Journal of Information Security and Applications 68 (2022) 103248

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Multi-datasource machine learning in intrusion detection: Packet flows, system logs and host statistics

Ying-Dar Lin [a], Ze-Yu Wang [a], Po-Ching Lin [b,*], Van-Linh Nguyen [b], Ren-Hung Hwang [b], Yuan-Cheng Lai [c]

[a] *Department of Computer Science, National Yang-Ming Chiao-Tung University, Hsinchu City 300, Taiwan*
[b] *Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi County 621, Taiwan*
[c] *Department of Information Management, National Taiwan University of Science and Technology, Taipei City 106, Taiwan*

ARTICLE INFO

Keywords:
System log
Traffic flow
Host statistics
Machine learning
Feature engineering
Intrusion detection

ABSTRACT

This work compares the performance of different combinations of data sources for intrusion detection in depth. To learn and distinguish between normal and malicious behavior, we use machine learning algorithms and train three typical models on three kinds of datasets: system logs, packet flows and host statistics. Unlike other studies, our study captures and monitors the behavior from multiple data sources in order to catch security attacks. Our aim is to figure out how to build the most effective dataset for machine learning with a combination of multiple sources. However, since there are no such datasets which have been generated from multiple sources for given attacks, we show how to build and generate a dataset with three data sources. We then compare the F1 score of the detection by applying machine learning algorithms for various combinations of the data sources. Our evaluation results show that the dataset of host statistics results in better performance (0.91) than traffic flows (0.63) and system logs (0.44) because it has the highest average F1-score in the three stages of attacks, while the other datasets may have poor F1-scores in some of the stages, particularly in the stage of impact. However, in the initial access stage of attacks, the dataset of logs performs the best (0.94), and the packet flows are suitable for detecting network DoS attacks (0.82). Furthermore, running this detection with all three data sources results in minor overheads of at most 2.1% CPU utilization. Finally, we analyze the important features of each model, such as the number of logs generated by `apache-access`, `in.telnetd` and `postfix` in the dataset of logs, SrcBytes and TotBytes in the dataset of flows, and MINFLT, VSTEXT and RSIZE in the dataset of statistics.

1. Introduction

The rapid development of networking technology in recent years has seen a significant boom in connected devices. However, this also poses threats to network infrastructure because widespread connected devices also increase the surfaces for attackers to exploit numerous vulnerabilities over a wide area. For example, many of such connected devices are low-cost ones that are subsumed under Internet of Things (IoT). Because they are often engineered for a dedicated purpose and may have limited resources such as energy and computation allocated to sufficient security protection, they are likely to become good targets for attackers [1]. The existence of vulnerabilities in a large-scale network of connected devices exacerbate the havoc that attackers can wreak [2]. Damages to the connected frameworks, such as critical infrastructure like power plants, may sometimes even cost lives [3]. To mitigate such security issues, deploying an intrusion detection system (IDS) is the most common practice.

As its name suggests, IDS refers to a security system that detects intrusions into a protected network or system. There are two major detection approaches: signature-based and anomaly-based detection. The former looks for known signatures of intrusions, while the latter looks for anomaly deviated from the normal model. As a result of fast mutation of malware and attack variants, signature-based methods eventually fail to identify a large number of new attacks, particularly zero-day ones. Anomaly-based methods can identify attacks by monitoring exceptions or significant deviation of activities from a given normal profile [4]. However, the challenge is how to sufficiently define a normal profile, given the diversity and complexity of computer networks. Machine learning (ML) promises to be the tool to tackle the problem because of its ability to learn the complexity of a system and network activities [5–7].

* Corresponding author.
E-mail address: pclin@cs.ccu.edu.tw (P.-C. Lin).

https://doi.org/10.1016/j.jisa.2022.103248

Available online 6 July 2022

Machine learning typically needs a representative dataset to train a specified model. In the context of ML-based intrusion detection, the dataset can be derived from two major sources: external and internal activities of the hosts. External activities are observed primarily from packet flows, while internal activities are recorded in the system logs generated from the services running on the host's system and the host's statistics of process activities. Such system logs contain a sequence of commands and actions carried out by the users or processes, and the host statistics include numeric data about the resource usage on the system. These internal activities provide useful information that cannot be found in network traffic, such as memory exploitation, and leveraging such information on a host-based IDS (HIDS) can complement the detection on a network-based IDS (NIDS). Thus, both IDS types can work together. An NIDS can be deployed at the network border to protect the hosts in the network, and can correlate the alerts from different hosts for a global view of malicious activities (e.g., to detect port sweeping). An HIDS is good at detecting malicious system activities at the host level, such as disk wipe.

Because attacks may involve various exploitation techniques, to increase the reproducibility of this work, we will describe each attack scenario precisely, so that others can reproduce our work. We use a set of definitions termed ATT&CK matrix [8], which involves predefined tactics and techniques organized by the MITRE cooperation and help to structure the attacks accurately. For example, in the ATT&CK definition, the attack patterns of a Mirai botnet [9] can be split into three stages, initial access, command and control, and impact. Note that a separate data source is preferred for each stage. For example, in the initial access stage, system logs are preferred, but in the command-and-control stage, packet flows are the priority selection. Finally, in the attack stage, statistics data are dominant. The goal of this work is to investigate whether detecting attacks from a combination of different data sources is better than from a single source or not in a realistic environment. For this purpose, we create a dataset from multiple data sources, and propose an ML-based detection approach that can feed and process data from the sources to empirically evaluate the detection performance.

The process of ML-based multi-datasource IDS is at the early stage of development. Some anomaly-based IDSs target either external behaviors [5,10] or internal behaviors [6,7]. However, using only a single data source in such studies may not help much in the case of complicated attacks, which may not leave key footprints in that data source. We believe that multiple data sources can reveal important information about sophisticated attacks, e.g., an abnormal transition of behavior or a repetition of a suspect source over time. Although the idea of combining network and host information for intrusion detection is not novel [11], to the best of our knowledge, there is no previous work dedicated to assessing the detection with multiple empirical data sources in various attack stages.

In this work, we also analyze each ML model to determine the effective features and how they influence predictions. We also aim to determine the optimal tuning parameters that enhance system performance. By collecting three datasets and applying ML to them, we determine the most preferable combination of datasets for each attack stage. Note that, for fair comparison, we align and extract the data sources by timestamp, and then use our pre-trained models for a voting ensemble. Once we calculate the F1-scores of all the combinations, the evaluation results are compared in detail.

We conducted the experiments on a Python-based platform to verify our proposed method. The workflow was as follows. First, we tested seven combinations of data sources to see which combination yielded the best performance. Second, we tuned various hyper-parameters to enhance the performance and accelerate the speed of the overall system. Finally, we analyzed the features in the models to understand how they affected predictions.

The main contributions of this work are as follows:

- We are the first to empirically assess the performance of intrusion detection from multiple data sources, and also design the ML method that can effectively combine the features from these sources for the detection.
- The detection performance was evaluated in multiple attack stages, and the most effective features in individual stages were also investigated and underlined.
- The source codes and the created datasets are released for future studies in the research community.[1]

The rest of this paper is organized as follows. Section 2 reviews the background knowledge related to this work and provides a comparison between earlier works and our proposed solution. The problem statement is formulated in Section 3. Section 4 describes in detail our core multi-datasource ML solution. Section 5 presents a sequence of configurations and implementation. The experimental results and analysis are given in Section 6. The final section concludes this paper and considers future work.

2. Background and related work

In this section, we review the staged attack scenarios for collecting system behaviors. Then we briefly review XGBoost and ensemble, which are the algorithms for evaluating our datasets. Finally, we discuss earlier papers using ML and IDSs to predict anomalous behavior.

2.1. ATT&CK

Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) [8] is a knowledge base designed by the MITRE corporation. It is aimed at describing the behavior of attacks, as well as their target platforms, and provides a taxonomy for both offense and defense. It can also help people to emulate adversarial scenarios and test defenses against common attack techniques. This project involves some core components such as tactics and techniques. The tactics depict the goals of the stages of attack, and the techniques indicate ways to achieve those goals. Some tactics of ATT&CK are applied in this work, e.g., initial access, command and control, and impact. We adopt these tactics and techniques to design attack scenarios, and reproduce them on our testbed to collect various attack behaviors.

2.2. Attack scenarios

The ATT&CK model precisely defines the attack scenarios. The design of such scenarios should follow two principles: popularity and reproducibility. In the former, the scenarios should take place in the real-world; in the latter, the attacks can be replayed and give similar results.

Table 1 summarizes the five attack scenarios used in this work, which we separate into three main stages: initial access, command and control (C&C), and impact. Initial access is the first attempt to get into a target, and then we attempt to install a backdoor or create accounts to deliver commands to control the target for further usage. Finally, we force the compromised target to execute various commands that impact on itself or infect more targets. Each attack scenario is briefly described as follows.

- **Botnet.** A bonet consists of compromised connected devices, namely bots, which are controlled by external bot herders via the command and control (C&C) channel. Botnets are still common in the Internet. To replay this attack, we select the open-source malware, Mirai, which consists of the scanner/loader and the C&C server. The scanner uses telnet brute force to scan and try to access those devices with weak passwords. Once it logs

[1] The datasets and source code are available at https://bit.ly/3rbTbiN.

Table 1
Attack scenarios.

Attack scenario	Initial access	C&C	Impact
Botnet	Telnet brute force	C&C network	Network DoS
Ransomware	Vulnerability exploitation	Backdoor	Data encrypted for impact
Resource hijacking	Vulnerability exploitation	Backdoor	Monero mining
Disk wipe	Vulnerability exploitation	Backdoor	Disk wipe
Endpoint DoS	Vulnerability exploitation	Create accounts	Endpoint DoS

into the devices, it will use the loader to download a malicious binary executable. After the binary has been executed, the C&C server will deliver commands and control the injected devices. The victims may further be used to make some impact, such as launching network denial-of-service (DoS) attacks or infecting more targets in the network.

