we implement the multi-tree ensemble to obtain a *FOSSForest*. As Algorithm 3 clarifies, consider a training set $X_n \in \mathbb{R}^d$ (d refers the feature dimention), it will generate in parallel N_{tree} *FOSSTrees* in *FOSSForest*. For each tree, the subset S_i ($i \in \{1, \cdots, N_{tree}\}$) are randomly drawn from X_n based on the given subset size ψ. These sampling processes are beneficial to increase the tree diversity and could avoid the quality problems of the data itself to improve the model robustness. Moreover, the maximum height h_{max} can be empirically set as $\lceil \log_2 \psi \rceil$. So far, the trained *FOSSForest* is ready to be used for detection.

B. Outlier Detection/Classification (#Norm 2)

When performing detection, *FOSSForest* will offer whether the test sample is an unknown (outlier) or a known class based on multiple *FOSSTree* voting. The instances identified as unknown classes will be dumped in a buffer to wait to be assigned fine-grained labels. We first clarify how to determine whether a test sample is an outlier. The proposed design aims to be suitable for detecting local anomalies, and we depict it with an illustration. Figure 6 (a) and (b) display two example distributions of global anomalies, the unknown class can be detected even using the typical distance/density-based methods. While in Figure 6 (c), not all classes are evenly distributed so they cannot be measured based on a uniform

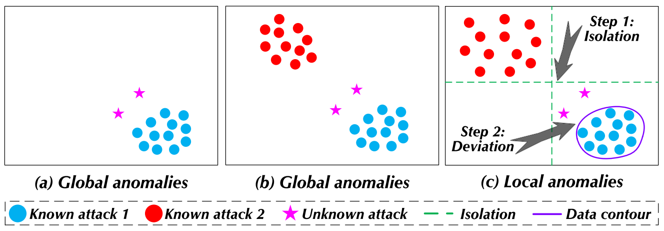


Fig. 6. Recognize outliers in local anomalies.

distance/density threshold. To tackle this task, we utilize fewer paths to determine the isolation path anomalies (step 1) and far from the data cloud centroid to determine instance deviation anomalies (step 2). Overall, the instance will be considered the unknown class if it is isolated by fewer paths and far from the data cloud centroid of the leaf node.

Isolation Path Anomaly. So-called isolation path anomaly refers to the instance with a short path length (traversed from the root node) being more likely to be an outlier than the long one. Therefore, the main problem is to determine the abnormal path length threshold for each tree. As suggested by Mu et al. [28], for an ordered leaf node length list, it is the appropriate threshold that the breakpoint makes the two sublists (corresponding to anomaly regions and normal regions) of cumulative frequency most similar. Specifically, we produce an ascending order list L which records all node path lengths for each *FOSSTree*. The determined threshold $\mathbb{L}$ will make the absolute value between the standard deviations from the two sublists[4] take the minimum value, as Eq. (3):

$$\mathbb{L} = \underset{\ell \in \{1, \cdots, max(L)\}\}}{\arg\ \min} |\sigma(L_l) - \sigma(L_r)| \tag{3}$$

where L_l and L_r denote the left sublist and the right sublist respectively, and $\sigma(\cdot)$ represents standard deviation.

Instance Deviation Anomaly. The second step is to calculate sample deviations from the corresponding leaf node for those satisfying isolation path anomalies. On the one hand, this process could cope with local anomalies in Figure 6. On the other hand, validating sample deviation can effectively prevent anomaly over-detection[5] since some traffic mutations, thereby reducing false positives and improving robustness. Inspired by previous research on mass estimation [24], we construct a data cloud for each leaf node. Particularly, given a leaf node with n instances $\{x_1, \cdots, x_n\}$, the centroid $\mathbb{C} = \frac{1}{n}\sum_{i=1}^{n} x_i$, and the radius r refers to the ℓ_p norm between $\mathbb{C}$ and the farthest instance. Then, if x_t makes Eq. (4) hold, we consider x_t to be an instance deviation anomaly in the data cloud,

$$\|x_t - \mathbb{C}\|_p \geq r \tag{4}$$

where $\|\cdot\|_p$ denotes the ℓ_p norm.

[4]These two sublists correspond to anomaly regions and normal regions, the part with shorter path lengths refers to anomaly regions. This is because an instance having a short path length, which is the number of edges it traversed from the root node to a leaf node, is more likely to be an anomaly than an instance having a long path length [28].

[5]So-called "anomaly over-detection" refers to if only use isolation path anomaly without instance deviation anomaly, the known-class samples may also be identified as anomalies. In Figure 6(c), after step 1, the pink pentagrams and blue circles are isolated, while the blue circles belong to anomaly over-detection. After instance deviation calculation in step 2, this problem could be mitigated and only the pink pentagrams are eventually identified as anomalies.

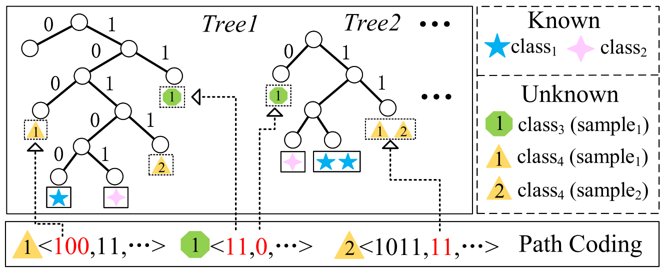


Fig. 7. Isolation-based path binary coding.

Multi-Tree Voting. For a test sample x, it will fall into one leaf node in each *FOSSTree*. If the isolation path anomaly and instance deviation anomaly are satisfied simultaneously, the corresponding tree will label *UnknownClass* for x, otherwise give the label of this leaf node. Therefore, the voting result $\mathcal{Y}$ is obtained by Eq. (5):

$$\mathcal{Y} = \arg\ \max_{y} F[y] \tag{5}$$

where $F[y]$ is the class frequency for class y, $y \in \{c_1, \cdots, c_n, UnknownClass\}$. Among them, c_i denotes the currently known classes (& benign), and n represents the number of existing known classes. For the samples identified as known classes prepare to be outputted, while all *UnknownClass* instances will be further assigned fine-grained labels.

Assign Fine-Grained Unknown Class Labels. All samples identified as *UnknownClass* will enter the buffer $\mathcal{B}$ to be assigned the fine-grained labels. To distinguish various attack categories based on their "isolation information" from the hyperplane partition, we perform the path coding for each instance. Figure 7 provides an illustrative explanation of path coding, for a instance x, we record its node binary coding in each *FOSSTree*. Thus the path coding of x in *FOSSForest* refers to $\mathcal{P} =< C_1, \cdots, C_{N_{tree}} >$, where N_{tree} denotes the number of *FOSSTrees*, and C_i $(i \in \{1, \cdots, N_{tree}\})$ represents the node coding in i-th *FOSSTrees*. We define the similarity of two node codings (C_i and C_j) as Eq. (6).

$$\mathcal{S}(C_i, C_j) = e^{-\frac{2 \times L_{\text{sub}}}{max(L_i, L_j)}} \in [e^{-1}, 1] \tag{6}$$

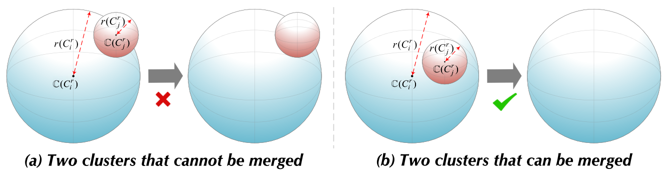
where L_i and L_j denote the coding string length of C_i and C_j respectively, and L_{sub} represents the common substring length starting from the first char, *i.e.,* $C_i[0 : L_{\text{sub}}] = C_j[0 : L_{\text{sub}}]$ and $C_i[0 : L_{\text{sub}} + 1] \neq C_j[0 : L_{\text{sub}} + 1]$. Then the isolation similarity of two samples $(\mathcal{P}_i, \mathcal{P}_j)$ is defined in Eq. (7):

$$\mathcal{I}(\mathcal{P}_i, \mathcal{P}_j) = \sum_{k=1}^{N_{tree}} \mathcal{S}(C_i^k, C_j^k)/N_{tree} \tag{7}$$

where C_i^k and C_j^k from the same tree. We next cluster those samples of $\mathcal{B}$ based on their isolation similarity $\mathcal{I}$ to each other. If the isolation similarity of two instances is less than the threshold A_d, it will add 1 to the number of adjacent samples for these two instances. For the instance whose adjacent number more than M_c could be considered a center, iteratively put samples adjacent to the center into a set to perform clustering.[6]

After the isolation similarity clustering, we examine which raw clusters need to be merged. This is for the consideration of

[6]This process (*i.e.,* isolation similarity clustering) is detailed on https://github.com/Secbrain/FOSS/tree/main/clustering.



