Computers & Security 142 (2024) 103859

Contents lists available at ScienceDirect

Computers & Security

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

Two-stage multi-datasource machine learning for attack technique and lifecycle detection

Ying-Dar Lin [a], Shin-Yi Yang [a], Didik Sudyana [a,*], Fietyata Yudha [a], Yuan-Cheng Lai [b], Ren-Hung Hwang [c]

[a] Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, 300, Taiwan
[b] Department of Information Management, National Taiwan University of Science and Technology, Taipei, 10607, Taiwan
[c] College of Artificial Intelligence, National Yang Ming Chiao Tung University, Tainan, 711, Taiwan

ARTICLE INFO

Keywords:
Ml-based IDS
Attack lifecycle detection
Multi-datasource IDS
Two-stage lifecycle detection

ABSTRACT

Intrusion detection systems (IDS) have increasingly adopted machine learning (ML) techniques to enhance their ability to detect a wide range of attack variants. However, the traditional focus in current research primarily revolves around identifying specific attack types or techniques using a single data source. However, this approach lacks a holistic perspective on attacks, which can result in missed detections. To improve the effectiveness of responding to detected attacks, it is essential to identify them based on their lifecycles and incorporate information from multiple data sources. In this study, we present three distinct approaches for detecting attack lifecycles, each leveraging different ML methodologies: a single-stage ML model, a two-stage ML+ML approach, and ML with sequence matching (ML+SM). Simultaneously, we explore the benefits of utilizing multiple data sources, including network traffic, system logs, and host statistics, to enhance technique detection capabilities. Our evaluation of these methods reveals that on lifecycle detection, the two-stage ML+ML approach outperforms the others, achieving an impressive F1 score of 0.994. In contrast, the single-stage and ML+SM methods yield F1 scores of 0.887 and 0.189, respectively. Furthermore, the integration of multiple data sources proves highly advantageous, with the combination of all three sources yielding the highest F1 score of 0.922 on technique detection.

1. Introduction

As cyber threats evolve, robust cybersecurity measures are essential (Hoque et al., 2014). Intrusion Detection Systems (IDS) are a key protective element. Traditionally, IDS relied on signature- and anomaly-based methods, with limitations in detecting unknown attacks and high false positives. Integrating machine learning into IDS is crucial for effectively addressing a broader spectrum of attack variations and behaviors (Hamid et al., 2016).

While machine learning-based IDS (ML-IDS) has demonstrated effectiveness in identifying attack variants, several challenges persist. Specifically, the effectiveness of an ML model often depends on the quality and diversity of its training dataset. Currently, there is a limited number of publicly available datasets for intrusion detection, with most well-known datasets primarily utilizing network traffic data as their main sources (Yang et al., 2022). For instance, CIC-IDS-2017 (Sharafaldin et al., 2018) is focused on network traffic data. On the other hand, datasets like DARPA Transparent Computing (TC) (Operationally Transparent Cyber Dataset, 2019) solely provide system log data. This limitation has directed many research efforts toward concentrating exclusively on detecting attacks from these specific data sources (Zavrak and İskefiyeli, 2020; Meng et al., 2019; Ham et al., 2014; Toupas et al., 2019; Meng et al., 2018; Sun et al., 2020; Bagui et al., 2022). In this context, a data source refers to a certain type of information used to detect malicious activities in a system. Solely relying on one data source, however, might be inadequate, especially when handling complex data that encompasses both external and internal behaviors. Recognizing this challenge, a study has explored the integration of various data sources, such as network traffic, system logs, and host statistics, to enhance detection comprehensiveness (Lin et al., 2022).

Another significant challenge arises when the focus is solely on detecting the presence of an attack (Zavrak and İskefiyeli, 2020; Meng et al., 2019; Ham et al., 2014; Sun et al., 2020; Meng et al., 2018; Toupas et al., 2019) or identifying specific attack techniques (Lin et al.,

* Corresponding author.
E-mail addresses: ydlin@cs.nycu.edu.tw (Y.-D. Lin), shinyee.cs10@nycu.edu.tw (S.-Y. Yang), dsudyana@cs.nycu.edu.tw (D. Sudyana), fyudha@cs.nycu.edu.tw (F. Yudha), laiyc@cs.ntust.edu.tw (Y.-C. Lai), rhhwang@nycu.edu.tw (R.-H. Hwang).

https://doi.org/10.1016/j.cose.2024.103859
Received 12 February 2024; Received in revised form 29 March 2024; Accepted 17 April 2024
Available online 26 April 2024

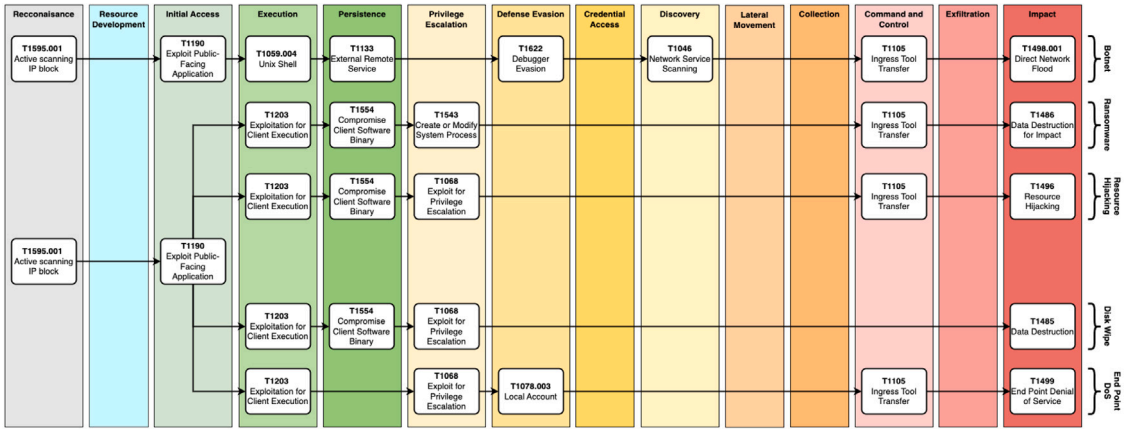

Fig. 1. Mapping attacks to sequences of techniques on CREMEv2 (Yudha, 2023).

2022; Bagui et al., 2022). While these methods are valuable, they may not provide a complete understanding of the attack landscape. ML-IDS systems that concentrate solely on these aspects might miss crucial details such as the attack's broader objectives, the assets it targets, and its duration. In many cases, sophisticated attacks, such as Advanced Persistent Threats (APTs), involve various infiltration techniques, long-term, and discreet persistence within a system environment (Alshamrani et al., 2019; Yue et al., 2024). By solely focusing on attack detection or technique identification, the broader context, including the attack's goals, target assets, and duration, may remain elusive.

Therefore, there is a need for further research to develop more comprehensive and robust ML-IDS that can detect attacks from multiple data sources throughout the entire attack process. A typical attack is composed of a sequence of different attack techniques, called attack lifecycle (Kaloudi and Li, 2020). Detecting lifecycles from the sequences of techniques provides valuable information for security operations centers (SOCs). This information can be leveraged to determine effective countermeasure procedures aimed at preventing the completion of these lifecycles, thus minimizing potential damage.

Furthermore, the insights gained from lifecycle detection can also be instrumental in the development of new IDS models. By understanding these lifecycles, IDS can be upgraded to detect them at the earliest stages, potentially identifying threats before the entire attack process unfolds. This proactive approach is a significant advancement in cybersecurity and contributes to more robust and timely threat mitigation.

The concepts of attack lifecycle and technique are defined by MITRE ATT&CK (Strom et al., 2018) framework, which is useful for developing effective defense strategies. This framework has defined attack tactics, which refer to the specific objectives of each stage of an attack, and the attack techniques that are used to accomplish these attack tactics.

To solve the problems mentioned above, in this study, we detect attack techniques and attack lifecycles by using multiple data sources, including network traffic, system logs, and host statistics, simultaneously. Network traffic contains some information in the network flow, such as the number of packets, ports, protocol, and duration. System logs are generated by the operating system, recording the activities related to the operating system and the software running on a computer or network. Host statistics include the status of each process, such as CPU usage, memory usage, and page fault rate. Combining these sources yields a comprehensive perspective on network and host activities, facilitating holistic threat assessment.

Moreover, diverse attack techniques manifest in these data sources. For example, network-based attacks often leave traces in traffic data, while malware infections show up in system logs (Beaman et al., 2021), and cryptocurrency-mining is better detected through network and host statistics (Gomes et al., 2020). Integrating these multiple sources enables effective detection of a wide range of attack techniques. In contrast to existing works, this research delves deeper into the synergistic effects of these data sources. We explore how their integrated analysis can significantly improve the detection of the attack lifecycles through analyzing the sequences of techniques.

In this work, we propose three lifecycle detection approaches: ML, ML+ML, and ML+SM. The single-stage ML approach utilizes multiple data sources to predict lifecycles through a single ML model. The ML+ML approach employs ML models to detect attack techniques from various data sources and classify associated lifecycles from technique sequences. The ML+SM approach combines ML models for technique detection and the sequence matching method (Levenshtein et al., 1966) for lifecycle classification. These approaches enhance our understanding of cyber attack characteristics.