- **Ransomware.** As the scalability of connected devices or hosts becomes greater and the services they provide are closer to daily life, ransomware poses a serious threat to computer systems. Ransomware often encrypts sensitive files with cryptographic algorithms, and users need to spend an amount of money to retrieve their data.
- **Resource hijacking.** Resource hijacking leverages the resources of a victim's machine to obtain benefits. The most common situation is to garner virtual currency. Resource hijacking usually consumes significant system resources, and then degrades normal usage.
- **Disk wipe.** Disk wipe aims to delete important or sensitive files and directories on a target machine, and it may have serious impact if the wiped content such as master boot record (MBR) is related to core system functionality.
- **Endpoint DoS.** Endpoint denial of service aims to block the availability of normal service to users by exhausting a system's resources. Different from network DoS, endpoint DoS attacks may affect the normal usage of a service without saturating the network used to access the service.

2.3. XGBoost and ensemble

Once we have completed reproducing attacks and acquiring the datasets, we can build the ML models from these datasets. We select XGBoost as the training algorithm for its proven effectiveness and scalability. After the training phase, the ensemble will continue to predict anomalies, so that we can compare the performance of each combination of datasets. The following details XGBoost and ensemble.

- **XGBoost** [12] XGBoost, the abbreviation of eXtreme Gradient Boosting, is an advanced version of a gradient boosting algorithm for solving classification problems. Like its original version, XGBoost uses gradient descent to minimize loss errors, generates multiple weak prediction models, and combines them into an estimator. However, it enhances the original version by adding tree pruning and sparsity-aware split fitting for high computational efficiency. It also supports parallelization and is cache-aware. Furthermore, XGBoost uses shrinkage and column subsampling to further prevent over-fitting. It is also a scalable algorithm, so that more attack scenarios as well as data in the future can be added. By using XGBoost, we expect to be able to build robust models for three network or system behaviors, and use them for the further ensemble process.
- **Ensemble** [13] Ensemble is another common technique in ML. It combines multiple models for making decisions; it will thus minimize errors and obtain better prediction results. Despite many ensemble algorithms, we just focus on two of them in this work, boosting and stacking. Boosting generates multiple weak models, assigns them different weights, and sequentially combines them just like XGBoost. Stacking is much more intuitive since it collects the predictions from each model, and then combines them to make the ultimate decision. In this work, we use boosting to build the models and stacking with the decision strategy to make predictions. The reason for using boosting is to leverage multiple models to minimize errors and increase prediction accuracy. As for stacking, we use three models to represent three system or network behaviors to predict anomalies because we wish to test each type of behavior and compare the accuracy of each dataset combination.

2.4. SHAP [14]

SHAP (SHapley Additive exPlanations) is a game-theoretic method to explain models, and it can help understand why a model makes a particular prediction. SHAP assigns each feature a value of importance for a certain prediction, so that one can analyze the values to interpret the models. It also provides data visualization tools for generating a summary plot of a model. For an overview of the feature's importance, we can plot the SHAP value of every feature for every data point in the dataset. The colors of the data points represent the feature value, and a feature with a higher SHAP value tends to have a higher prediction value. We can calculate the mean absolute value of the SHAP value for each feature to obtain its importance value.

2.5. Related works

Table 2 summarizes the current solutions to intrusion detection with network or system behaviors by ML. The approaches are separated into three main categories based on the input of the model. For packet flows, we summarize some state-of-the-art solutions to traffic classification problems as well as their datasets and algorithms [5,10,15]. These ML-based algorithms have different ways of selecting features. Some works use pre-defined features such as packet statistics [16–18], while the others prefer to extract the features automatically. For example, Wang et al. [19] use CNN to learn the special features and then apply LSTM to learn temporal features. Hwang et al. [5] also uses CNN to extract such features and an autoencoder to learn benign traffic. These related studies use different algorithms to make predictions; some apply supervised learning algorithms like random forest [17] or SVM [18], while the others use unsupervised deep learning algorithms such as autoencoder [20].

The most significant works that leverage system logs preprocess raw logs in a similar way: parsing the logs into log templates and combining them into a sequence of templates based on the time or sequence number. The main difference between each work is the algorithms used. Some focus on unsupervised learning. For example, Xu et al. [21] uses principal component analysis to analyze feature vectors. Otomo et al. [23] applies CVAE to log time series data and derive latent variables, and then use them to predict anomaly, while Zhang et al. and Du et al. [6,24] leverage LSTM to learn the temporal information of a log sequence. Many approaches try to develop supervised learning methods. For example, Sheluhin et al. [22] apply three methods: decision-tree, k-NN, and SVM. By contrast, Du et al. [6] prefer to use a previous log sequence to predict the most likely next sequence. For host statistics, few papers discuss this topic because of the paucity of supported datasets. The most closely-related work is from Ham et al. [7], which vectorizes collected data as features and uses SVM to detect malware on an Android system.

Table 2
ML-based anomaly detection methods.

Data source	Paper	Dataset	Algorithm	Performance
Packet flow	[16]	Self-collected	Autoencoder	100% (TPR) 0.007 ± 0.01 (FPR)
	[20]	NSL-KDD CICID 2017	Autoencoder	99% (Acc)
	[17]	ISCX 2012	Word embedding, CNN, RF	99% (Acc)
	[19]	DARPA 1998 ISCX 2012 FPR	CNN+LSTM	99% (Acc)
	[18]	KDD99	Autoencoder+SVM	95% (Acc)
	[15]	UNSW-NB15	Random forest	99.34% (Acc)
	[5]	USTC-TFC201 Self-collected	CNN+Autoencoder	0.99 (F1)
	[10]	CICIDS2017	Autoencoder + VAE	0.62 ∼ 0.98 (AUC)
System Log	[21]	HDFS, Darkstar	PCA, Decision Tree	69.6% (Acc)
	[22]	HDFS	SVC, DT, KNN	92% ∼ 99% (Acc)
	[23]	SINET4	CVAE + clustering	84.6% (Pre)
	[24]	HDFS Microsoft industrial dataset	Attention-based Bi-LSTM	0.81 ∼ 0.99 (F1)
	[6]	HDFS, OpenStack	LSTM	0.96 ∼ 0.98 (F1)
	[25]	HDFS, BGL	LSTM	0.95 ∼ 0.96 (F1)
	[26]	BGL	word2vec + RF	0.88 (F1)
Host statistics	[7]	Self-collected	SVM	99.9% (TPR) 0.4% (FPR)
All	Ours	Self-collected	XGBoost	Up to 0.99 (F1)

Historically, there are various favored datasets such as HDFS and KDD99 that feature in many studies. An HDFS dataset is a collection of system logs generated by applications running on Hadoop-based distributed file systems. It is labeled by domain experts and contains around 3% of abnormal data [27]. KDD99 is a widely-used dataset for evaluating intrusion detection methods [28]. It contains 41 features related to network traffic and records four kinds of attacks. We are also aware that there are several advanced techniques of feature characterization, representation and extraction presented in recent research studies [29–32]. Despite a number of existing datasets and methods, it is a challenge to figure out which combination of datasets yields the best detection results. Furthermore, no existing dataset satisfies our requirement of collecting datasets from multiple sources. In this work, we will not only build models to test three datasets simultaneously but also generate those datasets.

3. Problem on multi-source dataset comparison

In this section, we outline the notations used in this work and formalize the problem we wish to address.

3.1. Notations

Table 3 lists the notations applied in this work. We assume that an HIDS is able to monitor the packet flows, system logs, and host statistics on a system simultaneously, and detect attacks from the data with machine learning. Therefore, we collect three kinds of datasets in this work: packet flow dataset P, system log dataset L, and host statistics dataset S. Suppose that the data will be preprocessed and fed into the ML models for training and testing. We generate a dataset X by aligning and extracting the data points from the above three sources in combination, and separate the flows/rows by the timestamps of the data points in this combined dataset. The three pre-trained models are denoted by M^P, M^L, and M^S, and are derived from the ML-based algorithm, i.e., XGBoost in this work, in order to learn the behavior in the three datasets. Finally, we use a detection table to record the F1-scores of each data source for an attack scenario.

Table 3
Notation table.

Category	Notation	Description
Datasets	P	Packet flow dataset
	L	System log dataset
	S	Host statistics dataset
	X	The dataset from three sources
Models	M^P	Pre-trained packet flow model
	M^L	Pre-trained log model
	M^S	Pre-trained statistics model

3.2. Problem description

Given the multi-source datasets P, L, S and the three pre-trained ML models M^P, M^L, M^S, we aim to test on the data sources with seven combinations in total, which involve three models from a single data source, three models from two data sources in combination, and one from all the data sources. Based on the results we observed, the final task is to figure out which combination of data sources yields the best result. To avoid the impact of data imbalance problems, we fine-tune the parameters to balance the proportions of benign and malicious data in each dataset. Finally, we detail the hyper-parameter selection of the models to maximize detection performance.

4. Multi-datasource machine learning and ensemble method

In this section, we describe construction of our multi-datasource ML model. In Section 4.1, we give an overview of our proposal. The details of dataset preprocessing and aligned extraction are given in Sections 4.2 and 4.3. Feature importance and threshold decisions are covered in Section 4.4, and finally, Section 4.5 summarizes the extensive discussions about the ensemble method and the observed results.

4.1. Overview

This work develops a method to compare multiple data sources for ML-based detection. To compare the three data resources, we replayed

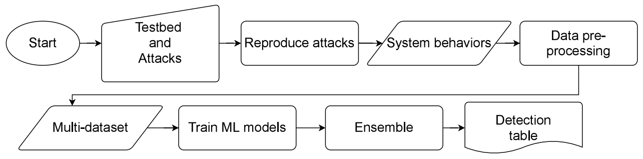


Fig. 1. Flow chart of this work.