(a) Two clusters that cannot be merged
(b) Two clusters that can be merged

Fig. 8. Clusters merging after path coding clustering.

instances from one class that could fall into multiple scattered leaf nodes in each tree, since the single-class sample diversity. Furthermore, it could improve the error-resilient ability of FOSS for clustering parameter selection. This merging process also leverages the technology of instance deviation anomaly in the aforementioned. The specific scheme is as follows: consider n raw clusters $\{C_1^r, \cdots, C_n^r\}$, for any two clusters C_i^r and C_j^r $(i, j \in [1, n]$ and $i \neq j)$, if Eq. (8) holds, C_i^r and C_j^r will be merged into a cluster.

$$\|\mathbb{C}(C_i^r) - \mathbb{C}(C_j^r)\|_p \leq |r(C_i^r) - r(C_j^r)| \tag{8}$$

where $\|\cdot\|_p$ denotes the ℓ_p norm, $\mathbb{C}()$ and $r()$ denote the centroid and radius of the data cloud in the corresponding raw cluster. Figure 8 provides an illustrative explanation for this idea, the subfigure (b) meets the condition of Eq. (8) so the two clusters can be merged while subfigure (a) cannot. Note that the results of the above-mentioned merging process will not be affected by the order of the merges. Finally, we can get m merged clusters $\{C_1^e, \cdots, C_m^e\}$ and each cluster C_i^e $(i \in [1, m])$ corresponds to a fine-grained unknown class label.

C. Model Update (#Norm 3)

We introduce here the model adaptation scheme against ever-changing legitimate traffic without retraining techniques. It includes a growing mechanism and a retiring one.

Growing Mechanism. As mentioned in § II-C, the legitimate traffic could change since the system service adjustment and so on. Therefore, the growing mechanism benefits to absorb new legitimate traffic into the model knowledge about benign samples in an incremental manner. Naturally, it also can be used to learn the emerging unknown attacks which are already assigned fine-grained labels. So the model growing mechanism can be triggered manually (added new training data from outside the model) or automatically (fine-grained unknown class detected inside the model).

We next clarify the model growing process with a case that updates previously detected fine-grained unknown classes in Figure 9. In the buffer, a series of instances that be identified as "New Class 1" (the yellow triangle) will be fed to the original model to perform the growing mechanism. These instances could fall into several leaf nodes and cause nodes to split further as described in § IV-A. Then, the original leaf node will be replaced with the newly created subtree to complete the model update. As a result, the new tree after updating can detect "New Class 1" with the way for known class identification. This growing process is carried out locally and doesn't need to retrain the existing backbone of the tree body. Particularly, the multiple *FOSSTrees* in the *FOSSForest* are independent and can be parallelly updated.

Retiring Mechanism. In addition, we consider the customers could offline some system services, thereby some old legitimate traffic will not be present in future business.

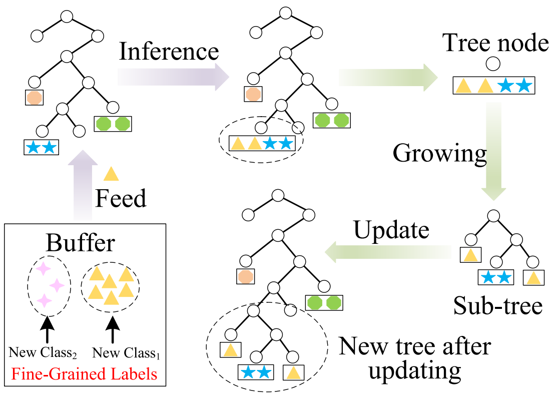


Fig. 9. Model growing in an incremental manner.

Maintaining the knowledge of outdated legitimate traffic may overcomplicate the model and cause extra overhead. More importantly, the adversaries could exploit those branches of outdated benign samples to launch attacks. A typical example from our real-world test is the DDoS attacks based on downgraded versions of SSL/TLS in § VI. Therefore, we design the retiring mechanism for FOSS to adapt to the legitimate traffic reduction.

For each *FOSSTree*, if no test sample falls on the specific leaf node of this tree in a period of time and this leaf node corresponds to the benign label, we could remove this leaf node and its instances. For a parent node that has two child leaf nodes, if one child node is removed, another child node will be merged with the parent node. And if all the two child nodes are removed, this parent node is directly removed. This process is performed recursively to complete the retiring mechanism for one tree, each independent in the *FOSSForest*. Noteworthy, some previous studies [17], [29] present to adjust the model prediction probability to forget some samples by their unlearning methods. Our design is not opposed to these technologies, their scheme can be combined in FOSS.

D. Theoretical Analysis

In this section, we conduct a theoretical analysis to prove that FOSS achieves a more effective feature selection than the completely random methods, and it is more adaptive for unknown class perception compared to the deterministic model. Moreover, we analyze its algorithmic complexity.

Feature Perception of Node Split in FOSS. We will analyze the feature perception of the Monte Carlo method in FOSS in terms of feature selection effectiveness and loss of insensitive dimensions. (i) Effectiveness analysis of the feature selection. Considering d-dimension feature set, its weighted entropy $\mathbb{H} = \{H_q(1), \cdots, H_q(d)\}$ can be calculated by Eq. (2). If the weighted entropy value is large, it means that this dimension has great randomness, which could be inefficient for hyperplane partition. Through formula derivation, we find that for the probability of performing effective node partitioning, FOSS is always greater than completely random methods, whatever the number of candidate dimensions in the Monte Carlo method n_{sam} is set to. The feature selection design of FOSS alleviates the struggle under the high-dimensional feature space of completely random model construction.

(ii) Loss measure of the insensitive feature dimensions. So-called insensitive features refer to those dimensions that produce little effect for hyperplane partition based on the current data. To adapt to detect the unknown classes that could present completely different distributions from existing samples, these insensitive dimensions cannot be discarded directly. So we quantify the loss of the insensitive feature dimensions for the deterministic model and FOSS. Through formula derivation, we obtain the consideration probability for insensitive features in the deterministic model (that calculates the entropy of all dimensions for each selection) is much smaller than that of FOSS. Thus, FOSS facilitates feature-aware generalization to isolate emerging attacks when running model inference. The specific derivation details can be found in the online repository.[7] Overall, FOSS reconciles the completely random and deterministic by introducing the Monte Carlo method, which advances the trade-off between the existing-sample perception and future-instance generalization.

Time and Space Complexity. FOSS mainly consists of five parts: feature extraction, model construction, detection & classification, fine-grained labels assignment, and model update. We summarize the corresponding time and space complexity in Table III. The notations refer to: the number of trees N_{tree}, the subset size ψ, feature dimensions d, buffer size s, number of flow $|X_n|$, number of packets M. Overall, the computational complexity of FOSS is proportional to the number of flows and packets (explanations of each part are in the online repository). Since it has no operation with high time or space complexity higher than quadratic terms, the whole process of FOSS introduces affordable low overhead.