This work makes three key contributions to the field: (1) it addresses the challenge of detecting sophisticated cyber attacks by identifying attack lifecycles from technique sequences, a novel approach in the domain, (2) we introduce a novel two-stage machine learning method utilizing multi-datasource for detecting attack techniques and classifying attack lifecycles, (3) we assess the effectiveness of the three proposed approaches (ML, ML+ML, ML+SM) in identifying attack lifecycles.

Furthermore, to enhance our knowledge, we also investigate the following issues: (1) single- vs. two-stage ML, to compare the performance of the three approaches in detecting lifecycles, (2) the performance evaluation of learning models on lifecycle detection, (3) we comprehensively assess the effectiveness of three distinct methodologies: ML, ML+ML, and ML+SM, in identifying attack lifecycles, offering groundbreaking insights into the detection and classification of cyber threats.

The rest of this paper is organized as follows: in Section 2, we provide the background and related works on ML-based intrusion detection. Section 3 and 4 present the problem statement and solution approaches, respectively. In Section 5, we describe the detailed implementation. The experiment results are shown in Section 6. Finally, Section 7 concludes the works and discusses future work.

2. Background and related works

In this section, we provide an overview of the background, including the MITRE ATT&CK and the attack scenarios used in this study. At the end of the section, we present a comparison with other works that differ from our approach

Table 1
ML-based intrusion detection approaches.

Detection target	Paper	Data source			Dataset	Approach
		Network traffic	System logs	Host statistics		
Attack level	Zavrak and İskefiyeli (2020)	Binary	–	–	CICIDS2017, NSLKDD	AE, VAE
	Toupas et al. (2019)	Multiclass	–	–	CICIDS2017	SMOTE, ENN, DNN
	Meng et al. (2019)	–	Binary	–	HDFS, BGL	LSTM
	Meng et al. (2018)	–	Multiclass	–	self-collected	PU learning, SVM
	Sun et al. (2020)	–	Multiclass	–	Thunderbird, BGL	SMOTE, Bi-LSTM
	Du et al. (2017)	–	Binary	–	HDFS, OpenStack	LSTM
	Ham et al. (2014)	–	–	Binary	self-collected	SVM
	Liu et al. (2022)	Multicass	Multicass	–	SCVIC-CIDS-2021	XGBoost
	Lin et al. (2022)	Binary	Binary	Binary	CREME	XGBoost
	Hwang et al. (2023)	Binary	Binary	Binary	CREME	DL
Technique level	Bagui et al. (2022)	Binary	–	–	UWF-ZeekData22	Multiple algorithms
Technique and lifecycle levels	Ours	Multiclass	Multiclass	Multiclass	CREMEv2 (Yudha, 2023)	XGBoost, ML

2.1. MITRE ATT&CK: Tactics, techniques and lifecycles

The MITRE institution has developed a framework called ATT&CK (Strom et al., 2018), which is used for understanding the techniques, tactics, and procedures (TTPs) employed by attackers during cyber-attacks. ATT&CK matrix presents attack stages and corresponding techniques used in each stage.

The lifecycle concept provides a comprehensive understanding of the technique sequences used in an attack. For example, advanced persistent threat (APT) attacks may have a long lifecycle that includes multiple tactics and techniques (Singh et al., 2019). By understanding the lifecycle of an attack, defenders can develop effective countermeasures to detect and prevent attacks (Wang et al., 2024).

2.2. Attack scenarios

In this study, we used the CREMEv2 dataset (Yudha, 2023), featuring five common real-world attacks: Mirai botnet, ransomware, resource hijacking, disk wipe, and endpoint DoS. Fig. 1 maps out the tactics and techniques for each attack in the dataset. Rows represent the reproduced lifecycles, columns indicate the tactics, and blocks display technique details, summing up to 17 attack techniques. Each attack's background and execution methods are further detailed.

Botnet attack can be used for further attacks, such as DDoS. An example is the Mirai botnet. In the context of the CREMEv2 dataset, the Mirai botnet exhibits diverse lifecycles, a reflection of its multifaceted nature. The dataset includes a more general representation of Mirai's lifecycle, starting from the creation of the Mirai botnet. The reason for this diversity in lifecycles is attributed to the complexity of Mirai's attack strategies (Affinito et al., 2023). It utilizes various techniques based on the MITRE ATT&CK framework, which encompasses a wide array of tactics and procedures for compromising and controlling target devices.

Resource hijacking involves seven techniques where attackers exploit a victim's system for personal gains. Falling under the 'Impact' tactic, a notable subtype is cryptojacking. Victims may experience heightened energy use, system slowdowns, and significant costs due to these attacks.

Disk wipe attack involves six techniques to try to delete vital files or directories on targets. In some cases, it is coupled with ransomware attacks, aiming to encrypt important files and demand payment in exchange for the decryption key.

Endpoint DoS attack, encompassing seven techniques and targeting specific devices, aims to exhaust their resources. Unlike broader network attacks, this focuses on the device itself. Overwhelming traffic or requests can make the device crash, leading to potential data loss or service disruptions.

2.3. Related works

The related works can be categorized into three groups based on their detection targets, as summarized in Table 1. The first group, 'attack level', aims to detect either the presence of an attack or its type. The second group, 'technique level', seeks to identify the techniques employed in an attack, and lastly, 'technique and lifecycle levels', seek to identify the lifecycle based on the sequence of techniques.

Currently, most of the research focuses on attack-level detection. Researchers such as Zavrak and İskefiyeli (2020) and Toupas et al. (2019) concentrated on identifying attacks in public network traffic datasets, harnessing deep learning (DL) models. In contrast, Meng et al. (2019, 2018), Sun et al. (2020), and Du et al. (2017) handled attacks in system log datasets. They processed text data and employed LSTM-based DL models. Meng et al. (2018) embraced a semi-supervised learning approach to distinguish benign from malicious data, using SVM models for classification. Ham et al. (2014) applied SVM models to classify statistical data gathered from mobile devices. Toupas et al. (2019) and Sun et al. (2020) acknowledged the challenge of data imbalance in multi-class classification tasks and employed resampling methods to address this issue. Liu et al. (2022) created a new dataset with network traffic and system logs data using the metadata from the well-known public dataset (CIC-IDS-2018) and used XGBoost to test the proposed dataset. However, this study only focuses on how to evaluate the proposed dataset. Notably, Lin et al. (2022) leveraged the XGBoost model for attack detection and explored the use of multiple data sources to enhance performance. However, these studies primarily concentrated on proposing ML models for binary and multi-class classification of attacks and paid less attention to the underlying attack techniques. Even when the dataset in Lin et al. (2022) contained both tactic and attack labels, the authors primarily focused on binary classification using the attack label for detection. Identifying the techniques employed by attackers provides valuable insights into their behaviors, enabling the development of more effective countermeasure strategies.

Moreover, a related study by Bagui et al. (2022) involved technique-level detection. They constructed a network traffic dataset and categorized it based on reconnaissance and discovery attack tactics, subsequently employing various ML models to assess dataset quality. While their work focused on attack techniques, it did not encompass the concept of attack lifecycles. By detecting the complete attack lifecycle, network administrators gain the capability to comprehensively track an attack sequence, thereby bolstering the efficacy of ML-IDS in identifying and responding to intricate attacks.

Among the mentioned studies, Lin et al. (2022) is closely related to our work. While they used a similar dataset and demonstrated the effectiveness of multiple data sources, the objectives and approaches differ. They conducted binary classification to detect attacks based on predefined tactics, whereas we performed multiclass classification to detect lifecycles based on attack technique sequences. Their work primarily focused on generating multiple data sources through attack

Table 2
Notations.