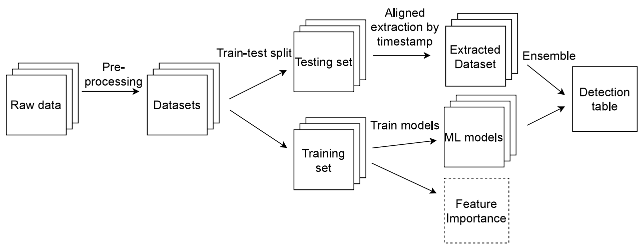


Fig. 2. Architecture overview.

attacks on the testbed and collected three kinds of data simultaneously. Fig. 1 presents the overview of the whole work, and Fig. 2 describes the architecture of the data flow. First, we built a testbed to generate the dataset from the three data sources, and then reproduced the attack scenarios described in Section 2 to capture the network or system behaviors. After collecting the data, we preprocessed and divided them into three datasets, and fed them into the ML models. The testing sets were then extracted from these three datasets after the preprocessing steps. We carried out multi-datasource ML and ensemble on the dataset with the aligned extraction (elaborated in Section 4.3) to evaluate the performance and efficiency of every possible combination of the data sources.

4.2. Data preprocessing

After replaying attacks on the testbed, we had three kinds of datasets: packet flow dataset P, system log dataset L, and host statistics dataset S. The data then had to be preprocessed before they were fed into the ML models. Below we will discuss in detail how this process works.

- **Packet Flow.** To acquire the packet flows, we recorded the packet information during attack reproduction and used a tool called `Argus` to generate flow information and divide the flows into subflows, based on *time windows*. The duration of data points in a subflow had to be within the time window size. We recorded and utilized only the packet headers; the header information was still available even if the payloads were encrypted. We used the features such as source and destination IP addresses to classify normal and malicious data and label them[2]; thus, these features could not be used in the training and thus should be filtered out. We pre-defined a set of features from the dataset and describe them in Section 5.3 If the flow features consisted of numeric values such as duration, we could directly apply such values; other features, e.g., features in strings, were transformed into integers by hashing. It is worth noting that to ensure the values of each feature were within the same scale, we applied min–max normalization to transform the data into values between 0 and 1.
- **System Log.** The first step of system log preprocessing is to separate the raw logs based on the hosts on which they are stored. The separation is based on the host field in the dataset. The next step is log parsing, which transforms the raw logs into log templates (see Table 4 in Section 5.3 for examples), in order to extract the common information from the raw logs and discard redundant information such as IP addresses or device identifiers in the logs. We used a tool called `drain` to parse the raw logs, and assign the corresponding log templates to each raw log. The final step is to generate counting vectors to record the number of log templates that occur within a given time slot (5s in this work). Each tuple of the counting vector represents the number of a certain log template, and the number of tuples is equal to that of unique log templates.
- **Host statistics.** We collected the host statistics related to the commands on the system within a fixed time slot. Preprocessing the statistics data is quite similar to that of packet flows, e.g., filtering out unusable features, and the features with numeric values will be directly fed to the model. We also applied normalization on the statistics data for the same reason as that for the packet flows.

4.3. Dataset with aligned extraction

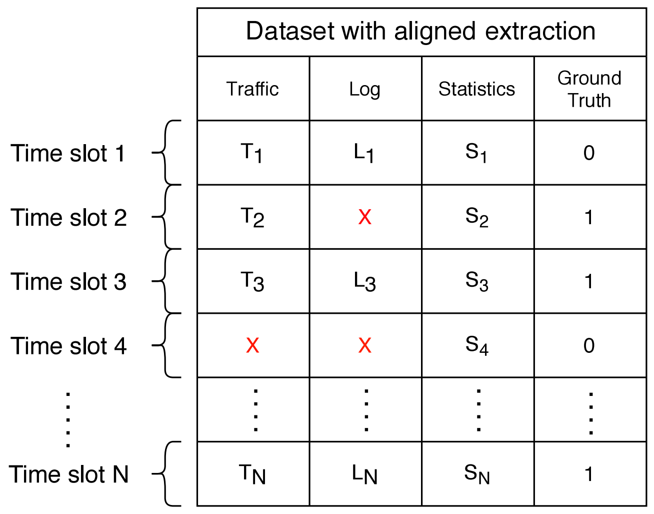
The goal of this work is to compare the ability of attack detection for each combination of data sources. If we test each dataset separately, differences in the time scale or the amount of data may affect the result. Thus, we may not know how each attack stage affects the overall detection performance; existing datasets have such key constraints. To obtain reasonable results, we compared the datasets within the same time scale as well as in the same attack scenario and stage. Through the separate collection of each dataset, we have the detail with each stage information as well as with similar timestamps. Suppose that the three datasets P, L and S are split into training sets and testing sets at a ratio of 70–30. We align and extract the testing sets of the three data sources by timestamp to generate a dataset, denoted by X. Fig. 3 illustrates the concept of the aligned extraction. In this process, we can make predictions in the time slots. If a value of some data source is missing in a time slot (e.g., the system is under attack but no data remain in the packet log), the missing values may degrade the performance of the learning model. Given the accurate launch time of each attack, we can label the attack activities by their timestamp for the ground truth of each data point. We then use the training datasets to train the three models M^P, M^L, and M^S. As a result of the use of XGBoost, these three models form a tree structure that looks like a decision tree, but is more complex. Finally, we use these models on the dataset X to compare the results. The detail of how to use the generated dataset for ensemble is described in Section 4.5

4.4. Feature importance and threshold decision

After training the three models, we could evaluate the importance of the features. We leveraged SHAP [14] to assign an importance value to each feature to know which feature affects the prediction most and how the features influence the predictions. Initially, we calculated the SHAP value of each data point in each feature, and then took the mean absolute of the values to obtain the importance value of each feature. Then we inspected the SHAP values of the data points in the feature. If the data points with large feature values had large SHAP values, the features tended to be positive for the prediction.

Other than feature importance, we also had to decide the thresholds for the three models. Because the objectives of these models are logistic regressions, their outputs are probabilities of whether the input is positive or not to the prediction. For this purpose, we mapped a logistic regression value to a binary category. If the output value was larger than the specified threshold, the data point was considered being from malicious behavior. Moreover, the proportion of benign data and malicious data in the training dataset can significantly influence the value of threshold. Also, since the amount of training data is different for each data source, the threshold of prediction for each model may be

[2] The labeling was feasible because the IP addresses of the attackers and the victim were known on the testbed.



	Dataset with aligned extraction			
	Traffic	Log	Statistics	Ground Truth
Time slot 1	T_1	L_1	S_1	0
Time slot 2	T_2	X	S_2	1
Time slot 3	T_3	L_3	S_3	1
Time slot 4	X	X	S_4	0
⋮	⋮	⋮	⋮	⋮
Time slot N	T_N	L_N	S_N	1

Fig. 3. Dataset with aligned extraction, where × in red means missing values.

different. For a fair comparison, we determined the thresholds which made the most accurate predictions with the best F1-score for each model by using grid-search on the pre-trained models and the training dataset. The search will try multiple threshold values and determine which threshold yields the optimal F1-score on the training dataset.

4.5. Ensemble

Each data point in the dataset after the aligned extraction involves the values from the three data sources in each time slot. We fed the values from each data source into its corresponding pre-trained model and recorded the prediction value. If the value was larger than the threshold for this model, the prediction result would be regarded as malicious, meaning that the value is related to malicious behavior. If there was any malicious prediction within a time slot, the whole data point in the time slot would be from malicious behavior; otherwise, it was from normal behavior. If a data source was missing in a time slot, we marked the prediction of this data source in that time slot as normal. Thus, each data point would result in up to three prediction results in the three models, and we used the ensemble method to determine the final result. For each combination of data sources, if one data source yielded malicious prediction, then the whole data point would be classified as malicious. For example, as Fig. 4 shows, for the three data sources in time slot 2, the prediction of the data point in the dataset was marked as malicious if one of the data sources produced malicious prediction. This design is so because we consider false negatives (i.e., malicious data points predicted as normal ones) as more serious than false positives (i.e., normal data points predicted as malicious ones). After getting all the prediction results, we used these values and the ground truth to evaluate the F1-scores, which are recorded in a detection table. In this table, each row represents a combination of data sources, and each column represents an attack stage. From this table, we can then identify the ability of each combination of data sources to detect the attack stages.

5. Implementation

In this section, we describe the implementation of this work in detail. In Section 5.1, we detail our testbed and the way to reproduce attacks on it. The data collection and preprocessing methods are described in Section 5.2. In Section 5.3, we interpret how to use the datasets to train the ML models. The open-source tools and modules used in this work are explained in Section 5.4

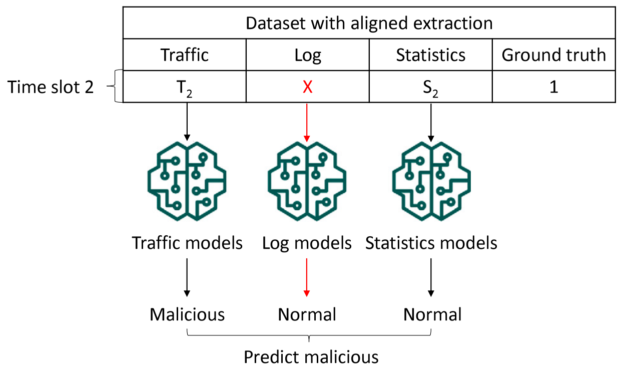


Fig. 4. The ensemble approach for each time slot.

5.1. Attack reproduction

Fig. 5 is the overview of our testbed. An attacker server is responsible for launching attacks and sending commands to the victims, and a vulnerable target server is under attack. We also have some vulnerable devices that can be exploited by attacks and some non-vulnerable devices that are immune to attacks. All the systems of the devices are Unix-based. To simulate benign data, we have a benign server running normal services such as a web application and an email service that interact with both vulnerable and non-vulnerable devices. Finally, a data logger server aggregates the three data sources from each device, and carries out data-preprocessing and ML. All the machines are virtual ones for better reproducibility in this work.