TABLE III
THE COMPLEXITY OF THE FOSS

Steps	Time Complexity	Space Complexity
Extract feature	$O(dM)$	$O(d\|X_n\|)$
Construct *FOSSForest*	$O(N_{tree}\psi n_{\text{sam}} \log \psi)$	$O(N_{tree}d\psi)$
Classify/detect outliers	$O(N_{tree}d\psi)$	$O(N_{tree}d\psi)$
Assign fine-grained labels	$O(s \log s)$	$O(ds)$
Update *FOSSForest*	$O(N_{tree}sn_{\text{sam}} \log s)$	$O(N_{tree}d(\psi + s))$
Total	$O\begin{pmatrix} dM + N_{tree}n_{\text{sam}} \\ (\psi \log \psi + s \log s) \\ +N_{tree}d\psi \end{pmatrix}$	$O\begin{pmatrix} d\|X_n\| + d \\ N_{tree}(\psi + s) \end{pmatrix}$

TABLE IV
DATASETS USED IN OUR EVALUATION

Dataset	Categories	Description
IDS2017	FTP-Patator, SSH-Patator, Web-XSS, Web-Brute-Force, SQL-Injection, Botnet, Heartbleed, Port-Scan	Benign + 8 types of attack
VPN2016	Web-Browsing, Email, Chat, Streaming, File-Transfer, VoIP, TraP2P	7 types of benign

V. EVALUATION

In this section, we comprehensively evaluate FOSS, with code available online.[8] Specifically, the experiments are designed to answer the following research questions.

RQ1. How FOSS's detection effect compared with SOTA?
RQ2. How does FOSS perform in dynamic scenarios?
RQ3. How to interpret FOSS from feature perception?
RQ4. How FOSS presents when suffer adversarial attacks?

A. Dataset and Metric

Datasets. The experiments are based on two public datasets: IDS2017 [30] and VPN2016 [31]. The IDS2017 includes 51GB of traffic traces generated based on the B-Profile system involving more than seven types of OS. The VPN2016 captured a total amount of 28GB of traffic data from common applications with Wireshark and Tcpdump. We summarize them in Table IV, which consists of 8 types of attacks and 7 classes of legitimate traffic. If not otherwise stated, the dataset division ratio is $train$:$test$ = 6:4, and the per-group division and experiment will be randomly performed 10 times. Note that sampling ψ will produce a smaller data subset in FOSS.

Baselines. Some state-of-the-art (SOTA) methods are briefly introduced as follows: (i) Binary-classification models. Whisper [1] utilizes sequential information based on the frequency domain features to detect malicious traffic. Diff-RF [8] takes into the frequencies of visits in the leaves on the isolated forest [27] basis to detect point-by-point and collective anomalies. Kitsune [7] discovers abnormal behavior by using AutoEncoder to examination on each packet.

(ii) Multi-classification models. DBSCAN [25] and K-means [19] are typical density-based and distance-based unsupervised clustering algorithms respectively. FARE [10] is a semi-supervised clustering method for classification under low-quality labels. Note that it needs to specify the number of classes for FARE, we set it as ground truth. Cls-Anomaly [9] employs Conditioned Variational AutoEncoder and extreme value theory to devote multi-classification for known attacks. SENC [28] completes the semi-supervised classification based on isolation forest, yet it assumes only to emerge one unknown class at one time.

Parameter Settings. The hyperparameters are set as follows: the number of Monte Carlo searches n_{sam} = 10, tree number t = 200, sampling ratio ψ = 60%, minimum value of node split S_{min} = 10, clustering parameters A_d = 30 and M_c = 300.

Metrics. Two popular benchmarks are used to evaluate the performance for identifying emerging classes [10], including the clustering accuracy (ACC) and adjusted mutual information (AMI). Their upper bounds are all 1 and the larger values mean the better effect. Specially, we additionally calculate the False positive rate (FPR), False negative rate (FNR), Precision (Pre), Recall (Rec), and F1-score in binary classification experiments.

B. Compare With SOTA (RQ1)

Compare Binary-Classification SOTA. We first compare the detection effect of FOSS with binary classification models. Given all these schemes are solely using benign samples to train models, this experiment puts only legitimate instances into the training set and computes the metrics in binary classification ("benign" or "attack"). The results are summarized in Table V, and the identification accuracy of the four models achieves more than 92%. Also, FOSS and Whisper outweigh Diff-RF and Kitsune. Among them, the performance of Diff-RF could be limited due to the deterministic feature selection process, while Kitsune and Whisper may not be

[7]https://github.com/Secbrain/FOSS/tree/main/theory
[8]See repository https://github.com/Secbrain/FOSS.

TABLE V

THE BINARY-CLASSIFICATION RESULTS

Metrics	ACC	AMI	Pre	Rec	F1	FPR	FNR
Whisper	96.50	96.78	97.10	95.95	96.52	2.94	4.05
Diff-RF	93.45	92.16	94.20	92.81	93.50	5.89	7.19
Kitsune	94.10	93.86	94.70	93.58	94.14	5.36	6.42
FOSS	**98.65**	**97.61**	**98.90**	**98.41**	**98.65**	**1.11**	**1.59**

TABLE VI

THE MULTI-CLASSIFICATION EVALUATION

Group	K-Means		SENC		Cls-Anomaly		FARE		FOSS	
	ACC	AMI	ACC	AMI	ACC	AMI	ACC	AMI	ACC	AMI
g_1	68.21	58.25	87.12	86.36	92.34	91.66	90.68	89.50	**95.84**	**94.35**
g_2	63.32	54.01	73.75	72.81	75.03	74.87	86.98	85.77	**92.20**	**91.55**
g_3	64.40	55.76	74.95	73.84	74.62	73.45	87.76	86.33	**93.08**	**92.41**
g_4	63.29	52.78	72.76	71.32	73.29	72.61	86.37	85.54	**92.03**	**91.38**
g_5	62.96	47.76	64.04	63.16	68.38	67.92	81.98	84.11	**87.06**	**86.32**

sensitive enough to subtle-feature anomalies. FOSS is relatively better, it realizes >98% ACC, <1.2% FPR, and >98% $F1$.

Compare Multi-Classification SOTA. Then we evaluate the multi-classification effect for FOSS and baselines. We conduct five groups (denoted as $g_i, i \in [1, 5]$) of experiments by varying the known/unknown attack proportion (denoted as N_k:N_u). In g_1 and g_5, N_k:N_u are set as 8:0 and 0:8 respectively, *i.e.,* all known and all unknown. For g_2∼g_4, N_k:$N_u = 4$:4 and the attack types in each group are randomly selected.

The experimental results of five groups are shown in Table VI. We find that FOSS always achieves better detection results (ACC > 92% and AMI > 91%) than others. And K-means performs similar performance in different groups due to minor dependencies on the training process. Likewise, DBSCAN requires no training and its detection results are the same in five groups, *i.e.,* ACC = 63.01% and AMI = 62.78%. For SENC, Cls-Anomaly, and FARE, Cls-Anomaly stands out in g_1 and FARE is more prominent in other settings. We continue to discuss these models and FOSS in different N_k:N_u.

Different Known/Unknown Proportion. In this section, we evaluate the capability of FOSS by varying N_k:N_u. N_k is set from 8 to 0, and N_u is from 0 to 8. Figure 10 reveals how model ACC and AMI change in four models (each group conducts 10 experiments with randomly selected attack types). The overall detection performance is FOSS > FARE > Cls-Anomaly > SENC. The completely random feature selection makes SENC unable to handle known classes well, and the assumption of identifying one unknown class at a time causes a significant accuracy drop when N_u increases. For Cls-Anomaly, it can obtain a good effect when all are known attacks, but the accuracy drops rapidly with larger N_u. Except for the group of $N_u = 0$, FARE is more prominent than Cls-Anomaly and SENC. Overall, FOSS fulfills the remarkable detection effect regardless of the known/unknown class proportion. Even if all attacks are unknown, FOSS can present more than 86% ACC and AMI. Meanwhile, the standard deviation of FOSS is relatively small (∼1%) which means FOSS could not be susceptible to randomness.