Dataset		
Dataset	D	$D = D^R \bigcup D^P$
Training Dataset	D_w^R	$D_w^R, w = 0, 1, 2$, 0: traffic flow, 1: system log, 2: host statistics
		$D_0^R = \{(x_i^N, y_i^T, y_i^L) \mid i \in [1, RN]\}$; RN : size of training traffic flow dataset
		$D_1^R = \{(x_j^{Log}, y_j^T, y_j^L) \mid j \in [1, RL]\}$; RL : size of training system log dataset
		$D_2^R = \{(x_k^S, y_k^T, y_k^L) \mid k \in [1, RS]\}$; RS : size of training host statistics dataset
Testing Dataset	D_w^P	$D_w^P, w = 0, 1, 2$
		$D_0^P = \{(x_i^N, y_i^T, y_i^L) \mid i \in [1, PN]\}$; PN : size of testing traffic flow dataset
		$D_1^P = \{(x_j^{Log}, y_j^T, y_j^L) \mid j \in [1, PL]\}$; PL : size of testing system log dataset
		$D_2^P = \{(x_k^S, y_k^T, y_k^L) \mid k \in [1, PS]\}$; PS : size of testing host statistics dataset
Technique Sequence Dataset	D^T	$D^T = D^{RT} \bigcup D^{PT}$
		$D^{RT} = \{([y_0^T y_1^T y_2^T ...], y_l^L) \mid l \in [1, RT]\}$; RT : size of training technique sequence dataset
		$D^{PT} = \{([y_0^T y_1^T y_2^T ...], y_l^L) \mid l \in [1, PT]\}$; PT : size of testing technique sequence dataset
Data Input	x	$x : x_i^N, x_j^{Log}, x_k^S$
		$x_i^N \in R^{NT}$, $x_j^{Log} \in R^{NL}$, $x_k^S \in R^{NS}$
		NT, NL, NS: size of traffic flow, system log, host statistics features
Technique Label	y^T	$y^T = \{y_i^T, y_j^T, y_k^T \mid y_i^T, y_j^T, y_k^T \in [1, \lvert T \rvert]\}$, $\lvert T \rvert$: Number of predefined techniques
Lifecycle Label	y^L	$y^L = \{y_i^L, y_j^L, y_k^L \mid y_i^L, y_j^L, y_k^L \in [1, \lvert L \rvert]\}$, $\lvert L \rvert$: Number of predefined lifecycles
Machine Learning		
Technique Classifier	M_w^T	$M_w^T, w = 0, 1, 2$
Optimal Technique Classifier	M_w^{T*}	M_w^T with highest F1 score
Lifecycle Classifier	M^L	M^L is ML model for lifecycle detection
Optimal Lifecycle Classifier	M^{L*}	M^L with highest F1 score
Ensemble Team	M^E	$M^E \mid M^E \in M$ where $M = \bigcup_w M_w^T$
Bset Ensemble Team	M^{E*}	M^E with highest F1 score
ML Model	M_n	$M_n = ML_n(D^R)$
ML Algorithm	ML_n	$0 \le n < \lvert ML \rvert$, $\lvert ML \rvert$: Number of ML algorithms
Cyber Attack		
Predefined Lifecycles	L	$L = \{L_m\}, 1 \le m \le \lvert L \rvert$, where L_m denotes the m-th lifecycle.

replays and assessing dataset quality. In contrast, our research centers on identifying attack technique sequences and leveraging them for lifecycle detection.

3. Problem statements

This section covers our discussion of the problem statement in two subsections: the notation table and the problem description. We start by explaining the notations. Next, we describe the main problem of the work.

3.1. Notations

Table 2 presents the notations used in this work, classified into three categories: dataset (network traffic, system log, host statistics), machine learning (algorithms, models, ensemble teams), and cyber attack (techniques, lifecycles). The details of each category are listed as follows:

- **Dataset:** Datasets comprise data inputs x as well as their corresponding technique labels y^T and lifecycle labels y^L. These datasets are further divided into two groups: training datasets D_w^R and testing datasets D_w^P, each containing three types of data sources. In this context, w represents the data sources: 0 denotes network traffic, 1 stands for system logs, and 2 indicates host statistics. Furthermore, the technique sequence dataset D^T is composed of sequences of y^T with corresponding label y^L.
- **Machine Learning:** To have the optimal classifiers, the best models M_w^{T*} and M^{L*} should be obtained for different tasks. Ensemble teams are also included in this category, and they are classified into two types: ensemble teams for models from each data source M^E and the best ensemble teams M^{E*}.
- **Cyber Attack:** In the final section on cyber attacks, we use a total of five attack lifecycles, $\lvert L \rvert$, and seventeen attack techniques, $\lvert T \rvert$.

3.2. Problem statements

In today's complex cybersecurity landscape, enterprises face continuous and multifaceted threats. Attacks on these enterprises are rarely straightforward; they typically initiate at the web interface, progress through internal systems, and ultimately target sensitive databases. This multi-stage attack employs various techniques across diverse data sources, encompassing network traffic, system logs, and host statistics.

To address these complex real-world challenges, we present a comprehensive threat model. Fig. 2 visually represents this model, underlining its significance as a fundamental framework for comprehending

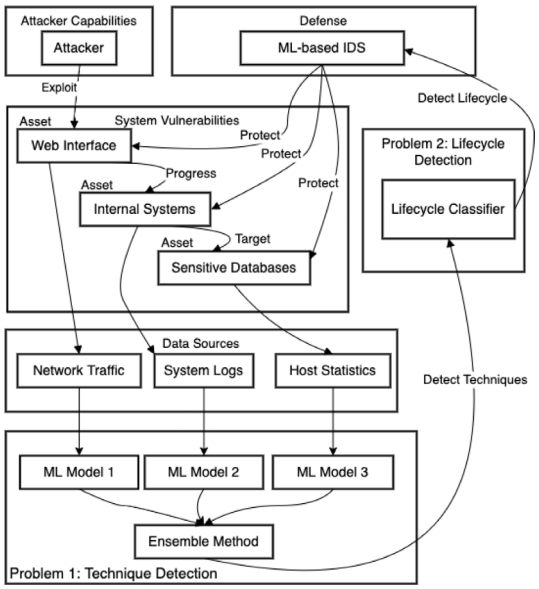

Fig. 2. The threat model illustrating the attacks and the challenges of technique and lifecycle detection in multi-datasource.

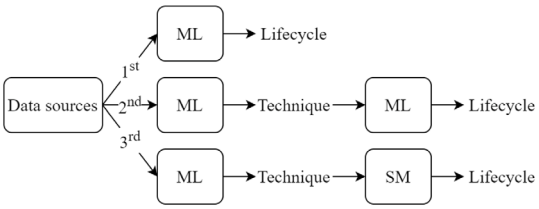

Fig. 3. Three lifecycle detection approaches.

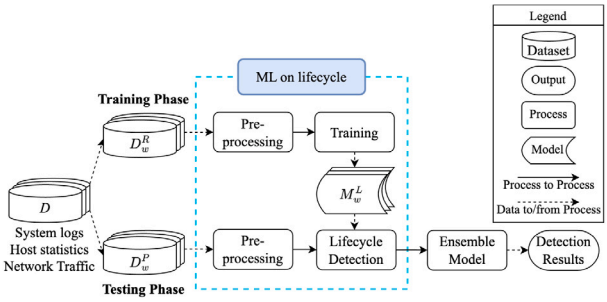

Fig. 4. ML on lifecycle.

and countering multifaceted threats. The threat model addresses two key challenges: the efficient detection of attack techniques distributed across diverse data sources, and the intricate task of assembling these techniques to gain insight into complete attack lifecycles.

Technique detection from multi-datasource

In the first problem, we detect attack techniques from multiple data sources, using an ensemble method to merge the results from each.

Given a training dataset from multi-datasource D_w^R, composed of x and its corresponding y^T, and machine learning algorithms ML_n, as well as a testing dataset D_w^P, we need to determine the best ensemble team M^{E*}. This ensemble should achieve the highest F1 score when evaluated using the testing dataset D_w^P. The statement for this problem is formally defined as follows:

Input: Training dataset D_w^R which is a set of x with y^T, machine learning algorithms ML_n, testing dataset D_w^P.

Output: Best ensemble team M^{E*}.

Objective: Maximize F1 score on ensemble team M^E tested using testing data D_w^P.

Lifecycle detection from technique sequences

In the second problem, we classify the attack lifecycle using a dataset composed of technique sequences. The objective is to find the best classifier for this task.

Given training dataset D^{RT}, which contain sequences of y^T with corresponding label y^L, machine learning algorithms ML_n and testing dataset D^{PT}, we need to determine the best lifecycle classifier M^{L*}. This classifier should achieve the highest F1 score when evaluated using the testing dataset D^{PT}. The problem statement for this problem is formally defined as follows:

Input: Training dataset D^{RT} which contain sequences of y^T with corresponding label y^L, machine learning algorithms ML_n, testing dataset D^{PT}.

Output: Best lifecycle classifier M^{L*}

Objective: Maximize F1 score on lifecycle classifier M^L tested using testing data D^{PT}

4. Solution approaches

This section begins with an overview of the solutions including three approaches to detect the attack lifecycles using multiple data sources. Then, we look at the details of the components in these three approaches: trained ML models from each data source, ensemble method, trained lifecycle detection model, and sequence matching method.

4.1. Solution overview

Fig. 3 illustrates three lifecycle detection methods. The first (single-stage ML) directly classifies data into target lifecycles, similar to identifying attack types but without analyzing technique sequences. The second (two-stage ML+ML) employs ML models twice: first to identify attack techniques, then to classify them into lifecycles after converting them into technique sequences. The third (two-stage ML+SM) detects techniques using ML, forms sequences, and applies sequence matching for detection. It is important to note that while we utilize various ML and DL models for technique and lifecycle detection, we still refer to our approach as 'ML' since DL is a subset of ML.

Fig. 4 presents the traditional approach for lifecycle detection through multiple data sources. The process begins with dividing each data source into training and testing datasets, pre-processing them, and training three ML models using the training datasets. The testing datasets are then used to detect lifecycles. In the end, the ensemble model combines the results to output the final outcome. The details of the training process and the ensemble process are described in Sections 4.2 and 4.3, respectively.