After preparing the testbed, we then replayed the following attacks on it. The attack scenarios are introduced as follows:

- **Botnet.** First, we created a C&C server of the Mirai botnet on the attacker server, and then scanned the vulnerable devices with weak telnet passwords. Once the credentials of devices were found, Mirai then injected a malicious binary and executed it. When enough bots have been exploited, the attack server would give commands to those bots to launch a DDoS attack on the target machine. We recorded the timestamp of each attack stage and labeled the system behaviors under attacks.
- **Ransomware.** We ran open-source Bash-ransomware to emulate the attack. First, we exploited a vulnerability by `unix/unreal_ircd_3281_backdoor`, a Metasploit module, on an IRC service to access the target, and then exploited a docker daemon vulnerability by `linux/local/docker_daemon_privilege_escalation` to perform privilege escalation. We inserted a backdoor after obtaining a root account. Finally, we transferred and executed the malicious binary code to encrypt files based on the pre-defined file extensions. After the encryption phase, users on the victim machines would receive a website link for paying the ransom.
- **Resource hijacking.** In this scenario, we first exploited a vulnerability on the Apache Continuum service by `linux/http/apache_continuum_cmd_exec` to obtain access to the target, and then inserted a backdoor by `linux/local/service_persistence` into it. Finally, we transferred and execute the mining program to dig Monero currency.
- **Disk Wipe.** In this scenario, we exploited the existing Ruby on Rails web application vulnerability by `multi/http/rails_secret_deserialization` on the target machine to obtain access to it with the root privilege, and then inserted a backdoor by `linux/local/service_persistence` for further control. Finally, we downloaded the disk wipe program and then executed it to delete some important files and directory. In this case, we deleted the entire `/boot` directory.

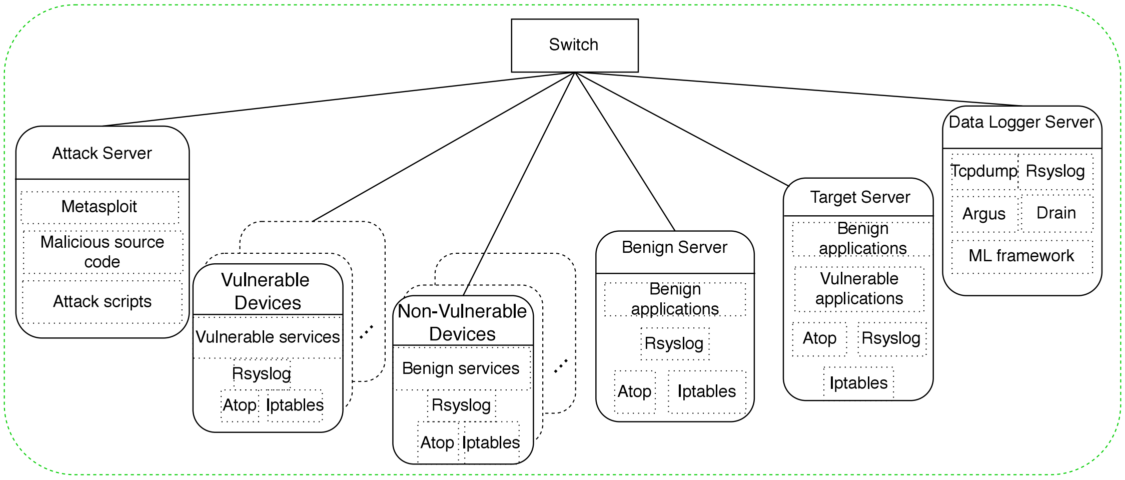


Fig. 5. The testbed architecture.

- **Endpoint DoS.** In this case, we exploited a vulnerability on the IRC service by `unix/unreal_ircd_3281_backdoor` and the docker daemon by `linux/local/docker_daemon_privilege_escalation` again to get the root account. After obtaining this account, we created an account on the victim machine, posed as a normal user, and executed a malicious fork bomb to hinder the normal process.

Note that it is possible to select different exploits and malware in the above attack scenarios (e.g., different types of botnets), and the above selections inevitably lead to some bias in the generated datasets. However, given numerous exploits and malware samples in the wild, it is difficult to exhaustively try them all. Despite this limitation, we believe that our selections can still represent certain common types in the attack scenarios, and it is definitely beneficial to cover more exploits and malware in the future work.

5.2. Data collection and extraction

When launching attacks on the testbed, we used existing tools to collect both benign and malicious data. For packet flows, we collected bi-directional session flows in the PCAP format to avoid duplication in both flow directions. We collected the system logs from multiple devices or hosts, and aggregated them into one machine. The host statistics were derived from the information of each process in a time slot.

After the collection, we extracted useful information from the raw data. For packet flows, we used existing tools to transform PCAP files into 5-tuple information. For logs, we used a log parser to transform the raw logs into log templates for further usage. For host statistics, because the collection tool already provided the features, we skipped further preprocessing to generate the dataset. During the attack reproduction, we recorded timestamps in each attack stage, and then used the information to label data. Other than the timestamp, we also referred to various features such as IP addresses in the packet flows or commands in the host statistics to label data points because we reproduced attacks on our testbed, and knew which IP addresses or commands generate the abnormal data. Note that the features used to label data points would not be involved in the training phase because the model would learn the ground truth and predict perfectly.

5.3. Feature selection for machine learning

The selection of input features depends on the dataset and should be determined beforehand. We make the raw logs structural, and merge multiple logs into a sequence to make them contain temporal information. The input features of the logs are the number of each log template. Table 4 lists only some examples of log templates and their descriptions as there are 88 log templates in total. Because the other two datasets already have temporal information and numeric values, the only thing we do is pre-defining a set of features and transforming those features with non-numeric values into numeric ones. Tables 5 and 6 show the features for packet flows and host statistics. For these two datasets, we used almost every feature from the output of the data extraction phase to reduce the manual intervention of selection, and we also wanted to explore which features contributed more to the prediction.

5.4. Open source tools

Table 7 summarizes the five categories of open source tools in this work. For the data collection, we used `tcpdump` to capture the packet information and `rsyslog` as well as `atop` to aggregate the logs and accounting information on each machine. After collecting the data, we extracted meaningful features from the raw data by `Argus` to preprocess the raw packet flows in the PCAP format, and by `drain` [33] as the log parser to transform the raw logs into the log templates. We also used existing tools to build the environment for launching the attacks. For example, we leveraged `Metasploit` [34], which contains hundreds of modules for exploiting vulnerabilities of the target machines. Because it is beyond this work to find out the latest vulnerabilities on the existing applications or systems, we just used a system with existing vulnerabilities (see Table 8). `Metasploitable` is suitable for this purpose. It has Linux and Windows versions that provide many built-in vulnerabilities. Another tool, `pymetasploit`, can provide an interface to manipulate Metasploit by Python scripts. We used it to automate the process of attack reproduction, and this tool helped us to record the timestamp and label the data automatically. The next category is related to ML-related tools. We used existing functions in the `scikit-learn` modules to manipulate data such as splitting the training and testing datasets, and build the ML models with the XGBoost library. After training the models, we could leverage SHAP to analyze the features per model. Finally, we modified and compiled the malicious source codes such as Mirai botnet, Bash ransomware, `XMRig` and `wipe` at our custom configuration to obtain the malicious binaries.

6. Experimental results

In this section, we present the experimental results of this work. In Section 6.1, we cover the configurations of the experiments, and compare the internal and external behaviors in Section 6.2, and detail the important features and the SHAP values in each model in Sections 6.3 and 6.4.

Table 4
Examples of log templates.

Log template	Description
"GET ⟨*⟩ HTTP/1.1" 200 ⟨*⟩ "-" "python-requests/2.16.0"	Generated when `apache-access` receives requests
Connect from ⟨*⟩ (⟨*⟩)	Generated when `in.telnetd` is successfully connected
Connect from unknown [⟨*⟩]	Generated when `postfix` is successfully connected
FAILED LOGIN ⟨*⟩ on ⟨*⟩ from '⟨*⟩' FOR ⟨*⟩ Authentication failure	Generated when the login fails to authenticate

Table 5
Features of packet flows.

TotPkts	TotBytes	Dur	Mean	StdDev	Sum	Min	Max
SrcPkts	DstPkts	SrcBytes	DstBytes	Rate	SrcRate	DstRate	

Table 6
Features of host statistics.

PID	RDDSK	WRDSK	WCANCL	DSK	MINFTL	MAJFTL	VTEXT
VSIZE	RSIZE	VGROW	RGROW	MEM	TRUN	CPU	

Table 7
List of open-source tools.

Category	Name	Functionality
Data collection	tcpdump	Collect packet flow data
	rsyslog	Collect raw log data
	atop	Collect statistics data
Feature extraction	argus	Extract packet features
	drain	Parse raw logs into log templates
Attack framework	Metasploit	Provide modules to exploit vulnerabilities
	Metasploitable3	Provide service with built-in vulnerabilities
	pymetasploit	Provide an interface to control Metasploit
ML framework	xgboost	A framework to train and test ML models
	scikit-learn	Python module for machine learning
	SHAP	Game-theoretic method to explain ML models
Malicious code	Mirai	Famous botnet source code
	Bash ransomware	Simple bash cryptoware
	XMRig	High-performance Monero miner
	wipe	Erase files from magnetic media

6.1. Parameter configurations

The configurations of this work are as follows: the time slot of the dataset with aligned extraction, the hyper-parameters, and the prediction threshold of each model. The time slot length of the dataset X is set to 5 s. Table 9 summarizes the parameters for the model training in this work. The training in XGBoost is carried out over 100 epochs. To reduce the model's complexity and avoid over-fitting, we set the maximum depth of the model as 2. We assign a learning rate of 1. The objective is to use logistic regression for binary classification, and the output is the probability of abnormality. Finally, we also specify a built-in parameter in the XGBoost framework, $scale\_pos\_weight$, which controls the balance of normal and abnormal training weights by scaling the gradient for the malicious data points relative to normal ones. Here we set it as 10 to address the data-imbalance problem.