C. Multi-Stage Dynamic Evaluation (RQ2)

In this section, we design a case that mimics the network traffic dynamics of real-world scenarios with 7 stages according to the IDS2017 dataset timeline (as shown in

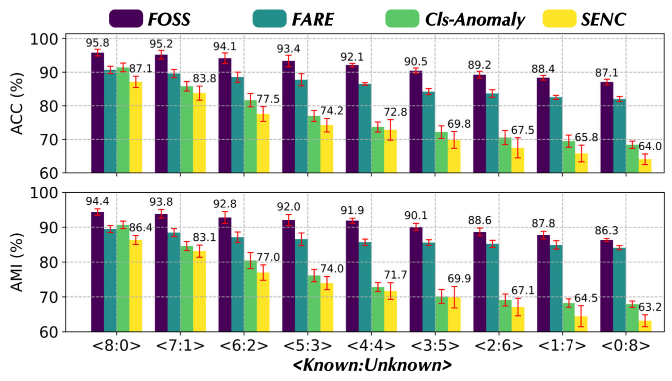


Fig. 10. Evaluation in different known/unknown proportion.

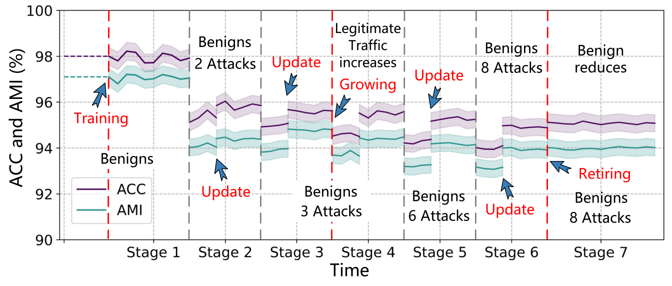


Fig. 11. Dynamic evaluation case refers to the IDS2017 dataset timeline.

Figure 11). Specifically, in Stage 1 (corresponding to Monday in IDS2017), it only provides legitimate traffic from IDS2017 to train FOSS. In stages 2, 3, 5, and 6 (corresponding to Tuesday, Wednesday, Thursday, and Friday in IDS2017), it emerges 2, 1, 3, and 2 different types of attacks respectively (refers to IDS2017 timeline), and the model will automatically update in these stages. Stages 4 and 7 represent increasing legitimate traffic from VPN2016 and reductions, thereby triggering the model growth and retirement mechanisms, respectively. Such a multi-stage process simulates emerging new unknown attacks (*i.e.,* no prior knowledge in the previous training set) and ever-changing legitimate traffic in the real world. Figure 11 displays the model performance in this dynamic process, and the detection effect of FOSS could generally achieve more than 92%. The overall accuracy may drop slightly with the class number increases. In stages 2-6, we find the effects will improve after updating the unknown classes than when outlier detection. It can be attributed to the known classification could be not so difficult as unknown detection. From stage 7, we observe that the retiring mechanism could facilitate improving the ACC and AMI due to removing the outdated nodes. By recording the runtime, FOSS spends 296s for training in stage 1; 27s, 17s, 75s, 43s, and 46s for growing in stages 2-6, respectively; as well as 25s for retiring in stage 7. It indicates that incremental model updates are better than model retraining in terms of time overhead.

D. Deep Insights to Interpretability (RQ3)

Feature Selection in Model Building. We design the Monte Carlo method for FOSS in § IV-A, and this section will provide the experimental observation of feature selection when model building. During FOSS training, we examine the tree node splitting processes and record the candidate feature dimension with its calculated H_q from the corresponding subset. Figure 12 shows two node-split instances with

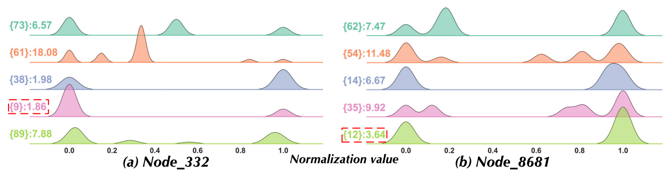


Fig. 12. The feature selection process in the model building of FOSS. The data in the two nodes are different.

$n_{sam} = 5$, we see a data group that exhibits separability tends to hold a smaller entropy value H_q. In Figure 12(a) and (b), $H_q = 1.86$ in $dim = 9$ and $H_q = 3.64$ in $dim = 12$ get the minimum entropy value respectively. And they tend to be distributed at both ends with clear separability. Note the data of the two nodes are different and cannot enforce a horizontal comparison. It demonstrates experimentally the effectiveness of our design that uses the weighted entropy to guide dimension selection with the Monte Carlo method.

Case Study: SSH Patator. We continue in-depth comprehending the attack recognition in FOSS, based on the case of the SSH Patator. To understand how FOSS identifies specific attacks based on features, we traverse all the decision paths from the trained model to record the splitting features used for each class (may correspond to multiple leaves). It will generate 106-dimensional buckets for each attack, and each bucket counts the times of this dimension used for node splitting. In general, the feature used at high frequency is more helpful to classify a specific attack than the one used at low frequency. After normalization for per-class buckets, the corresponding results of the SSH Patator attack are shown in Figure 13(a), it exhibits the top-10 and bottom-10 feature dimensions. Among them, a large fraction of the top-10 is about time delta such as dim_{87}, dim_{16}, dim_{51}, dim_{86}, dim_{19}, and dim_{18}.[9] We examine the original packets of the attack traffic and find that universally exists a $\sim$2s time interval before disconnecting. After this gap, the victim will send an encrypted SSH packet to the attacker, followed by server-induced TCP waves, and finally disconnecting. We think this phenomenon is incurred from the attack implementation of SSH Patator. To maximize each-flow effect as possible, the crack tool does not intend to actively disconnect whether the attack is successful or the test sample library is used up, yet waiting for the server to send a termination request. Therefore, this particular representation is embodied in the relevant dimension, *e.g.*, $dim_{16} : delta\_max$, $dim_{19} : delta\_std$. On the contrary, the bottom-10 mainly contains dimensions that are not very relevant to this attack: such as $dim_{10} : flag\_mf$ is about IP fragment, $dim_{46} : offset\_for$ is about IP offset, $dim_{66} : factor\_std\_for$ is about window size scaling factor.

Figure 13(b) and (c) display the 2D distribution after dimensionality reduction with t-SNE [32] based on the top-10 features and bottom-10 features respectively. It is clear that subfigure (b) which uses the last 10 dimensions cannot portray SSH Patator well given the blue dots are highly coincident with other attacks. While in subfigure (c), the SSH Patator presents better separability compared to the instances of other types. Particularly, dimensionality reduction may not be intuitive enough since each node split in FOSS refers to

[9]The details of the feature vector refer to https://github.com/Secbrain/FOSS/tree/main/features.

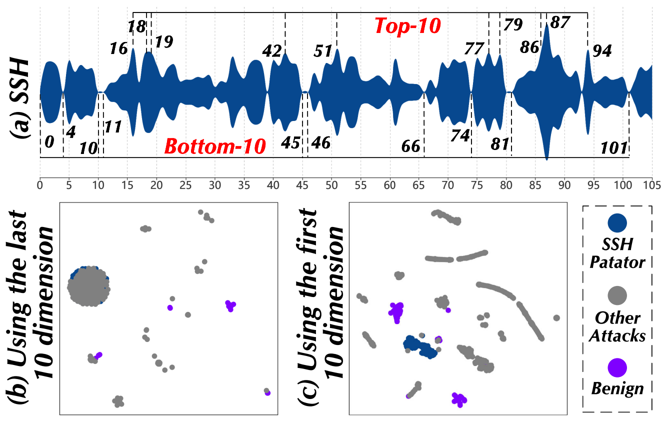


Fig. 13. Feature analysis and visualization for SSH Patator.

only one dimension. So we plot the sample scatterplot for each dimension of the top-10 feature,[10] it shows that SSH Patator can indeed be distinguished from the vast majority of other-categories samples on these key features.