Fig. 5 outlines the ML+ML approach for lifecycle detection in two stages. Initially, techniques are detected using multiple data sources. After splitting and pre-processing the data, three ML models are trained and tested, with results consolidated via an ensemble model. The second stage classifies lifecycles using technique sequences derived from the ensemble. The sequences serve as the training dataset to train and test an ML model for lifecycle detection. Details of each stage are provided in Sections Section 4.2, 4.4, and 4.3.

Fig. 6 illustrates the ML+SM approach for lifecycle detection using sequence matching. The process starts by splitting each data source into training and testing datasets. After pre-processing, we train three ML models using these datasets. Techniques are detected using the testing datasets and merged via an ensemble model. Technique sequences are

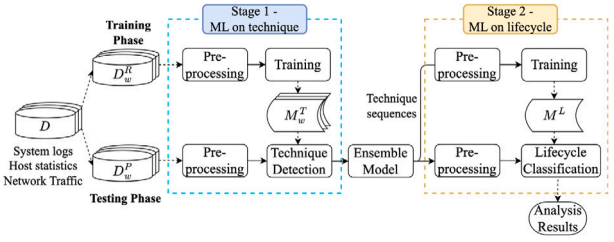

Fig. 5. ML on technique + ML on lifecycle.

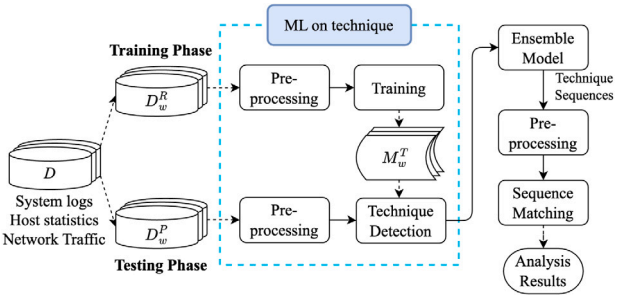

Fig. 6. ML on technique + sequence matching on lifecycle.

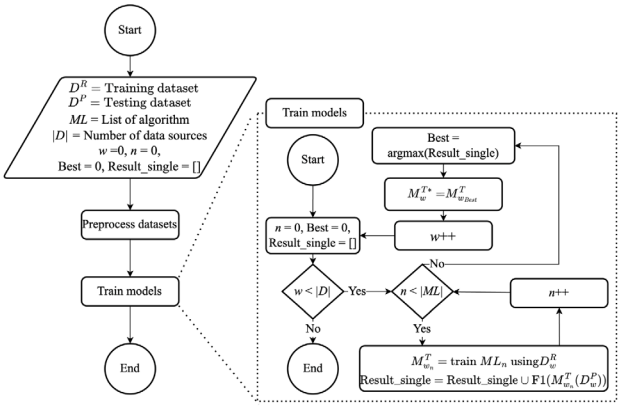

Fig. 7. Trained ML models from each data source.

then classified to their lifecycles using sequence matching. The training, ensemble, and sequence matching processes are detailed in Sections Section 4.2, 4.3, and 4.5, respectively.

4.2. Trained ML models from each data source

Fig. 7 shows the procedure for training and testing ML models using data from three data sources. During the training phase, we utilize several ML algorithms, leading to the development of an array of ML models. The models generated from this phase are utilized for both the single-stage ML approach and the initial stage of the two-stage models.

Before training, the datasets undergo preprocessing, which includes steps such as one-hot encoding for categorical features, data cleaning, removal of duplicated features, and feature scaling. In the model training phase, each algorithm from the list ML is trained using the designated data source. The model's performance is assessed on the testing dataset, D^P, and the best results are stored in `Result_single`. If a model outperforms others, the `Best` value is updated. The process iterates through all data sources and algorithms, ensuring every combination is evaluated. By the end, the top-performing model is identified as $M^T *$.

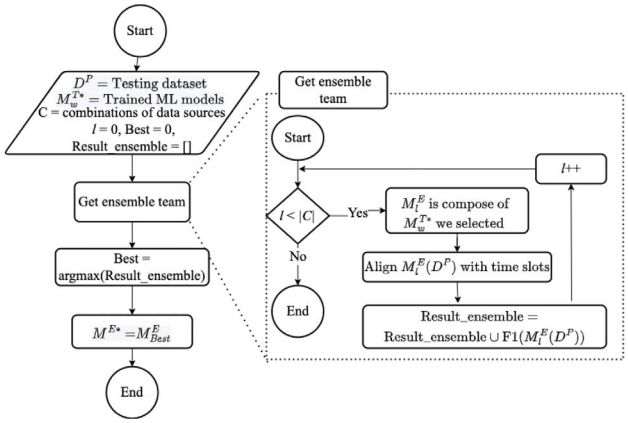

Fig. 8. Procedure of creating the ensemble team.

4.3. Ensemble method

The rationale behind using the ensemble method is to mitigate the effects of individual model weaknesses or errors from each data source (Zhang et al., 2022). Fig. 8 illustrates the procedure of creating an ensemble team and assessing its performance. Inputs include the best ML model for each data source from Section 4.2 and a testing dataset across three data source types.

In this study, the diversity of the models in the ensemble is inherently enhanced by the varied nature of the data sources. Each model, trained on this heterogeneous data, develops unique strengths and weaknesses, contributing to a more robust ensemble. By integrating these diverse models, our ensemble approach gains a comprehensive perspective, increasing its robustness and accuracy in detecting and classifying complex cyber threats. This method capitalizes on the diverse data characteristics to ensure a more effective and nuanced threat detection system.

Ensemble teams are formed by combining classifiers trained with different ML models and data sources. There are four possible ensemble team combinations: (1) models from traffic, system log, and host statistics, (2) models from traffic and system log, (3) models from system log and host statistics, and (4) models from traffic and host statistics. Each model is evaluated using its respective data source. The predicted outcomes and ground truths are subsequently synchronized according to their time slots.

A voting mechanism is then employed to produce consolidated results for each time slot and the F1 score of the ensemble team is computed for all ensemble teams. The output is the ensemble team exhibiting the highest F1 score.

Fig. 9 provides an illustrative example of how these results are merged. For every data point in the dataset, there are three types of data sources per time slot (e.g., 1s). Each data source is fed into its corresponding pre-trained model to derive a detection result. We then calculate the count of both the prediction result labels and the actual labels for each time slot. We implement soft voting for each category which aggregates the predicted results for each class across all the models, and then the class with the highest average score is taken as the final result. Due to the typically higher prevalence of the benign class (e.g., label 0), a predetermined threshold for this class is set. If the benign class surpasses this threshold, the data is classified as benign for that time slot. If not, it is classified under the class with the highest representation between the other classes. We perform exhaustive searching to identify the optimal threshold. The searching method is not just to achieve the highest F1 score, but also to ensure that, under this threshold, the detection results do not exclusively consist of label 0 or entirely lack label 0.

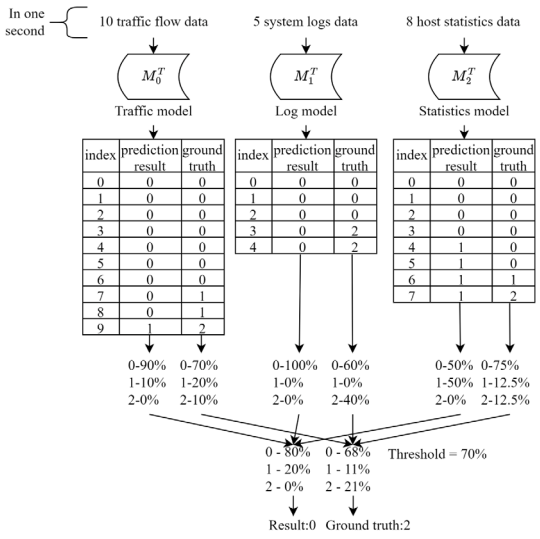

Fig. 9. Illustration of ensemble method.

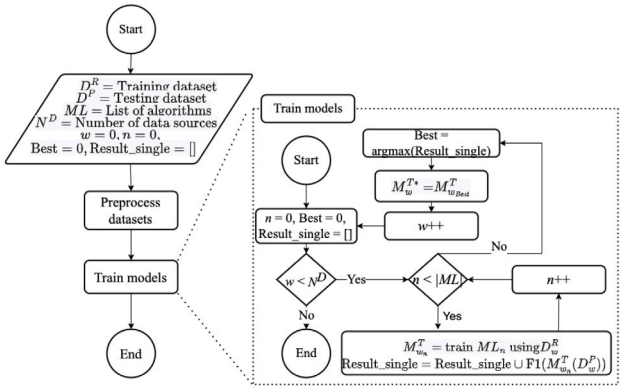

Fig. 10. Trained lifecycle detection model.

4.4. Trained lifecycle detection model

Fig. 10 depicts the process of employing ML models to classify testing data. The generated models are then used in the second stage of the ML+ML method. Both training and testing datasets are used as input, associating each technique with a lifecycle label. Prior to use, these datasets undergo preprocessing. We select technique sequences based on lifecycle timestamps and label them for training. We also remove consecutive duplicate techniques to prepare the sequences for matching and training.

For each model, it is trained with the training dataset and subsequently tested with the testing dataset to calculate the F1 score. The F1 scores obtained from testing with each model are compiled into a list, and the model with the highest F1 score is selected. The outcome of this process is the classifier that achieves the highest F1 score for identifying the lifecycle.