After training with each data source, we get three models and need to assign the best thresholds for each model. Table 10 shows the threshold and the corresponding F1-score of each model. We use grid search on the training dataset to evaluate the thresholds. For each type of model, the highest F1-score can reach almost 0.99, showing that all the models can learn from data and detect abnormality very well.

6.2. External vs. Internal behavior

After training three models, we then tested various combinations of the models on the dataset X. Fig. 6 shows the result of each F1-score on the stage basis. We used all the data from five attacks and three stages to train and test the models in the comparison. For each of the stages, we extracted the data from the stage in the five attacks and carried out the same procedure to get the results.

The first observation was that if we took more data sources as a reference, we got higher F1-scores (overall 0.97). As shown in the evaluation results (Fig. 6), the host statistics performed better (0.91) than the packet flows (0.63) and the system logs (0.44) because it has the highest average F1-score in the attack stages, while the other datasets may have poor F1-scores in some stages, particularly in the stage of impact. The result can be influenced by the number of malicious data points in the dataset X. From Tables 11 and 12, we can see that the caught malicious data points in the system logs were fewer than those in the packet flows and host statistics. In our evaluation metric, if there are no data from a data source in the dataset X, the model will predict normal. As a result, many false negatives occur on packet flows and logs. This is why we use the ensemble method as presented in Section 4.5 to avoid such false negatives. Fig. 6 also manifests the importance of combining the models from multiple data sources, which is a key contribution of this work.

Furthermore, we considered the overhead of each data source. As Fig. 7(a) shows, the statistics and flows generated the most data per second (192.34 KB/s and 172.4 KB/s, respectively). Although the total data volume was more than that from a single data source, the overhead in storage was negligible, since it was not necessary to keep the data after detection. The overhead of the CPU utilization in dealing with more data was also affordable. As presented in Fig. 7(b), the CPU usage of detection with each data source was at most 2.1%; thus, the data sources can be all used in the intrusion detection with ML. Another observation from the result is in the initial access stage, where the logs have the best performance (0.94) of the three data sources because the attack performs anomalous usages and triggers software or systems that generate logs in this phase.

Other than stage-based comparison, we also carried out an attack-based comparison. Fig. 8 shows the results of the five attacks. The outcome of this comparison is similar to the stage-based comparison. Although the F1-scores of the logs and the flows look low, the results are still acceptable. The low rate is inevitable because of the lack of malicious data points in the dataset X rather than the problem of their learning models. However, from the results, we can see that except statistics, packet flows perform well on the network DoS attacks from Mirai (0.82).

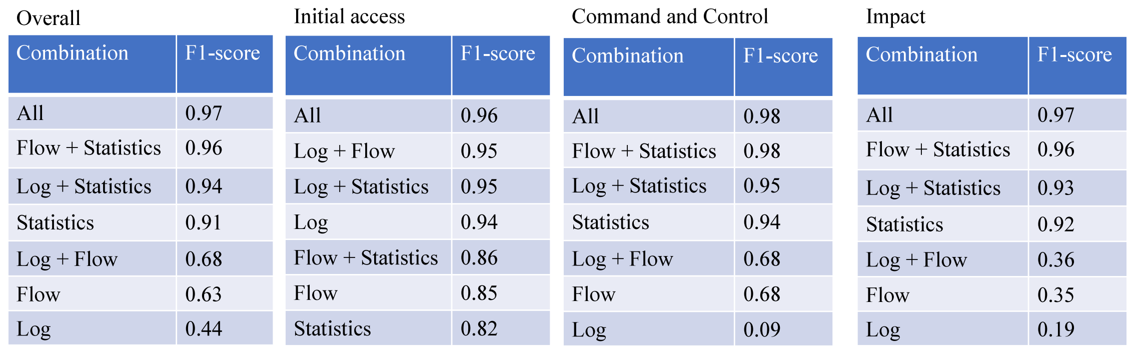
6.3. Important features in each model

After obtaining three models for each data source, we calculated and displayed the important features in each model. These are evaluated according to the absolute value of the SHAP values of the data points. SHAP can interpret the models and show which feature affects the prediction largely. If the importance value of a feature is larger than that of another, then the former feature is more important than the latter.

Fig. 9 (left) shows the feature importance of the log model. Because we used the number of log templates as features, the important features mean that the number of certain log occurrences can help predict anomalies. Both benign and malicious log templates can influence the prediction results. The three most important features in the

Table 8
List of Metasploit modules.

Category	Name	Functionality
Exploit	unix/unreal_ircd_3281_backdoor	Exploit a backdoor on unreal ircd
	linux/local/docker_daemon_privilege_escalation	Get root privileges from accounts accessing the docker daemon
	linux/http/apache_continuum_cmd_exec	Perform a command injection on Apache Continuum application
	linux/local/service_persistence	Insert a backdoor on target, and make it auto-restart
	multi/http/rails_secret_deserialization	Perform remote command execution on Ruby on Rails applications
	multi/handler	Generic payload handler
Payload	cmd/unix/reverse_perl	Create an interactive shell via Perl
	cmd/unix/reverse_python	Connect back and create a command shell via Python
	linux/x86/meterpreter/reverse_tcp	Remotely control the compromised system
	ruby/shell_reverse_tcp	Connect back and create a command shell via Ruby
Post	multi/manage/shell_to_meterpreter	Upgrade a command shell to meterpreter



Overall

Combination	F1-score
All	0.97
Flow + Statistics	0.96
Log + Statistics	0.94
Statistics	0.91
Log + Flow	0.68
Flow	0.63
Log	0.44

Initial access

Combination	F1-score
All	0.96
Log + Flow	0.95
Log + Statistics	0.95
Log	0.94
Flow + Statistics	0.86
Flow	0.85
Statistics	0.82

Command and Control

Combination	F1-score
All	0.98
Flow + Statistics	0.98
Log + Statistics	0.95
Statistics	0.94
Log + Flow	0.68
Flow	0.68
Log	0.09

Impact

Combination	F1-score
All	0.97
Flow + Statistics	0.96
Log + Statistics	0.93
Statistics	0.92
Log + Flow	0.36
Flow	0.35
Log	0.19

Fig. 6. Stage-based comparison.

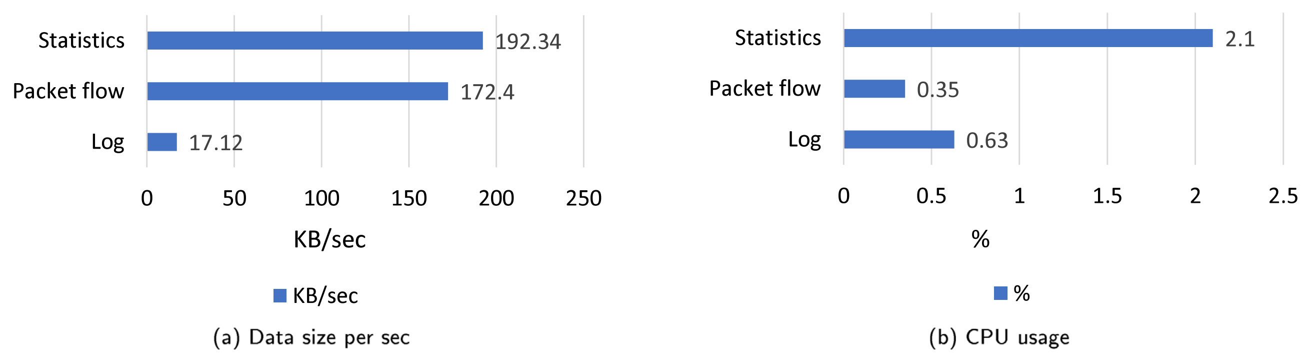
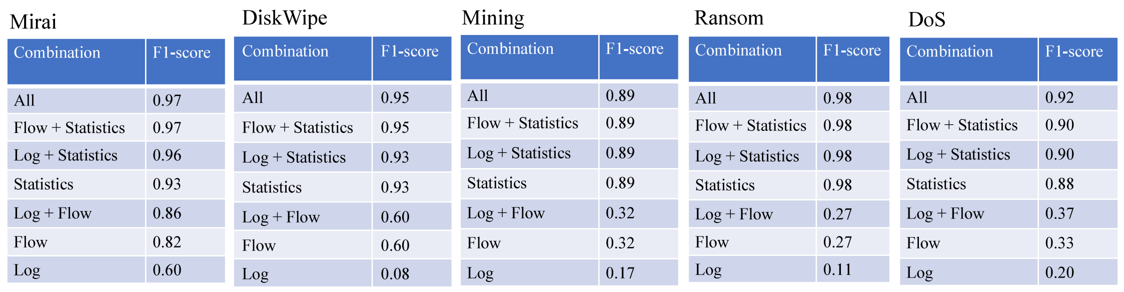


Fig. 7. Overhead of each data source.



Mirai

Combination	F1-score
All	0.97
Flow + Statistics	0.97
Log + Statistics	0.96
Statistics	0.93
Log + Flow	0.86
Flow	0.82
Log	0.60

DiskWipe

Combination	F1-score
All	0.95
Flow + Statistics	0.95
Log + Statistics	0.93
Statistics	0.93
Log + Flow	0.60
Flow	0.60
Log	0.08

Mining

Combination	F1-score
All	0.89
Flow + Statistics	0.89
Log + Statistics	0.89
Statistics	0.89
Log + Flow	0.32
Flow	0.32
Log	0.17

Ransom

Combination	F1-score
All	0.98
Flow + Statistics	0.98
Log + Statistics	0.98
Statistics	0.98
Log + Flow	0.27
Flow	0.27
Log	0.11

DoS

Combination	F1-score
All	0.92
Flow + Statistics	0.90
Log + Statistics	0.90
Statistics	0.88
Log + Flow	0.37
Flow	0.33
Log	0.20

Fig. 8. Attack-based comparison.