E. Robustness and Adversarial Attack (RQ4)

We evaluate here the robustness of FOSS against adversarial attacks in terms of train-time data pollution and test-time evasion attacks.

Train-Time Data Pollution. Data pollution (poisoning attack) refers to some potential attackers deliberately mixing malicious traffic into the routine operations, allowing FOSS to learn impure legitimate traffic. We develop this scenario by randomly selecting some attack samples and mixing them into the benign training set. Among them, we set the proportion of polluted benign data $P_d \in \{5\%, 10\%, 20\%\}$ for model with N_k:N_u = {8:0, 4:4, 0:8}, and perform 10 experiments per setting group. From Figure 14, we observe that FOSS inevitably exhibits performance loss for ACC and AMI. The overall trend is basically similar and the unknown class number N_u has a greater impact than the pollution ratio P_d. For instance, the AMI loss from $\sim$1% to $\sim$3% for <N_k:N_u>_P_d = <8:0>_5 and <N_k:N_u>_P_d = <8:0>_20, while AMI reduces from $\sim$1% to $\sim$11% for <N_k:N_u>_P_d = <8:0>_5 and <N_k:N_u>_P_d = <0:8>_5. This can be attributed to the training sampling ψ in our design § IV-A, which can alleviate the impacts of poisoned samples to a certain extent. However, this mitigation is gradually weakened when there are fewer known classes, since the remaining contaminated samples may also guide a wrong learning direction for the unknown class detection. At the worst, the data pollution causes $\sim$14% performance loss when possesses zero known class in <N_k:N_u>_P_d = <0:8>_20. We admit that data poisoning is indeed a tricky problem, yet we would like to argue this issue could be alleviated when the polluted instances are disclosed. Given the retiring mechanism for the model update mentioned in § IV-C, we could artificially or automatically trigger instance forgetting and node deletion to rectify the wrong branches.

Test-Time Evasion Attack. Another adversarial attack we consider is test-time crafting traffic samples to evade detection. Due to the complexity of network interactions, attackers cannot arbitrarily change traffic content like other tasks such as

[10]https://github.com/Secbrain/FOSS/tree/main/evaluation

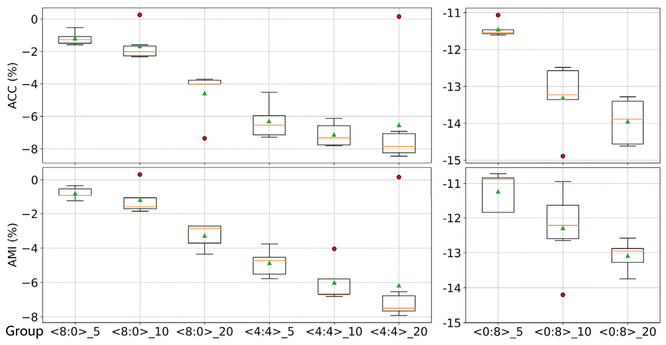


Fig. 14. Data pollution evaluation. Setting: <N_k:N_u>_P_d.

TABLE VII

EVASION ATTACK RESULTS. R_e/R_i: EVASION/INSERTION RATIOS

Attack		Known:Unknown					
R_e (%)	R_i (%)	8:0		4:4		0:8	
		ACC	AMI	ACC	AMI	ACC	AMI
5	0	93.97	93.25	91.05	90.39	86.24	85.30
10	0	92.71	92.22	89.95	89.34	85.12	84.22
20	0	90.42	90.23	88.99	88.23	83.75	83.14
0	5	94.83	93.66	91.37	90.79	86.79	85.83
0	10	93.88	92.52	90.25	89.95	85.96	85.25
0	20	92.17	91.84	89.54	89.04	84.55	84.08
5	5	93.42	92.96	90.44	89.89	85.15	84.11
10	10	92.14	91.22	88.95	88.35	84.22	82.94
20	20	88.92	88.27	87.18	86.84	82.24	81.15

image classification. For example, the attackers need to guarantee the tampered sample could pass the flow check (*i.e.,* IP Checksum) and can preserve the original malicious functions. The most recent research [33] presents that using Selective Symbolic Execution (S2E) to analyze the TCP implementation of OS kernel enables producing *insertion* and *evasion* packets to elude inspection. Therefore, we construct the insertion and evasion packets with various ratios in the test-input instances and examine the detection effect.

The results are shown in Table VII, we conduct 9 groups of experiments with the diverse ratios of insertion and evasion, *i.e.,* R_i, $R_e \in \{5\%, 10\%, 20\%\}$. The observation is a little different from the above results, and the loss of detection effect presents similarly in different N_k:N_u settings. When only exist one type of attack, the impact of evasion is greater than insertion, *e.g.,* the AMI reduces to 90.42% and 92.17% when $R_e = 20\%$ and $R_i = 20\%$ for model with N_k:N_u = 8:0. The worst results occur when both $R_e = 20\%$ and $R_i = 20\%$, cause less than 7% performance drop for ACC and AMI. Particularly, the influences of both evasion and insertion together are less than the sum of two individual impacts. We suspect the impacts from the two types of attacks have a few counteracting effects.

VI. Real-World Test: DDoS Detection

To further explore the detection effect of FOSS in the real world, we deploy FOSS on the datacenter of the local Internet Service Provider (ISP) to analyze mirrored egress traffic.[11] Particularly, the ISP purchases intrusion detection services from 17 different security vendors. The final label for each traffic session will be voted on based on the detection results

[11]All traffic data is anonymized before mirroring, and sensitive payloads that may involve privacy are zero-replaced.

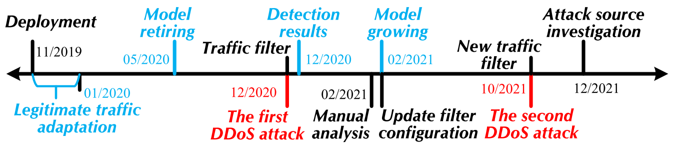


Fig. 15. The main timeline—major attack (red), FOSS (blue), and events (black) in this real-world DDoS detection.

of these vendors.[12] This egress traffic involves >200k active hosts and we monitored it across 2020 and 2021. Figure 15 displays the main timeline of a series of activities. In the beginning, we deploy FOSS and conduct the legitimate traffic adaptation for about two months. Among them, we create one FOSS instance by training with traffic from each week, eight models in total. This is beneficial to capture diverse benign samples and reduce random errors. While FOSS shows a good detection effect on the public dataset, we still want to stay conservative in this real-world testing to avoid massive false positives. Therefore, we identify the example as an unknown attack when all eight models report anomalies, instead of results voting. Meanwhile, the ISP purchases the filtering service of the security manufacturers, and we refer to these results for analysis. Our report mainly revolves around two representative attacks (red mark in Figure 15) from a similar source, launched in 12/2020 and 10/2021 respectively. Noteworthy, the retiring mechanism is triggered in 05/2020. We examine the corresponding traffic and find it is due to the new TLS protocol replacing the outdated SSL3.0 version from the previous business.