4.5. Sequence matching method

The method used from this phase are utilized for the second stage of ML+SM method. Fig. 11 describes the process of evaluating a testing dataset using a similarity metric based on edit distance (Levenshtein et al., 1966) and predefined lifecycle sequences. The inputs include the similarity metric, the predefined sequences, and the testing dataset. The predefined sequences are derived from the order of reproduced attack scenarios in the CREMEv2 dataset. We provide a detailed explanation

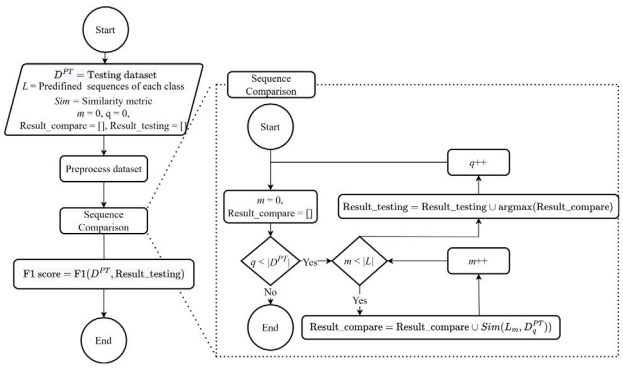

Fig. 11. Sequence matching method.

and the example of these sequences in Section 5.2. After preprocessing the dataset, each sequence is compared to the lifecycle patterns, assigning the class with the maximum similarity. This is repeated for all sequences, resulting in an F1 score for the method.

Let S_1 and S_2 be two attack technique sequences. Specifically, $ED(S_1, S_2)$ is the edit distance of two sequences S1 and S2 proposed by Levenshtein et al. (1966) and $max(len(S_1), len(S_2))$ represents the longest length between S_1 and S_2. The similarity metric between sequences S_1 and S_2 can be described as

$$Sim(S_1, S_2) = 1 - \frac{ED(S_1, S_2)}{max(len(S_1), len(S_2))}. \quad (1)$$

$Sim(S_1, S_2)$ can be used to compute similarity between predefined sequences of each lifecycle class and used to detect technique sequence, and do classification based on the similarity of each class.

5. Implementation

In this section, we present the implementation of our solution. First, we summarize the open sources and tools used for the implementation. Second, we focus on describing the details of the implementation.

5.1. Open sources and tools

We classify open sources and tools into two categories: dataset and libraries. We describe the details of the categories as follows.

Dataset

In this study, we employed CREMEv2 (Yudha, 2023) as the dataset to train our ML models. CREMEv2, an upgrade version of CREMEv1 (Bui et al., 2021), distinguishes itself in two ways: a broader array of attack techniques across lifecycles and multiclass labeling representing both attack techniques and lifecycles, in contrast to the binary labels of CREMEv1. This toolchain, based on ATT&CK, automates the reproduction of attack techniques and lifecycles using Metasploit as the tool to launch the techniques. It encompasses five attack lifecycles, 17 techniques, and three data types: network traffic, system logs, and host statistics, as explained in Section 2.

The five attack scenarios in CREMEv2 encapsulate a wide spectrum of behaviors, from network-level anomalies (Mirai) to system-level disruptions (disk wipe), thereby providing a robust testing ground for the ML-IDS. The feature learning and detection mechanisms of the proposed approach focus on underlying patterns and anomalies that are not strictly attack-specific, enhancing its potential to generalize across a multitude of attack types.

In CREMEv2, network traffic was captured via Tcpdump and converted into numerical data using Argus for ML model input. System logs were collected using Rsyslog and then parsed with the Drain tool. These logs were subsequently transformed into log templates suitable for ML

models capable of Natural Language Processing (NLP) through the Text Tokenization technique. Host statistics, obtained with Atop for Linux system monitoring, were filtered to remove unusable features, creating numerical input for models. All the data sources were collected over a duration of 3 h and 29 min.

CREMEv2, as the updated version of CREMEv1, adds a router machine and constructs a host-only network, isolating the primary operating system from the virtual machines to lessen the effects of external threats. The constructed testbed includes ten virtual entities: a controller, a data logger, four client machines, and a single machine that plays the roles of an attacker, a target, and a benign server. The intention behind this configuration is to foster a controlled setting that is optimal for simulating various types of cyber attacks, while also ensuring comprehensive data collection and logging.

In terms of executing attack scenarios, CREMEv2 employs an array of attack tools that are in line with the principles of the MITRE ATT&CK framework. This range encompasses tools for simulating botnets, executing disk wipes, initiating ransomware, conducting resource hijacking, and launching endpoint DoS attacks. For instance, it uses a pre-compiled version of Mirai for botnet scenarios, Metasploit for advanced access techniques, and bespoke scripts for ransomware and resource hijacking. These tools are meticulously selected to correspond with distinct attack methods according to the framework, facilitating precise replication of cyber threat behaviors.

The procedure for generating the dataset starts with the automated enactment of these attack scenarios through the tools described above. Command and coordination of the attack simulations are managed by the central controller server, which oversees the interplay between all participating entities. As the attacks proceed, real-time data is captured from the clients, the targeted server, and the benign server, which is then routed to the data logger for unified logging. This centralized collection system simplifies the data handling process and supports the creation of detailed datasets that encapsulate the complexity of each cyber attack scenario. Post data acquisition, an elaborate labeling process is conducted based on the breakpoint information collected during the attack stages.

The data distribution for different techniques across each data source is shown in Table 3. We then used 80% of the data as training, 10% as validation, and 10% as testing. This table highlights a noticeable class imbalance, with some classes having significantly fewer instances. To rectify this and improve our models' F1 scores, we applied the SMOTE technique to balance the data classed within each data source. It generates synthetic samples for the minority class by interpolating existing minority instances, creating a more balanced dataset for machine learning model training.

Hardware and software

For our experiments, we utilized specific hardware and software configurations. The hardware specifications included an AMD Ryzen 7 5800-X 8-core processor and 128 GB of RAM. The operating system used was Ubuntu 22.04, and no GPU was installed.

In terms of software, we employed Python 3.8 as the primary programming language. To address dataset imbalance, we utilized the open-source library Imbalanced-learn 0.10.1, which offers resampling techniques. For ML tasks, Scikit-learn 1.2.0 (Pedregosa et al., 2011) served as our main library, providing various algorithms such as decision trees and SVMs. Additionally, we utilized XGBoost 1.7.3, a gradient-boosting decision tree algorithm, as one of our ML models.

For DL tasks, we leveraged Tensorflow 2.6.5. (Abadi et al., 2015). To visualize our results effectively, we utilized Seaborn 0.12.2 and Matplotlib 3.6.2. These software tools collectively facilitated the execution of our experiments and the analysis of our results.

Table 3
Summary of data labels for techniques.

Label	Data source		
	Network traffic	Accounting	Syslog
0	50 000	59 297	2572
1	50 000	3967	1955
2	176	174	408
3	–	32	–
4	404	467	415
5	–	32	–
6	46	207	264
7	–	32	87
8	32	24	302
9	12	96	–
10	32	24	86
11	606	10 500	–
12	–	30	98
13	12 647	–	–
14	–	–	122
15	–	20	139
16	–	32	83
17	–	32	181

5.2. Implementation details

In this section, we detail the solution's implementation as discussed in Section 4. We first preprocessed our dataset for ML model training. This involved cleaning data, selecting appropriate features for each data source, and standardizing the data.

For system logs, we employed label encoding and sequenced event templates every second. Initially, we extracted useful information from unstructured log files using the Drain tool. This process yielded a log template for each Syslog (Lin et al., 2022). Next, each template was assigned a unique number, in a process we referred to as label encoding. For instance, if there are three log templates (template A, template B, and template C), they are labeled 1, 2, and 3, respectively. With templates converted into numbers, we then collected all log template numbers in sequence for every second. This sequence of log template numbers serves as our data for training and testing.

For both host statistics and network traffic data, we first applied one-hot encoding to non-numerical data. Following this step, we proceeded to standardize the features. Subsequently, any duplicated or non-varying features were eliminated. This process resulted in 38 features for network traffic and 36 features for host statistics. After processing the datasets, we divided them into training and testing sets. The training set was balanced using the SMOTE (Chawla et al., 2002) method.

Furthermore, for host statistics and network traffic data, which primarily consist of numerical features, we employed standard ML models. Current literature suggests that these models are highly effective for such datasets, delivering robust performance with relatively low computational demands (D'hooge et al., 2020). However, for a comprehensive and fair assessment, we also included Deep Neural Networks (DNN) as a comparative benchmark.

Moreover, system logs predominantly contain textual information. Analyzing sequential textual data requires models that can understand context and the inherent order of the sequence. Thus, we employed DL models specifically designed for such tasks, including Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU). These models possess the capability to perform intricate natural language processing techniques, making them well-suited for analyzing system logs (Sworna et al., 2023; Satilmiş et al., 2024).