Table 9
Model configuration.

Algorithm	Maximum depth	Learning rate	Objective	scale_pos_weight	Epochs
XGBoost	2	1	Logistic	10	100

log model are the number of log templates generated by `apache-access`, `in.telnetd` and `postfix` in each time slot.

Fig. 10 (left) shows the order of important features in the packet flow model. We observed that only two features, ScrByte and TotByte, outperformed the others. This particular result means that many data points are related to network DoS attacks, and the flow model can inspect the two features to make predictions.

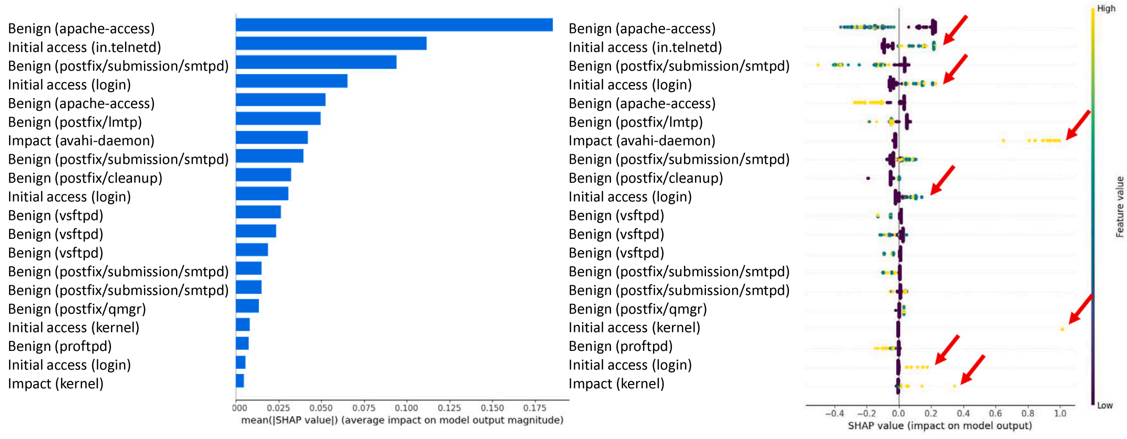


Fig. 9. Important features and SHAP values in the log model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

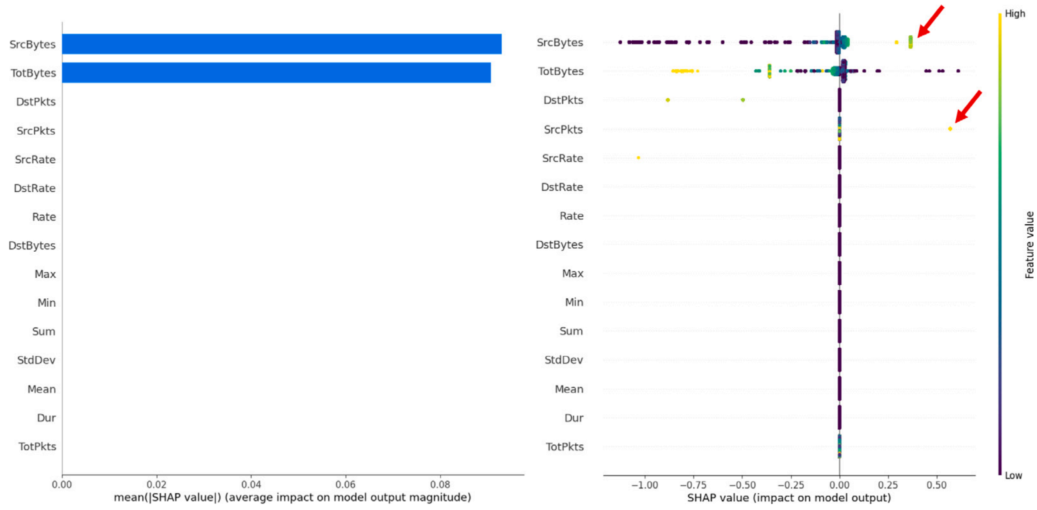


Fig. 10. Important features and SHAP values in the flow model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 10
Model threshold.

Model	Threshold	Best F1-score
System log	0.59	0.99
Packet flow	0.88	0.99
Host statistics	0.49	0.93

Table 11
Data points of the dataset with aligned extraction.

	Initial access	C&C	Impact	All
Mirai	85	60	15	160
Ransomware	4	11	17	32
Resource hijacking	2	9	12	23
Disk wipe	1	12	18	31
End point DoS	3	9	13	25
All	95	101	75	271

Fig. 11 (left) shows that important features in the statistics models are related to memory usages such as virtual memory (VSTEXT) or resident memory usage (RSIZE) on a system. It is worth noting that a special feature called MINFLT represents the number of page faults caused by a process, and the statistics model takes this information as a reference to make predictions.

6.4. SHAP value of features in each model

Once we know the important features of each model, we further inspect the features and see whether they are positive or negative to detection. We dump the SHAP value of each data point to know how the features impact the predictions. Fig. 9 (right) shows the summary plot of the overall log model. Each row represents one feature, and it has the same order as the feature importance on its left. The yellow points in each row are the data points with high importance values in that feature, and those imply more confidence that an attack occurs. If the SHAP value of a data point is positive, it means that the data are likely to be malicious. We can conclude that, if the yellow points in a feature are more likely to be positive, this feature may tend to be malicious. In this case, most benign features are the ones with their yellow points tending to be negative. The malicious features such as the logs in initial access or impact indicate that their yellow points are on the right-hand side.

Fig. 10 (right) is the summary plot in the overall flow model. We see that only two important features are in this model. SrcBytes is positive because its yellow points are likely to be positive. It means that if a data point has a large value of SrcBytes, it is likely to be predicted as positive. This is reasonable because many network DoS flows are in our dataset, and such an attack tends to generate a large number of bytes from the source to the destination during an attack. Contrary to

Table 12
Proportion of malicious data points caught per stage in attacks.

	Initial access			C&C			Impact		
	Logs	Flows	Statistics	Logs	Flows	Statistics	Logs	Flows	Statistics
Mirai	$\frac{85}{85}$	$\frac{72}{85}$	$\frac{68}{85}$	$\frac{4}{60}$	$\frac{42}{60}$	$\frac{57}{60}$	$\frac{12}{15}$	$\frac{15}{15}$	$\frac{13}{15}$
Ransomware	$\frac{2}{4}$	$\frac{3}{4}$	$\frac{4}{4}$	$\frac{1}{11}$	$\frac{5}{11}$	$\frac{10}{11}$	$\frac{0}{17}$	$\frac{2}{17}$	$\frac{17}{17}$
Resource hijacking	$\frac{1}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{1}{9}$	$\frac{6}{9}$	$\frac{8}{9}$	$\frac{1}{12}$	$\frac{2}{12}$	$\frac{11}{12}$
Disk wipe	$\frac{0}{1}$	$\frac{1}{1}$	$\frac{1}{1}$	$\frac{2}{12}$	$\frac{10}{12}$	$\frac{11}{12}$	$\frac{0}{18}$	$\frac{5}{18}$	$\frac{17}{18}$
End point DoS	$\frac{2}{3}$	$\frac{3}{3}$	$\frac{3}{3}$	$\frac{1}{9}$	$\frac{5}{9}$	$\frac{7}{9}$	$\frac{1}{13}$	$\frac{2}{13}$	$\frac{12}{13}$
All	$\frac{90}{95}$	$\frac{81}{95}$	$\frac{78}{95}$	$\frac{9}{101}$	$\frac{68}{101}$	$\frac{93}{101}$	$\frac{14}{75}$	$\frac{26}{75}$	$\frac{70}{75}$

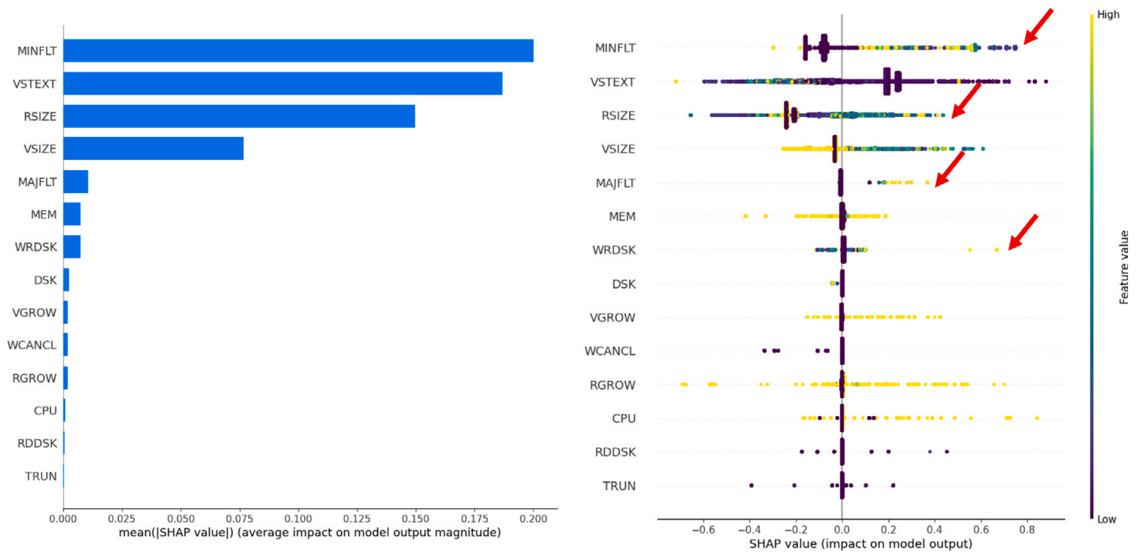


Fig. 11. Important features and SHAP values in the statistics model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the SrcBytes value above, because we run many benign services on the testbed, if a data point with a large value of TotBytes and a small value of SrcBytes, then it then tends to be normal data. As a result, we mark the TotBytes feature as normal.