Detection Results and Observations. For the first attack, we plot the number (after $\log_{10}$-transformed) of detected attacks from the filter and FOSS in Figure 16. In this attack event, the filter found 27 types of attack and FOSS detected 49 categories. After manual analysis, we find 25 types of attacks are coincident (blue mark) between the filter and FOSS, the top-5 are "SYN Flood", "UDP Flood", "ACK Flood", "NTP Reflection", "SSDP Reflection". The yellow mark refers to different labels in the filter and FOSS, *i.e.,* it is reported as "HTTP Post Flood" and "HTTP Get Flood" in the filter while FOSS regards them as one attack. This is because our feature extraction does not involve the detailed fields of the application layer protocol (*e.g.,* HTTP), and could not distinguish them. The discovered attacks only in FOSS include 21 TCP-based (green mark) and 2 SSL/TLS-based (red mark). These TCP-based attacks are mainly flooding by combining various flags, some typical representatives are "ACK-PSH", "URG-ACK", "SYN-FIN", "ACK-RET-FIN", "URG-SYN" and so on. The samples of these types are not very large compared to the previous attack types but do cause certain bandwidth consumption. The other is the SSL/TLS-based attack, the detected two attacks are very similar to "SSL Renegotiation" and "THC-SSL Attack" by artificial comparison. They continue to establish handshakes and negotiate keys to achieve the effect of DDoS, with different SSL/TLS versions (*e.g.,* SSL3.0 and TLS1.2). We examine and find that the reason why FOSS can distinguish these two types is not based on the SSL/TLS versions (lacking corresponding feature fields), but the characteristics of the two attacks are different,

[12]Such a labeling strategy is a common approach in the real world. For example, previous research [34] on malware detection usually labels the instance based on the reporting results/engines of VirusTotal [35].

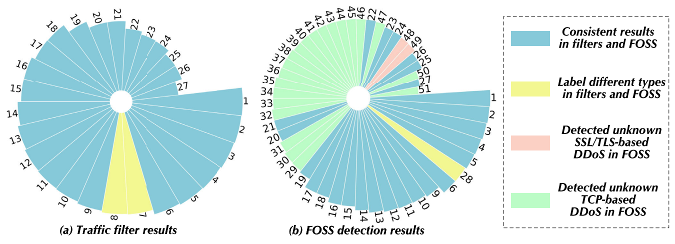


Fig. 16. The detection results in traffic filters and FOSS.

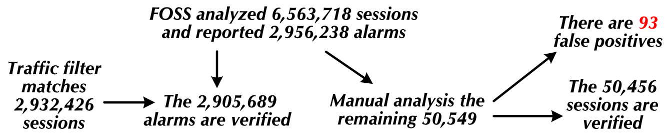

FOSS analyzed 6,563,718 sessions and reported 2,956,238 alarms
Traffic filter matches 2,932,426 sessions
The 2,905,689 alarms are verified
Manual analysis the remaining 50,549
There are 93 false positives
The 50,456 sessions are verified

Fig. 17. The quantitative analysis of Figure 16.

especially in the number of packets and byte length. Moreover, thanks to the retiring mechanism of FOSS in 05/2020 remove the nodes corresponding to the outdated benign samples of SSL/TLS, thereby FOSS could detect these SSL/TLS-based attacks. Some quantitative analyses of this attack event are shown in Figure 17, the traffic filters match 2,932,426 DDoS flows, and FOSS reports 2,956,238 attack flows. Among them, 2,905,689 flows from FOSS's alarm results are verified to be correct by the filter report, this means that FOSS at least missed 26,737 attack flows (attributed to we mark malicious when all 8 models report anomalies). In addition to those verified by the filter results, the remaining 50,549 flows are analyzed manually. The results show that 50,456 sessions are attacks and only 93 are false positives. This indicates that the scheme of multiple model validation indeed achieves low false alarms, with false positives accounting for only 0.003146% (*i.e.,* 93÷2,956,238) of all alarms. Overall, FOSS realizes predominant outcomes in fine-grained unknown attack detection, which can be beneficial to strengthen filtering services and coping strategies in practice.

Attack Source Analysis. Based on the detection results of FOSS, the manufacturer has updated the filter configuration to improve the traffic scrubbing abilities. In the second attack, it emerges 19 consistent types (blue), 15 TCP-based types (green, only from FOSS), 2 SSL/TLS types (red), and 2 HTTP types (yellow, refers to different labels). The high consistency of attack types implies the detection results in the first attack event are mostly correct since they could exploit similar attack methods given the similar attack source. Then, we plot the local network topology diagram for the second event in the online repository,[13] the "red node" denotes the victims and the "black node" represents the attackers. We can see that the victim whose IP is "X.X.21.X" has been attacked in a concentrated manner, and the attackers are well-targeted. While some other servers (*e.g.,* "X.X.76.X", "X.X.69.X", "X.X.241.X") were attacked indiscriminately and the topological relationships are relatively scattered. Under-identification of DDoS attack traffic may cause bandwidth congestion and missing some damaged nodes, and the detection results of FOSS provide significant help to a certain extent. In addition, profiling the used attack techniques and the compromised source address could benefit in characterizing the portrait of attacking organizations. We have started to utilize FOSS to support and enhance the corresponding research about attack organization portrays.

[13]https://github.com/Secbrain/FOSS/blob/main/evaluation/realworld/tuopu.png

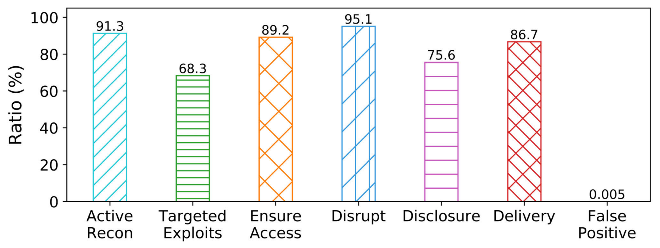


Fig. 18. The detection results (action intent and FP) for the APT scenario.

VII. Discussion

Feature Extraction. In the real-world test, we observe that FOSS could not distinguish different HTTP Flood attacks due to lacking the feature about fields of application layer protocols. A very recent art nPrint [38] proposes a unified expression for the common protocols and automated extraction selection methods. FOSS could leverage it to improve the feature extraction to capture more protocol details and build a more comprehensive feature vector.

Attack Category Recovery. When identifying fine-grained labels for unknown classes, it could occur to overestimate or underestimate the attack categories. We attribute it to two main reasons: (i) The extracted protocol features have different granularities. For example, some customers may need to distinguish between different HTTP flooding and some may not. Therefore, building a customized classification scheme in the output layer according to different needs may be beneficial to promoting FOSS to widespread use. (ii) The isolation clustering can be affected by the parameters more or less, some solutions might be effective, *e.g.,* sampling techniques for extremely unbalanced data. We will investigate these to advance the practicality of FOSS.

Scenario Extension. In addition to DDoS and intrusion detections, FOSS can be extended to more application scenarios. Considering the advanced persistent threat (APT) dataset usually includes unknown attacks, we conduct extended experiments based on the Collegiate Cyber Defense Competition (CCDC) dataset [39] (which involves multiple attack activities in the kill chain, the ground-truth labels reference the results of rule alarms/logs [39], [40]). Among them, the benign traffic of ten randomly selected PCAPs is used to build 10 FOSS instances, and then another ten randomly selected PCAPs are tested. The identification strategy is consistent with § VI, *i.e.,* identify attack when all ten models report anomalies. The results are summarized in Figure 18. We find that FOSS realizes different detection ratios for various action intents. Particularly, FOSS detects 68.3% for "Targeted Exploits" given current feature extraction lacks application-level semantics (*e.g.,* HTTP fields). While FOSS identifies 95.1% "Disrupt" since it mainly involves network DoS. Meanwhile, FOSS still maintains low false positives (only ~0.005%) via multiple-model verifications. Overall, it is promising to use FOSS in more security scenarios.

Combination with Existing Works. There are some related studies proposed by the community that can be combined with FOSS. For example, Du et al. [17] propose the unlearning framework to explore lifelong anomaly detection problems. This is similar to the retiring mechanism of FOSS, both of which can effectively correct the model when false positives/negatives are revealed. Meanwhile, Unlearn [17] presents a selection method for important sample sets to maintain low storage overhead, which can be used into FOSS (*e.g.,* when legitimate traffic changes). Moreover, a series of research directions focus on the problem of concept drift [41], [42], [43], *e.g.,* spatial and temporal biases [44].