After acquiring pre-trained ML models for each data source, the next step involves ensemble methods. These methods combine ML models from different data sources using diverse ML algorithms. Given that each data source records information at varying intervals, we

synchronize detection results and associated ground truths by merging them at one-second intervals. To consolidate these results, we employ the soft voting technique.

After voting, we applied a threshold to determine whether the prediction result for each second belongs to class 0 or not. If the percentage of class 0 for each data source within one second exceeds the threshold, we labeled it as 0; otherwise, we selected the label with the highest percentage among the remaining labels. The percentage is determined by the number of occurrences of each prediction result divided by the total number of predictions in each second.

Fig. 9 provides a detailed illustration of the ensemble process. For example, if network traffic produces 10 detection results with corresponding ground truths in one second, they are merged into a single detection result with its corresponding ground truth. If between those 10 results, label 0 appears more than 9 times, we assign label 0 as the detection result for that second, applying the same process to the ground truths.

Finally, we assessed the performance of the ensemble teams using the F1 score. This F1 score has been primarily utilized as the evaluation metric due to its balanced consideration of both precision and recall, crucial in the context of IDS. These IDS environments demand a metric that equally values the correct identification of attacks (precision) and the system's ability to identify as many attacks as possible (recall). Given the intricate nature of attack lifecycles and techniques, which often involve overlapping and nuanced behaviors, the F1 score provides a more comprehensive measure of a model's accuracy than either precision or recall alone. Hence, the F1 score is an appropriate and efficient indicator for assessing our model's efficacy in detecting complex cyber threats. We then evaluated seven different combinations of data sources to determine the best-performing ensemble team.

Furthermore, to generate technique sequences for lifecycle detection, we fed multiple data sources into the ensemble model, and then sorted its detection results by seconds. Based on the start and end timestamps of each attack lifecycle, we extracted the technique sequences. These sequences resemble a series of numbers, with each number representing a predicted technique every second; '0' denotes no attack technique, while other numbers correspond to specific techniques.

However, the original datasets present a challenge due to their limited sequence data, covering only five lifecycles. To address this limitation, we enhanced the training data volume for the DL models used in lifecycle detection by adjusting the number of 'label 0' in a sequence, thereby varying the intervals between attack techniques. For instance, a lifecycle sequence such as '10002220033' was augmented to '1022200000033'. This approach primarily impacted the number of label 0 in the sequences, but we took care to retain essential features within the lifecycles.

After refining the technique sequence dataset, we removed label 0 and any consecutive duplicate techniques before using the data to train and test the learning models for the second stage of the ML+ML approach. Treating lifecycle detection as an NLP task, we selected several DL models for evaluation. These models were trained and tested using the sequence datasets, and their performance was evaluated using the F1 score. Finally, we chose the model with the highest F1 score to handle this task.

For lifecycle detection, we also utilized sequence matching. Initially, we designated the original technique sequence as the pattern for each lifecycle class. During testing, technique sequences from the dataset were chosen and preprocessed to align with the patterns. The process of sequence matching is illustrated in Fig. 12. We preprocessed the technique sequence derived from the first stage ML model by eliminating occurrences of label 0 and merging consecutive identical numbers. After preprocessing, we compared the sequence with the pattern sequences. We employed edit distance, calculated through dynamic programming, to measure similarity between the selected and pattern sequences. The most similar lifecycle class was then chosen as the detection result.

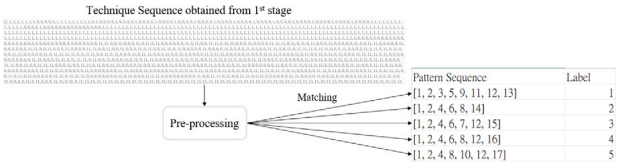

Fig. 12. An Example of Sequence Matching.

Table 4
Hyperparameter settings for ML.

Classifier	Hyperparameter	Value
Logistic Regression	C	1
Decision Tree	max depth	8
	min sample leaf	1
	min sample split	2
SVM	C	1.15
Random Forest	max depth	11
	min sample leaf	1
	min sample split	2
Bagging Ensemble	max samples	2
Gradient Boosting	max depth	8
	min sample leaf	1
	min sample split	2
XGBoost	objective	multi:softprob
	eval metric	merror
	booster	gbtree
	learning rate	0.3
	gamma	0.2
	max depth	9
	min child weight	3

6. Experiment results

This section begins with a description of the hyperparameter configuration for all classifiers that we used. Then, the rest of this section shows the experiment results to address the issues introduced in Section 1.

6.1. Experimental setup

We employed a range of ML and DL models for each data source, selecting widely used options in the ML-IDS domain. We fine-tuned the hyperparameters of network traffic and host statistics models through a grid search, ensuring optimal performance. The key hyperparameters for these models are detailed in Table 4.

In addition to ML models, we incorporated several DL models. The DNN model was employed for both host statistics and network traffic data sources, while the other models were used for system logs, technique sequences data sources, and lifecycle detection. Table 5 shows the architectures of those neural networks.

6.2. Single- vs. Two-stage ML

This section assesses three lifecycle detection methods (ML, ML+ML, ML+SM) outlined in Section 4 to identify the most effective one. In the initial ML stage, we employed XGBoost for network traffic and host statistics, LSTM for system logs, and Bi-LSTM for ML+ML's second stage. Fig. 13 compares F1 scores for attack lifecycle detection among these approaches. The standalone ML method scored 0.887, indicating solid performance. However, it is surpassed by the two-stage ML approach (ML+ML). The single-stage method struggles with distinguishing techniques across multiple lifecycles, potentially causing misclassification. In contrast, the ML+ML method excels with a 0.994 F1 score. Here, the second-stage model learns the technique sequences generated by the first-stage model. Even if the first-stage

Table 5
DL model architectures.

Classifier	Layer	Type	Output shape	Activation function
DNN	1	Dense	(256)	relu
	2	Dense	(128)	relu
	3	Dense	(64)	relu
	4	Dense	(32)	relu
	5	Dense	(number of labels)	softmax
RNN	1	Embedding	(1950, 16)	
	2	SimpleRNN	(1950, 64)	tanh
	3	SimpleRNN	(32)	tanh
	4	Dense	(number of labels)	softmax
LSTM	1	Embedding	(1950, 16)	
	2	LSTM	(1950, 64)	tanh
	3	LSTM	(32)	tanh
	4	Dense	(number of labels)	softmax
Bi-LSTM	1	Embedding	(1950, 16)	
	2	Bi-LSTM	(1950, 128)	tanh
	3	Bi-LSTM	(64)	tanh
	4	Dense	(number of labels)	softmax
GRU	1	Embedding	(1950, 16)	
	2	GRU	(1950, 64)	tanh
	3	GRU	(32)	tanh
	4	Dense	(number of labels)	softmax
Bi-GRU	1	Embedding	(1950, 16)	
	2	Bi-GRU	(1950, 128)	tanh
	3	Bi-GRU	(64)	tanh
	4	Dense	(number of labels)	softmax

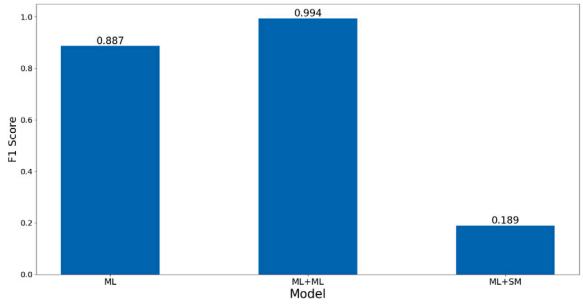

Fig. 13. Performance of three approaches.

models misclassify some techniques, the second-stage model incorporates these errors as features within the technique sequences for lifecycle recognition, enhancing overall performance.

Conversely, the ML+SM approach yields the lowest performance, achieving only an F1 score of 0.189. There are two primary reasons for the poor performance of this method. Firstly, the comparison is based on pattern sequences, heavily relying on the ML models' ability to correctly recognize techniques from data sources and merge them accurately to form the correct sequence. Second, the similarity in sequence patterns poses a challenge. As observed in Fig. 1, the initial four attack techniques for ransomware, resource hijacking, and disk wipe are identical, making it challenging to precisely match the identified technique sequence to the correct lifecycle.

Moreover, Fig. 14 presents the confusion matrices for these three approaches. The confusion matrix for the single-stage ML approach reveals that only benign data can be correctly classified, with a true positive rate of 0.98. In contrast, the true positive rates for the other lifecycle classes only reach 0.78. This confirms our earlier explanation that the presence of techniques shared across different attack lifecycles poses challenges for accurate detection. While the single-stage ML approach achieves a high overall F1 score, it is evident that this method struggles to precisely classify lifecycles.

Because a technique can be used in several lifecycles, it posed challenges for single-stage ML and ML+SM. In single-stage ML, without the context of technique sequences, the method struggled to accurately classify lifecycles. Similarly, ML+SM faced difficulties due to the similarity between technique sequences from different lifecycles, leading to misclassification.

However, the two-stage ML+ML approach effectively addressed this issue by using ML in the second stage. This allowed the model to learn and differentiate the unique sequences of techniques associated with each specific lifecycle. By understanding the intricate patterns of technique sequences within each lifecycle, the two-stage ML+ML approach achieved more accurate and reliable classification results.