For the SHAP value in the statistics model, as we see in Fig. 11 (right), the yellow points spread widely and this is hard to interpret. We do, though, make the following observations. For example, MINFLT and MAJFLT are both features related to malicious behavior. If a process causes too many page faults, it is likely an abnormal process. For example, in this work, the attack scenarios such as mining, data encryption and disk wipe, use tools to consume system resources and impact on the victim machine. If the requested file or data is not in the memory, the system will cause page faults and move data from the disk to the memory, and the number of page faults will increase. The second important feature in the statistics model is VSTEXT, which means that the virtual memory is used by the shared text of a process. We observed that the benign services such as `proftpd` and `apache2` running in the testbed had larger shared text than the malicious binaries, and they had a larger value in this feature. As a result, VSTEXT is more likely to be a normal feature. Moreover, RSIZE means the resident memory usage of a program is regarded as a malicious feature. Because we found that when the malicious binaries such as mining or data encryption were executed, they would remain in the memory for a long time to process the data on the system and then increase the usage of the resident memory.

To summarize the relationships between attacks and features, we mapped each attack stage onto its related malicious features. As Fig. 12 shows, we describe which malicious features affect the predictions most in each stage per attack. For the system logs, we found that the malicious features were the log templates generated in each attack stage, e.g., `connect from <*> (<*>)` occurred frequently in the initial access stage of Mirai and `<*> docker0: port 2(veth61255d9) entered <*> state` appeared in the C&C stage of the ransomware attack. For packet flows, the main malicious features are SrcBytes and SrcRate. If the attack stage invoked large file transfers, such as the initial access stage of resource hijacking or the impact stage of Mirai, the SrcBytes values tended to be large. Because the backdoor needed to communicate with the server frequently, the SrcRate values in these stages could increase. As for the host statistics, we observed that MINFLT and RSIZE were the malicious features in almost all the stages because these attack techniques issued page faults or included data processing, and increased the memory usage. For the impact stage, there are more malicious features related to the predictions. For example, the values of the DSK feature may increase in the impact stage of the ransomware because of the encryption of files, and the values of the WRDSK feature may be high when the disk wipe is executed. As a result, the malicious features will be different when applying different attack techniques.

7. Conclusions and future work

In this work, we designed a Python-based framework and used ML to test for three data sources, i.e., system logs, packet flows, and host statistics, to know which combination of the data sources perform best when detecting anomalies in both stage-based and attack-based aspects. We wanted not only to determine comparable results but also to interpret which features influence the prediction most and whether the important features affect the malicious or normal prediction.

To answer these questions, we first designed a Unix-based testbed, applied five attacks, and collected three kinds of data simultaneously.

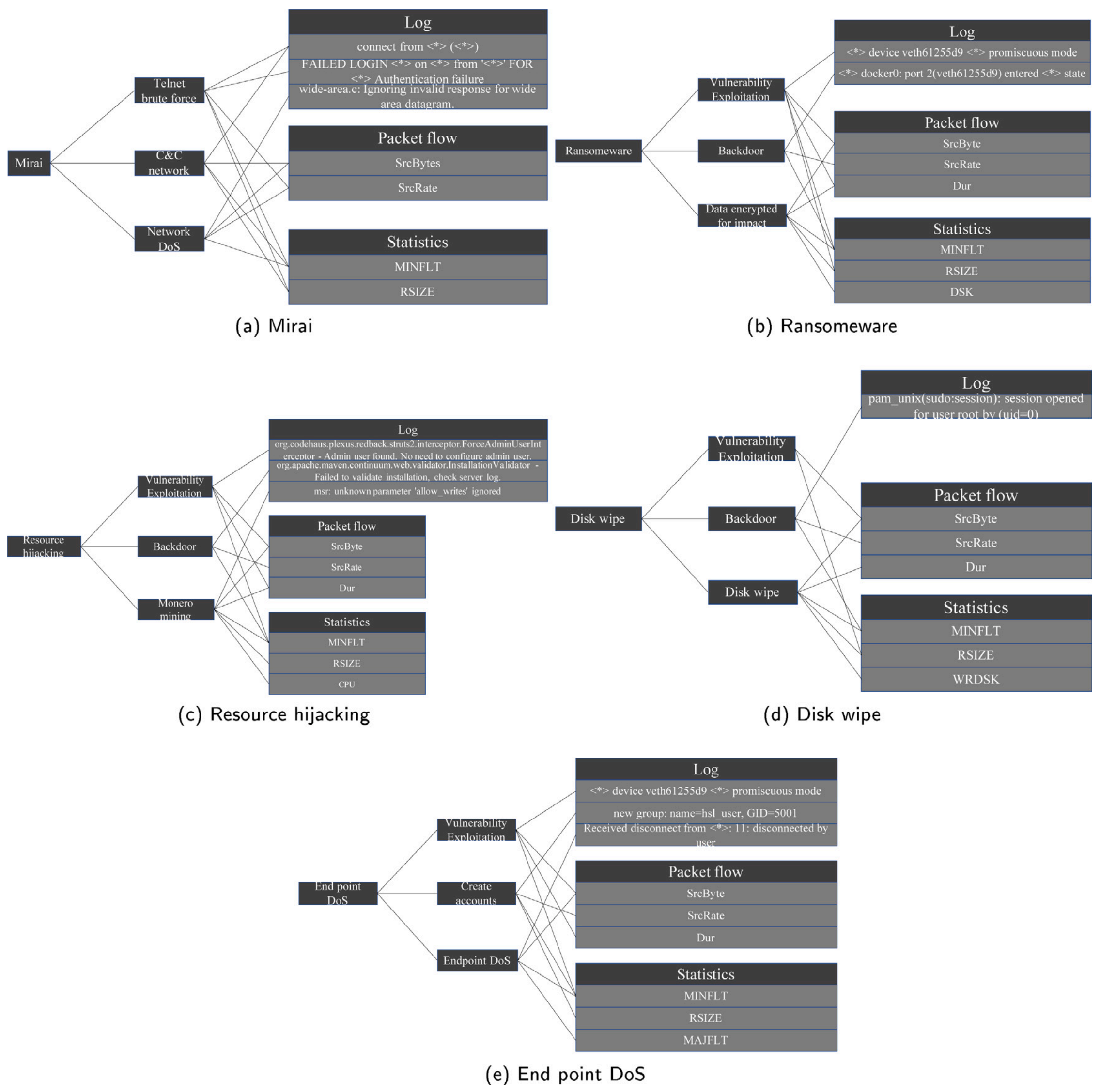

Mirai
Telnet brute force
C&C network
Network DoS
Log
connect from <*> (<*>)
FAILED LOGIN <*> on <*> from '<*>' FOR <*> Authentication failure
wide-area.c: Ignoring invalid response for wide area datagram.
Packet flow
SrcBytes
SrcRate
Statistics
MINFLT
RSIZE
(a) Mirai
Ransomeware
Vulnerability Exploitation
Backdoor
Data encrypted for impact
Log
<*> device veth61255d9 <*> promiscuous mode
<*> docker0: port 2(veth61255d9) entered <*> state
Packet flow
SrcByte
SrcRate
Dur
Statistics
MINFLT
RSIZE
DSK
(b) Ransomeware
Resource hijacking
Vulnerability Exploitation
Backdoor
Monero mining
Log
org.codehaus.plexus.redback.struts2.interceptor.ForceAdminUserInterceptor - Admin user found. No need to configure admin user.
org.apache.maven.continuum.web.validator.InstallationValidator - Failed to validate installation, check server log.
msr: unknown parameter 'allow_writes' ignored
Packet flow
SrcByte
SrcRate
Dur
Statistics
MINFLT
RSIZE
CPU
(c) Resource hijacking
Disk wipe
Vulnerability Exploitation
Backdoor
Disk wipe
Log
pam_unix(sudo:session): session opened for user root by (uid=0)
Packet flow
SrcByte
SrcRate
Dur
Statistics
MINFLT
RSIZE
WRDSK
(d) Disk wipe
End point DoS
Vulnerability Exploitation
Create accounts
Endpoint DoS
Log
<*> device veth61255d9 <*> promiscuous mode
new group: name=hsl_user, GID=5001
Received disconnect from <*>: 11: disconnected by user
Packet flow
SrcByte
SrcRate
Dur
Statistics
MINFLT
RSIZE
MAJFLT
(e) End point DoS

Fig. 12. Mapping from the attack scenarios to the malicious features.

After collecting all the data, we preprocessed and split them into training and testing datasets. We used the training dataset as well as the XGBoost algorithm to train the three models corresponding to the three data sources, and the F1-scores of each model can reach around 0.99. We then compiled the testing dataset with a fixed time slot by aligned extraction, and used the ensemble method to evaluate seven combinations of the three models to obtain the results. We conclude that combining multiple data sources can effectively improve the accuracy of ML-based detection in different attack stages with an affordable overhead. Taking more data sources as a reference, the F1-score can increase to 0.97 for the combined data sources, and the overhead of detection with each data source was affordable (at most 2.1% CPU usage). This ensemble approach is particularly important because different attacks in the different stages may manifest different volumes of clues in different data sources. The F1-score for each combination can be affected by the number of its corresponding data points. We also found that the F1-score of the host statistics (0.91) is better than the flows (0.63) and the logs (0.44).