As a representative work, OWAD [45] explores the normality shift detection, explanation, and adaption for anomaly detection. We discuss some combining strategies for OWAD with FOSS. (i) OWAD determines drifts through output calibration and distribution comparison, which can also benefit FOSS to detect distribution shifts. (ii) OWAD's explanation focuses on the normality shift (tracking the most influential samples), our interpretability considers feature-level contribution for prediction results (§ V-D), and they can complement each other. (iii) For the model update, OWAD and FOSS follow different methods. OWAD adjusts the loss function to implement shift adaptation, while FOSS is essentially the tree-based architecture, which is pronely available to grow and retire/prune. Overall, FOSS is not against the current evolution trend (can be combined with existing works), and provides some novel perspectives for the anomaly detection landscape.

Limitations and Future Works. Our work has a few limitations. First, although FOSS devises the Monte Carlo method to search for feature selection, it will still suffer from a large number of invalid dimensions. Some pre-processing for dimensionality reduction and data cleaning should be considered in the improvement plans. Second, different customers may require various detection granularity, the future work may consider a customized scheme, *e.g.,* change the output layer of FOSS. Third, applying the automated feature extraction and model parameters tuning into FOSS will lead in a good direction. Fourth, to provide customers with more reliable protection, the powerful adversary using a combination of multiple attacks needs to be further studied. Finally, as part of future work, we would explore which components could run in parallel to maximize efficiency.

VIII. Related Work

Besides SOTA baselines in § V-A, we list briefly some related work.

NIDS with Known Attacks Classification. To classify known attacks, some works [3], [4], [5], [6] design NIDSs based on statistical features by supervised learning methods, *e.g.,* random forests, deep neural networks [22]. Some other arts utilize Markov [46] or recurrent neural networks [20] to portray the sequential features (*e.g.,* packet length sequence) for attacks. While these methods are less suitable for detecting unknown attacks.

NIDS with Unknown Attacks Detection. These methods mainly involve three types of technologies: unsupervised, semi-supervised and zero-shot learning. (i) *Unsupervised learning methods* such as clustering algorithms (*e.g.,* K-means [19] and DBSCAN [25]) have been applied to identify outliers in network traffic. They are also known as "zero-positive" learning [17], [18] due to solely using benign samples for training. (ii) *Semi-supervised learning methods* such as Cls-Anomaly [9], FARE [10], and SENC [28] are usually composed of unsupervised and supervised learning. (iii) *Zero-shot learning methods* such as ZSL and GZSL have been used to classify unknown classes in NIDS [47]. With the non-incremental learnability, and the need for rich "side information" to construct the feature mapping, ZSL/GZSL methods are not suitable for our problem. Overall, their focus is different from ours, FOSS devotes to fine-grained detection and ever-changing legitimate traffic adaption.

Sample/Distribution Drift in Anomaly Detection. Furthermore, concept drift is also an important research problem in this landscape [41], [42]. TESSERACT [44] reveals the "spatial bias" and "temporal bias" in malware classification. Meanwhile, TESSERACT introduces a new metric for classifier robustness and presents an algorithm to tune its performance. A recent work, QWAD [45], studies the normality shift detection, explanation, and adaptation for anomaly detection. We discuss some combination strategies with existing works in § VII.

Some Recent Advances for NIDS. Security communities propose a series of advanced research directions for intrusion detection including solutions based on DPDK [23] or programmable switches [14], [49], [50] to adapt to high-speed bandwidth. Among them, deploying the tree-based model is a common solution whether it is a software platform [23] or a hardware primitive [50], so FOSS is promising to advance the in-network traffic anomaly detection in high-speed bandwidth. Leveraging formal verification to analyze the security of NIDS [33]. And some research devoted the automated characterization [38] and the interpretability for NIDS [18]. It is potential to combine these related studies with FOSS to explore aspects of robustness, feature selection scalability, and model interpretability.

IX. Conclusion

This paper presents FOSS, a fine-grained NIDS towards identifying both known/unknown attack types, as well as supporting adapting to variable legitimate traffic in an incremental manner. Based on our proposed three key norms, we implement FOSS and extensively evaluate it on the public dataset and real-world test. Moreover, we produce a series of experiments for FOSS in terms of robustness, adversarial attacks, interpretability and feature perception, etc. We demonstrate the effects of FOSS outperforming existing SOTA methods and its availability in practice.

Acknowledgment

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

References

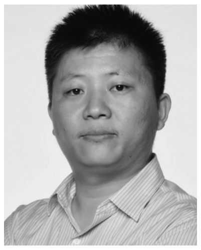
[1] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. CCS*, Nov. 2021, pp. 3431–3446.

[2] H. Li, H. Hu, G. Gu, G.-J. Ahn, and F. Zhang, "VNIDS: Towards elastic security with safe and efficient virtualization of network intrusion detection systems," in *Proc. CCS*, Oct. 2018, pp. 17–34.