Table 6 shows the detection times for various methods applied to a dataset consisting of 6000 data points. These methods were assessed sequentially to evaluate their efficiency in a controlled environment. The detection time, measured in seconds, spans several categories: Host Statistics, System Log, Network Traffic, Ensemble on Lifecycles, Ensemble on Techniques, ML on Lifecycles, and Sequence Matching, leading to the total time. The single-stage ML method recorded detection times of 23.9, 36.2, and 30.8 s for Host Statistics, System Log, and Network Traffic, respectively, with a total aggregated time of 93.95 s. The ML+ML approach, which integrates ML for both technique and lifecycle detection, showed slightly higher times due to the added complexity, totaling 98.4 s. The ML+SM method, combining machine learning with sequence matching, marked the longest detection time at 101.76 s, reflective of the additional time required for sequence matching. Each time value presented is the mean from multiple tests, denoted alongside their standard deviations, ensuring the reliability and reproducibility of the results. This detailed detection time analysis helps in understanding the efficiency and resource utilization of each proposed detection method.

Furthermore, a direct comparison of results with previous works is not feasible due to distinct methodologies and objectives. Earlier studies either focused on binary detection at the technique level or on identifying attacks without considering their lifecycle stages. In contrast, our approach is unique with its two-stage attack detection process. We initially detect individual techniques and then analyze their sequences to classify attack lifecycles. This multi-class, sequential analysis differs fundamentally from past works that employ binary classification or a single-stage approach.

6.3. Learning models on lifecycle detection

This experiment seeks to identify the best model for lifecycle detection within our ML+ML approach. We utilized the dataset from the first stage of both ML+ML and ML+SM approaches for this purpose. To overcome limited lifecycle data points, we augmented the dataset by adjusting the number of label 0 in each sequence. This expansion yielded 357 data points for each lifecycle.

Fig. 15 shows the performance of various DL models in classifying the lifecycle. Notably, bidirectional models such as Bi-LSTM and Bi-GRU outperform their unidirectional counterparts. This suggests their capacity to capture more intricate patterns in the data, with the highest-performing model achieving an F1 score of 0.994.

Overall, these results underscore the effectiveness of employing deep learning models, specifically those based on LSTM and GRU, for classifying attack lifecycles from technique sequences. Particularly, when we enabled these models to learn contextual relations bidirectionally, the performance is further enhanced.

6.4. Single- vs. Multi-datasource

In this section, we assess the potential benefits of integrating multiple data sources for detection tasks. Specifically, we aim to determine if these sources, when combined, can enhance performance and whether they complement each other.

After obtaining the detection results from each data source, we employed ensemble techniques to integrate these results across multiple

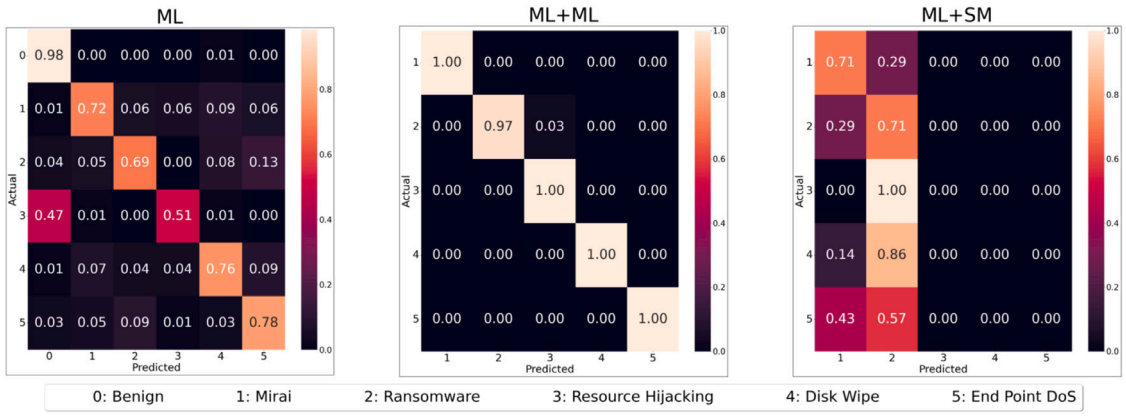

Fig. 14. Confusion matrices of three approaches.

Table 6
Testing time (in seconds) for various methods.

Method	Testing Time (in seconds)							
	Host statistics	System Log	Network traffic	Ensemble on lifecycles	Ensemble on techniques	ML on lifecycles	Sequence matching	Total time
ML	23.9 ± 0.35	36.2 ± 0.25	30.8 ± 0.44	3.05 ± 0.005	–	–	–	93.95
ML+ML	24.3 ± 0.29	35.7 ± 0.6	32 ± 0.26	–	3.06 ± 0.003	3.34 ± 0.018	–	98.4
ML+SM	25.7 ± 0.32	34.5 ± 0.44	36 ± 0.53	–	3.26 ± 0.01	–	2.3 ± 0.013	101.76

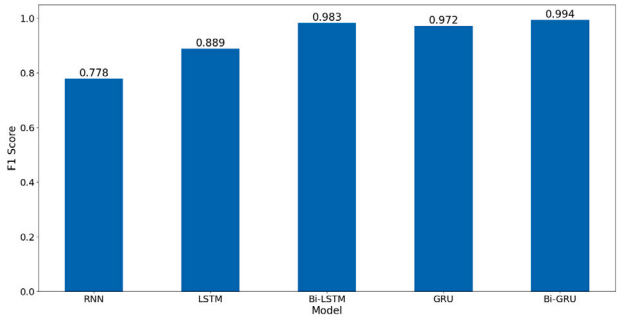

Fig. 15. Model evaluation for lifecycle.

Table 7
Aggregated data label threshold.

Data sources	Threshold
T	0.98
L	1
A	0.87
T+L	0.81
T+A	0.863
L+A	0.84
T+L+A	0.805

T/L/A: Traffic/System Log/Accounting

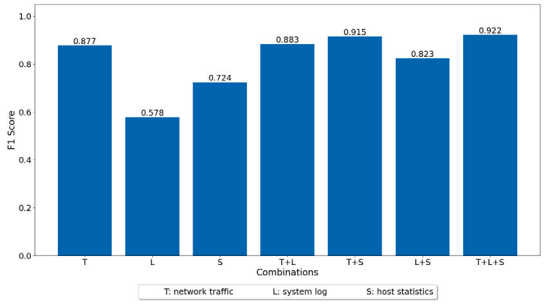

Fig. 16. Evaluation for single- vs. multi-datasource.

ML models and data sources. These data points were aggregated into one-second time slots to facilitate the ensemble process. We conducted an exhaustive search to determine the optimal threshold for classifying aggregated data as benign or indicative of attack techniques. If the number of data points within one second exceeded this threshold, the data point was classified as benign. Otherwise, it was classified according to the highest representation among the other technique classes. To ensure a fair comparison between multi-datasource and single-datasource approaches, we merged data points from individual sources in a similar manner. The optimal thresholds, presented in Table 7, were selected not only to achieve the highest F1 score but also to ensure that detection results maintain a balance between label 0 and other labels. For instance, if there are 100 data points for network traffic within a second, they would be consolidated and classified as label 0 if 98 of those data points are detected as label 0.

The results from Fig. 16 confirm that combining various data sources enhances the performance. Among the data sources, network traffic is predominant with an F1 score of 0.877.

The ensemble utilizing all three data types outperforms the seven combinations, achieving an F1 score of 0.922. Furthermore, every ensemble team that includes more than one data sources outperforms each individual data source within that team. This strongly supports the notion that leveraging multiple data sources significantly improves the detection performance.

6.5. Learning models on technique detection

In this section, we analyze the performance of both ML and DL models in detecting techniques across different data sources. As previously explained in Section 5.2, when conducting technique detection tests for each data source, we merged the data points into one-second intervals. This approach was taken to simplify the comparison between various combinations of single and multiple data sources.

We also examine the training times of these models on each data source, which is critical for understanding the computational efficiency of the models in addition to their detection performance. The training times, measured in minutes, are summarized in Table 8.

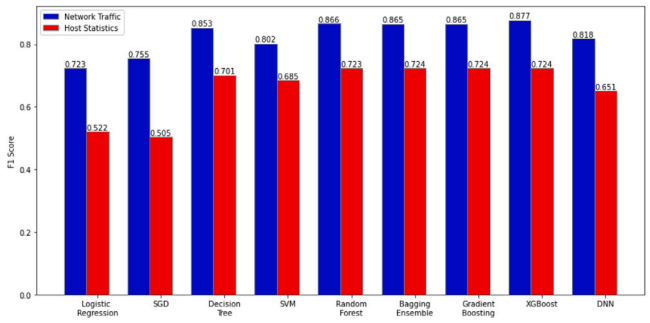

Fig. 17. Model evaluation for network traffic and host statistics in technique detection.