We also highlight the important features from the individual data sources. From the evaluation results, we found that the numbers of log templates could be important features. The logs generated by malicious behaviors tended to have larger SHAP values, implying more malicious features, while the logs generated from normal behaviors may have smaller SHAP values. In the model from packet flows, if the data point had a large SrcBytes value, it was likely to indicate malicious behavior. By contrast, if a flow had a large TotBytes value and a rather small SrcBytes value, it tended to be normal. Finally, the host statistics model favored using features such as resident memory usage (RSIZE) or the number of page faults (MINFLT) to make predictions. If a data point caused too many page faults or used too much resident memory, it tended to be malicious.

Three aspects in this work can be further improved in the future. First, we can add more attack scenarios and system statistics into our framework to cover more scenarios and enhance the ability of the models. Second, it would certainly be useful to evaluate and validate the performance of the system in a real-world situation, instead of using synthetic datasets. Third, the impact of adversarial attacks on the ML-based detection models [35] can be also evaluated in the future.

CRediT authorship contribution statement

Ying-Dar Lin: Problem definition, Paper revision. **Ze-Yu Wang:** Algorithm design, Software, Experimental study, Writing. **Po-Ching Lin:** Writing – review & editing, Supervision, Validation. **Van-Linh Nguyen:** Paper revision. **Ren-Hung Hwang:** Conceptualization, Supervision. **Yuan-Cheng Lai:** Algorithm design, Paper revision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Sengupta J, Ruj S, Bit SD. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and iIoT. J Netw Comput Appl 2020;149:102481–500.

[2] Zhang Z-K, Cho MCY, Wang C-W, Hsu C-W, Chen C-K, Shieh S. IoT security: ongoing challenges and research opportunities. In: IEEE 7th international conference on service-oriented computing and applications. 2014, p. 230–4.

[3] Tankard C. The security issues of the internet of things. Comput Fraud Secur 2015;2015(9):11–4.

[4] Khraisat A, Gondal I, Vamplew P, Kamruzzaman J. Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2019;2(1):20–41.

[5] Hwang R-H, Peng M-C, Huang C-W, Lin P-C, Nguyen V-L. An unsupervised deep learning model for early network traffic anomaly detection. IEEE Access 2020;8:30387–99.

[6] Du M, Li F, Zheng G, Srikumar V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the ACM SIGSAC conference on computer and communications security; 2017, p. 1285–98.

[7] Ham H-S, Kim H-H, Kim M-S, Choi M-J. Linear SVM-based android malware detection for reliable IoT services. J Appl Math 2014;2014.

[8] Strom BE, Applebaum A, Miller DP, Nickels KC, Pennington AG, Thomas CB. Mitre att&ck: Design and philosophy. Technical Report, 2018.

[9] Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, Durumeric Z, Halderman JA, Invernizzi L, Kallitsis M, et al. Understanding the mirai botnet. In: 26th USENIX security symposium (USENIX security). 2017, p. 1093–110.

[10] Zavrak S, İskefiyeli M. Anomaly-based intrusion detection from network flow features using variational autoencoder. IEEE Access 2020;8:108346–58.

[11] Liu M, Xue Z, Xu X, Zhong C, Chen J. Host-based intrusion detection system with system calls: Review and future trends. ACM Comput Surv 2018.

[12] Chen T, Guestrin C. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM sigkdd international conference on knowledge discovery and data mining; 2016, p. 785–94.

[13] Kabari LG, Onwuka UC. Comparison of bagging and voting ensemble machine learning algorithm as a classifier. Int J Adv Res Comput Sci Softw Eng 2019;19–23.

[14] Lundberg SM, Lee S-I. A unified approach to interpreting model predictions. In: Advances in neural information processing systems, Vol. 30. Curran Associates, Inc.; 2017, p. 4765–74.

[15] Alrashdi I, Alqazzaz A, Aloufi E, Alharthi R, Zohdy M, Ming H. Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning. In: IEEE 9th annual computing and communication workshop and conference (CCWC). 2019, p. 0305–10.

[16] Meidan Y, Bohadana M, Mathov Y, Mirsky Y, Shabtai A, Breitenbacher D, Elovici Y. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervasive Comput 2018;17(3):12–22.

[17] Min E, Long J, Liu Q, Cui J, Chen W. TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest. Secur Commun Netw 2018;2018:1–9.

[18] Al-Qatf M, Lasheng Y, Al-Habib M, Al-Sabahi K. Deep learning approach combining sparse autoencoder with SVM for network intrusion detection. IEEE Access 2018;6:52843–56.

[19] Wang W, Sheng Y, Wang J, Zeng X, Ye X, Huang Y, Zhu M. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. IEEE Access 2017;6:1792–806.

[20] Min E, Long J, Liu Q, Cui J, Cai Z, Ma J. Su-ids: A semi-supervised and unsupervised framework for network intrusion detection. In: International conference on cloud computing and security. Springer; 2018, p. 322–34.

[21] Xu W, Huang L, Fox A, Patterson D, Jordan MI. Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles; 2009, p. 117–32.

[22] Sheluhin O, Osin A. Anomaly states monitoring of large-scale systems with intellectual analysis of system logs. In: 24th conference of open innovations association (FRUCT). IEEE; 2019, p. 395–401.

[23] Otomo K, Kobayashi S, Fukuda K, Esaki H. Latent variable based anomaly detection in network system logs. IEICE Trans Inf Syst 2019;102(9):1644–52.

[24] Zhang X, Xu Y, Lin Q, Qiao B, Zhang H, Dang Y, Xie C, Yang X, Cheng Q, Li Z, et al. Robust log-based anomaly detection on unstable log data. In: Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering; 2019, p. 807–17.

[25] Meng W, Liu Y, Zhu Y, Zhang S, Pei D, Liu Y, Chen Y, Zhang R, Tao S, Sun P, et al. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: IJCAI. 2019, p. 4739–45.

[26] Wang J, Tang Y, He S, Zhao C, Sharma PK, Alfarraj O, Tolba A. LogEvent2vec: LogEvent-to-vector based anomaly detection for large-scale logs in internet of things. Sensors 2020;20(9):2451–69.

[27] Borthakur D, et al. HDFS architecture guide. Hadoop Apache Project; 2008.

[28] Özgür A, Erdem H. A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. PeerJ Preprints 2016.

[29] Sarhan M, Layeghy S, Portmann M. Towards a standard feature set for network intrusion detection system datasets. Mob Netw Appl 2021;27:357—370.

[30] Lopez-Martin M, Carro B, Arribas JI, Sanchez-Esguevillas A. Network intrusion detection with a novel hierarchy of distances between embeddings of hash IP addresses. Knowl-Based Syst 2021.

[31] Chou D, Jiang M. A survey on data-driven network intrusion detection. ACM Comput Surv 2022;54:1–36.

[32] Lopez-Martin M, Sanchez-Esguevillas A, Arribas JI, Carro B. Supervised contrastive learning over prototype-label embeddings for network intrusion detection. Inf Fusion 2022;79:200–28.

[33] He P, Zhu J, Zheng Z, Lyu MR. Drain: An online log parsing approach with fixed depth tree. In: IEEE international conference on web services (ICWS). IEEE; 2017, p. 33–40.

[34] Kennedy D, O'gorman J, Kearns D, Aharoni M. Metasploit: The penetration tester's guide. No Starch Press; 2011.

[35] Apruzzese G, Andreolini M, Ferretti L, Marchetti M, Colajanni M. Modeling realistic adversarial attacks against network intrusion detection systems. ACM Digit Threats: Res Pract 2021.

Ying-Dar Lin is a Chair Professor of computer science at National Yang-Ming Chiao-Tung University (NYCU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose during 2007–2008, CEO at Telecom Technology Center, Taiwan, during 2010–2011, and Vice President of National Applied Research Labs (NARLabs), Taiwan, during 2017–2018. He was the founder and director of Network Benchmarking Lab (NBL) in 2002–2018, which reviewed network products with real traffic and automated tools, and has been an approved test lab of the Open Networking Foundation (ONF). He also cofounded L7 Networks Inc. in 2002, later acquired by D-Link Corp, and O'Prueba Inc. a spin-off from NBL, in 2018. His research interests include network security, wireless communications, network softwarization, and machine learning for communications. His work on multi-hop cellular was the first along this line, and has been cited over 1000 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Fellow (class of 2013), IEEE Distinguished Lecturer (2014–2017), ONF Research Associate (2014–2018), and received in 2017 Research Excellence Award and K. T. Li Breakthrough Award. He has served or is serving on the editorial boards of several IEEE journals and magazines, including Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST, 1/2017-12/2020). He published a textbook, Computer Networks: An Open Source Approach, with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Ze-Yu Wang received Master degree of Computer Science in Network Engineering, from National Chiao Tung University, Taiwan in 2021. His research interests include machine learning, network security, and intrusion detection.

Po-Ching Lin received his Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2008. He joined the faculty of the Department of Computer Science and Information Engineering, CCU, in August 2009. He is currently a professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems.

Van-Linh Nguyen received his Ph.D. degree in computer science and information engineering from National Chung Cheng University (CCU), Taiwan, in 2019. He joined the Department of Information Technology, Thai Nguyen University of Information and Communication Technology (TNU-ICTU) in 2012, where he is now an assistant professor. His research interests include cybersecurity, vehicular networks, autonomous driving, and edge computing.

Ren-Hung Hwang received his Ph.D. degree in computer science from the University of Massachusetts, Amherst. He joined the Department of Computer Science and Information Engineering, National Chung Cheng in 1993, where he is now a distinguished professor and the chief information technology officer. He has published more than 250 international journal and conference papers. He served as the Dean of the College of Engineering during 2014 2017. He received the IEEE Best Paper Award from IEEE Ubi-Media 2018, IEEE SC2 2017, and IEEE IUCC 2014. His current research interests include Internet of Things, network security, cloud/edge/fog computing, and 5G V2X networks.

Yaun-Cheng Lai received his Ph.D. degree in the Department of Computer and Information Science from National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology in August 2001 and has been a distinguished professor since June 2012. His research interests include performance analysis, software-defined networking, wireless networks, and IoT security.