[3] A. Saha, N. Ganguly, S. Chakraborty, and A. De, "Learning network traffic dynamics using temporal point process," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 1927–1935.
[4] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1147–1161, Apr. 2017.
[5] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "EBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.
[6] T. Shapira and Y. Shavitt, "FlowPic: Encrypted internet traffic classification is as easy as image recognition," in *Proc. IEEE INFOCOM Workshops*, Apr. 2019, pp. 680–687.
[7] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. NDSS* Reston, VA, USA: The Internet Society, 2018, pp. 1–15.
[8] P.-F. Marteau, "Random partitioning forest for point-wise and collective anomaly detection—Application to network intrusion detection," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2157–2172, 2021.
[9] J. Yang, X. Chen, S. Chen, X. Jiang, and X. Tan, "Conditional variational auto-encoder and extreme value theory aided two-stage learning approach for intelligent fine-grained known/unknown intrusion detection," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3538–3553, 2021.
[10] J. Liang, W. Guo, T. Luo, V. Honavar, G. Wang, and X. Xing, "FARE: Enabling fine-grained attack categorization under low-quality labeled data," in *Proc. NDSS*. Reston, VA, USA: The Internet Society, 2021.
[11] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*. Washington, DC, USA: IEEE Computer Society, May 2010, pp. 305–316.
[12] FireEye. (2020). *M-Trends Reports: Insights Into Today's Breaches and Cyber Attacks*. [Online]. Available: https://content.fireeye.com/m-trends/rpt-m-trends-2020
[13] Z. Zhao, Z. Li, Z. Song, W. Li, and F. Zhang, "Trident: A universal framework for fine-grained and class-incremental unknown traffic detection," in *Proc. WWW*, May 2024, pp. 1608–1619.
[14] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. NDSS*. Washington, DC, USA: The Internet Society, 2020, pp. 1–18.
[15] Z. Liu et al., "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2021, pp. 3829–3846.
[16] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Proc. NDSS*. Reston, VA, USA: The Internet Society, 2014, pp. 1–15.
[17] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proc. CCS*, Nov. 2019, pp. 1283–1297.
[18] D. Han et al., "DeepAID: Interpreting and improving deep learning-based anomaly detection in security applications," in *Proc. CCS*, Nov. 2021, pp. 3197–3217.
[19] J. A. Hartigan and M. A. Wong, "A K-means clustering algorithm," *J. Roy. Stat. Soc. C, Appl. Statist.*, vol. 28, no. 1, pp. 100–108, 1979.
[20] Z. Zhao et al., "ERNN: Error-resilient RNN for encrypted traffic detection towards network-induced phenomena," *IEEE Trans. Depend. Sec. Comput.*, early access, 2023, doi: 10.1109/TDSC.2023.3242134.
[21] Z. Song et al., "I^{2}RNN: An incremental and interpretable recurrent neural network for encrypted traffic classification," *IEEE Trans. Depend. Sec. Comput.*, early access, 2023, doi: 10.1109/TDSC.2023.3245411.
[22] Z. Zhao et al., "CMD: Co-analyzed IoT malware detection and forensics via network and hardware domains," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5589–5603, May 2024.
[23] Z. Zhao, Z. Liu, H. Chen, F. Zhang, Z. Song, and Z. Li, "Effective DDoS mitigation via ML-driven in-network traffic shaping," *IEEE Trans. Depend. Sec. Comput.*, early access, 2024, doi: 10.1109/TDSC.2023.3349180.
[24] S. Aryal et al., "Improving iForest with relative mass," in *Proc. PAKDD*, vol. 8444. Cham, Switzerland: Springer, 2014, pp. 510–521.
[25] M. Ester et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD*. Palo Alto, CA, USA: AAAI Press, 1996, pp. 226–231.
[26] F. T. Liu, K. M. Ting, and W. Fan, "Maximizing tree diversity by building complete-random decision trees," in *Proc. PAKDD*, vol. 3518. Hanoi, Vietnam: Springer, 2005, pp. 605–610.
[27] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *Proc. ICDM*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 413–422.
[28] X. Mu, K. M. Ting, and Z.-H. Zhou, "Classification under streaming emerging new classes: A solution using completely-random trees," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 8, pp. 1605–1618, Aug. 2017.
[29] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *Proc. IEEE Symp. Secur. Privacy*. Washington, DC, USA: IEEE Computer Society, May 2015, pp. 463–480.
[30] Canadian Institute for Cybersecurity. (2018). *Intrusion Detection Evaluation Dataset (CICIDS2017)*. Accessed: Nov. 27, 2020. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html
[31] G. Draper-Gil et al., "Characterization of encrypted and VPN traffic using time-related features," in *Proc. ICISSP*. Setúbal, Portugal: SciTePress, 2016, pp. 407–414.
[32] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.
[33] Z. Wang et al., "SymTCP: Eluding stateful deep packet inspection with automated discrepancy discovery," in *Proc. NDSS*. Reston, VA, USA: The Internet Society, 2020, pp. 1–17.
[34] O. Alrawi et al., "The circle of life: A large-scale study of the IoT malware lifecycle," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2021, pp. 3505–3522.
[35] VirusTotal. (2020). *Virustotal*. [Online]. Available: https://www.virustotal.com/gui/home/upload
[36] Z. Zhao et al., "DDoS family: A novel perspective for massive types of DDoS attacks," *Comput. Secur.*, vol. 138, Mar. 2024, Art. no. 103663.
[37] X. Ling et al., "DDoSMiner: An automated framework for DDoS attack characterization and vulnerability mining," in *Proc. ACNS*, vol. 14584. Cham, Switzerland: Springer, 2024, pp. 283–309.
[38] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proc. CCS*, Nov. 2021, pp. 3366–3383.
[39] F. Hassanabad. (2018). *CCDC Dataset*. [Online]. Available: https://github.com/FrankHassanabad/suricata-sample-data
[40] A. Nadeem, S. Verwer, S. Moskal, and S. J. Yang, "Alert-driven attack graph generation using S-PDFA," *IEEE Trans. Depend. Sec. Comput.*, vol. 19, no. 2, pp. 731–746, Mar./Apr. 2022.
[41] R. Jordaney et al., "Transcend: Detecting concept drift in malware classification models," in *Proc. 26th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2017, pp. 625–642.
[42] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending TRANSCEND: Revisiting malware classification in the presence of concept drift," in *Proc. SP*, May 2022, pp. 805–823.
[43] L. Yang et al., "CADE: Detecting and explaining concept drift samples for security applications," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2021, pp. 2327–2344.
[44] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2019, pp. 729–746.
[45] D. Han et al., "Anomaly detection in the open world: Normality shift detection, explanation, and adaptation," in *Proc. NDSS* Reston, VA, USA: The Internet Society, 2023, pp. 1–18.
[46] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 781–789.
[47] J. L. R. Pérez et al., "A Grassmannian approach to zero-shot learning for network intrusion detection," in *Proc. ICONIP*, vol. 10634. Guangzhou, China: Springer, 2017, pp. 565–575.
[48] Z. Li et al., "MetaNet: Interpretable unknown mobile malware identification with a novel meta-features mining algorithm," *Comput. Netw.*, vol. 250, Aug. 2024, Art. no. 110563.
[49] Z. Zhao, Z. Li, Z. Song, and F. Zhang, "Work-in-progress: Towards real-time IDS via RNN and programmable switches co-designed approach," in *Proc. RTSS*, Dec. 2023, pp. 1–4.
[50] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ML-based network security applications," in *Proc. NDSS* Reston, VA, USA: The Internet Society, 2021, pp. 1–18.

Ziming Zhao (Student Member, IEEE) is currently pursuing the Ph.D. degree with Zhejiang University, Hangzhou, China. He has published more than ten papers in international journals and conference proceedings, including IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON MOBILE COMPUTING, CCS, RTSS, INFOCOM, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, ESE, COSE, and AAAI. His research interests include machine learning, traffic identification, AI security, and privacy-preserving.

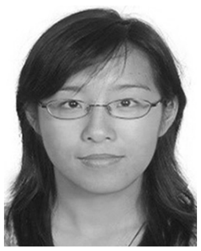
Zhaoxuan Li (Student Member, IEEE) is currently pursuing the Ph.D. degree with the State Key Laboratory of Information Security (SKLOIS), Institute of Information Engineering (IIE), Chinese Academy of Sciences (CAS), Beijing, China. He has published more than ten papers in international journals and conference proceedings, including IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON MOBILE COMPUTING, ESE, COMNETS, and ICWS. His research interests include blockchain security, formal methods, and traffic identification.



Xiaofei Xie received the B.E., M.E., and Ph.D. degrees from Tianjin University. He is currently an Assistant Professor with Singapore Management University, Singapore. His research mainly focuses on program analysis, traditional software testing, and quality assurance analysis of artificial intelligence. He was a recipient of the three ACM SIGSOFT Distinguished Paper Awards in FSE'16, ASE'19, and ISSTA'22.

Jiongchi Yu received the bachelor's degree from Zhejiang University, China. He is currently pursuing the Ph.D. degree with the School of Computing and Information Systems, Singapore Management University (SMU), Singapore. His research focuses on traditional software testing and security issues of cloud native infrastructures.

Fan Zhang (Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, University of Connecticut, CT, USA, in 2011. He is currently a Full Professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include system security, hardware security, network security, cryptography, and computer architecture.



Rui Zhang is an Associate Researcher with SKLOIS, Institute of Information Engineering, Chinses Academy of Sciences (CAS), China. She was a Post-Doctoral Researcher with the Institute of Software, CAS. She was a Visiting Scholar with Georgia Institute of Technology, from 2009 to 2010 and 2018 to 2019. She has published more than 40 technical papers in international journals and conference proceedings. Her research interests include blockchain security, security protocol, and applied cryptography.

Binbin Chen (Member, IEEE) received the B.Sc. degree in computer science from Peking University and the Ph.D. degree in computer science from the National University of Singapore. Since July 2019, he has been an Associate Professor with the Information Systems Technology and Design (ISTD) Pillar, Singapore University of Technology and Design (SUTD). He currently holds a joint appointment as a Principal Research Scientist with the Advanced Digital Sciences Center, which is the University of Illinois Research Center in Singapore. His current research interests include wireless networks, cyber-physical systems, and cyber security for critical infrastructures.

Xiangyang Luo received the B.S., M.S., and Ph.D. degrees from the State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China, in 2001, 2004, and 2010, respectively. He is currently a Professor with the Key Laboratory of Cyberspace Situation Awareness of Henan Province. He is the author or co-author of more than 150 refereed international journals and conference papers. His research interests are multimedia security and network security.

Ming Hu (Member, IEEE) received the B.E. and Ph.D. degrees from East China Normal University, Shanghai, China, in 2017 and 2022, respectively. He is currently a Research Fellow with Nanyang Technological University (NTU), Singapore. His research interests include the area of design automation of cyber-physical systems, federated learning, program synthesis, and software testing.

Wenrui Ma was born in 1985. She received the Ph.D. degree in computer science from Florida International University in 2018. She is currently an Assistant Professor with the School of Computer Science and Technology, Zhejiang Gongshang University. Her research interests include high performance network architecture and network security.