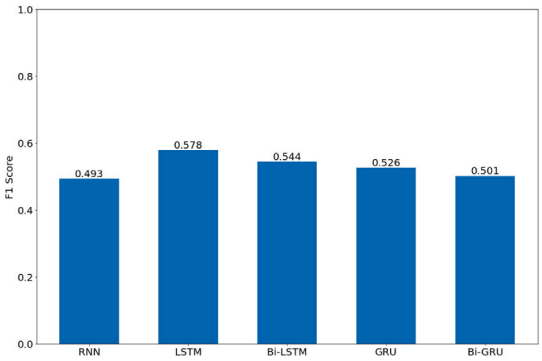

Fig. 18. Model evaluation for system logs in technique detection.

Technique detection from network traffic

Fig. 17 illustrates the performance of ML models in classifying network traffic. It is evident that the majority of models excel, signifying their adeptness in capturing the correlation between labels and features.

Although most models exhibit strong performance, we observed a slight superiority between tree-based models. This indicates their suitability for network traffic analysis. XGBoost emerged as the top-performing model among these, achieving an impressive F1 score of 0.877.

The training times for traffic data, as detailed in Table 8, reveal that while SVM and Gradient Boosting models require the longest training periods due to their complex computations, XGBoost stands out for its high F1 score combined with reasonable training time. This balance makes XGBoost particularly appealing for applications requiring both high accuracy and computational efficiency.

Technique detection from host statistics

Fig. 17 shows the ML model performance for host statistics. Notably, certain models such as stochastic gradient descent (SGD) and logistic regression exhibit a decline of at least 0.2 in their F1 scores compared to the remaining models, possibly due to their simpler structures failing to capture the intricate relationships between the features. Conversely, tree-based classifiers, specifically bagging ensemble, gradient boosting, and XGBoost, perform exceptionally well, achieving an F1 score of 0.724. Remarkably, even a standalone decision tree reaches an F1 score of 0.701, underscoring the efficacy of tree-based models. Their ability to decode complex non-linear relationships through the if-else rules, combined with computational efficiency, makes them an ideal choice in this scenario.

The training times for host statistics data in Table 8 highlight the efficiency of tree-based models, such as Decision Tree, Bagging Ensemble, and XGBoost, not only in their superior F1 scores but also in their relatively short training times, making them highly suitable for rapid model development cycles.

Technique detection from system logs

Fig. 18 shows the performance of various DL models when applied to system logs. Despite being used for NLP tasks, these models underperformed, with the highest F1 score being only 0.578 from LSTM.

The significant factor influencing this outcome is the dataset labeling process. After extracting data from system logs, we labeled it using the attacker's hostname in conjunction with the attack's start and end times. For instance, if a data point contains the attacker's hostname and falls within the attack process timeframe, we label it an attack. However, some benign log data might also include the attacker's hostname. This inclusion introduces outliers into our labeling, affecting label quality. To enhance the performance, exploring alternative methods of labeling the data is crucial moving forward.

Table 8
Training times of models across each data source, measured in minutes.

Model	Data source		
	Traffic	Host statistics	System Logs
Logistic Regression	2.22	4.76	
SGD	0.06	0.07	
Decision Tree	0.03	0.02	
SVM	34.07	96.98	
Random Forest	0.11	0.15	–
Bagging Ensemble	0.02	0.14	
Gradient Boosting	39.37	38.70	
XGBoost	6.82	9.85	
DNN	12.50	21.18	
RNN			105.48
LSTM			158.67
Bi-LSTM	–	–	131.75
GRU			195.75
Bi-GRU			146.33

The training times for system logs, detailed in Table 8, reveal that DL models, particularly RNN and its variants, necessitate longer training periods. This underscores the importance of considering computational efficiency alongside model accuracy, especially in real-world applications where rapid processing of logs is paramount.

7. Conclusions and future work

In this study, we assessed three lifecycle detection approaches using a combination of three data sources. Results varied considerably among the methods. The single-stage ML model achieved a respectable F1 score of 0.887 but lacked insight into technique sequences. In contrast, the ML+ML approach, featuring a two-stage learning process, excelled with an impressive F1 score of 0.994. However, the ML+SM method, reliant on sequence matching, underperformed due to sequence discrepancies, yielding an F1 score of only 0.189.

Furthermore, an ensemble approach that combined all the data sources yielded an F1 score of 0.922 on technique detection, emphasizing the value of leveraging multiple data sources for intrusion detection.

Our findings also highlight the robust detection capabilities of ML and DL models for detecting attack techniques. Notably, tree-based models, especially XGBoost, excelled in network traffic and host statistics, achieving F1 scores of 0.877 and 0.724, respectively.

In real-world scenarios, these findings enable comprehensive intrusion detection, offering a holistic threat perspective. Employing multiple data sources ensures thorough network monitoring, reducing the risk of undetected breaches, especially in complex attack scenarios.

However, this proposed approach comes with limitations. Coordinating multiple data sources requires careful synchronization and

integration. Challenges may arise concerning data privacy, storage, and real-time processing. The ensemble approach, while comprehensive, demands more computational resources, impacting scalability in larger networks. Organizations must balance these benefits and limitations to optimize method application in their specific contexts.

In future work, there are two directions that can be explored. Firstly, expanding our detection capabilities to include unknown or emerging attack vectors, moving beyond the known lifecycles we currently focus on. This involves exploring new methodologies to identify novel attack patterns. Secondly, investigating the resilience of our models against adversarial attacks is crucial. Enhancing model robustness against such manipulations will ensure the reliability of intrusion detection systems in the face of evolving cyber threats.

CRediT authorship contribution statement

Ying-Dar Lin: Conceptualization, Funding acquisition, Methodology, Project administration, Supervision. **Shin-Yi Yang:** Writing – original draft, Visualization, Validation, Software, Formal analysis, Data curation. **Didik Sudyana:** Validation, Resources, Methodology, Investigation, Formal analysis, Data curation, Visualization, Writing – review & editing. **Fietyata Yudha:** Validation, Methodology, Formal analysis, Data curation. **Yuan-Cheng Lai:** Supervision, Formal analysis, Conceptualization, Validation. **Ren-Hung Hwang:** Conceptualization, Formal analysis, Methodology, Supervision, Validation.

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.

Data availability

No data was used for the research described in the article.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. [Online]. Available: https://www.tensorflow.org/, Software available from tensorflow.org.

Affinito, A., Zinno, S., Stanco, G., Botta, A., Ventre, G., 2023. The evolution of Mirai botnet scans over a six-year period. J. Inform. Secur. Appl. 79, 103629.

Alshamrani, A., Myneni, S., Chowdhary, A., Huang, D., 2019. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. IEEE Commun. Surv. Tutor. 21 (2), 1851–1877.

Bagui, S., Mink, D., Bagui, S., Ghosh, T., McElroy, T., Paredes, E., Khasnavis, N., Plenkers, R., 2022. Detecting reconnaissance and discovery tactics from the MITRE ATT&CK framework in Zeek Conn logs using Spark's machine learning in the big data framework. Sensors 22 (20), 7999.

Beaman, C., Barkworth, A., Akande, T.D., Hakak, S., Khan, M.K., 2021. Ransomware: Recent advances, analysis, challenges and future research directions. Comput. Secur. 111, 102490.

Bui, H.-K., Lin, Y.-D., Hwang, R.-H., Lin, P.-C., Nguyen, V.-L., Lai, Y.-C., 2021. CREME: A toolchain of automatic dataset collection for machine learning in intrusion detection. J. Netw. Comput. Appl. 193, 103212.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. SMOTE: Synthetic minority over-sampling technique. J. Artif. Intell. Res. 16, 321–357.

D'hooge, L., Wauters, T., Volckaert, B., De Turck, F., 2020. Inter-dataset generalization strength of supervised machine learning methods for intrusion detection. J. Inform. Secur. Appl. 54.

Du, M., Li, F., Zheng, G., Srikumar, V., 2017. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17, Association for Computing Machinery, New York, NY, USA, pp. 1285–1298. http://dx.doi.org/10.1145/3133956.3134015.

Gomes, G., Dias, L., Correia, M., 2020. CryingJackpot: Network flows and performance counters against cryptojacking. In: 2020 IEEE 19th International Symposium on Network Computing and Applications. NCA, pp. 1–10. http://dx.doi.org/10.1109/NCA51143.2020.9306698.

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

Hamid, Y., Sugumaran, M., Balasaraswathi, V., 2016. Ids using machine learning-current state of art and future directions. Brit. J. Appl. Sci. Technol. 15 (3).

Hoque, N., Bhuyan, M.H., Baishya, R.C., Bhattacharyya, D.K., Kalita, J.K., 2014. Network attacks: Taxonomy, tools and systems. J. Netw. Comput. Appl. 40, 307–324.

Hwang, R.-H., Lee, C.-L., Lin, Y.-D., Lin, P.-C., Wu, H.-K., Lai, Y.-C., Chen, C., 2023. Host-based intrusion detection with multi-datasource and deep learning. J. Inform. Secur. Appl. 78, 103625. http://dx.doi.org/10.1016/j.jisa.2023.103625.

Kaloudi, N., Li, J., 2020. The AI-based cyber threat landscape: A survey. ACM Comput. Surv. 53 (1), http://dx.doi.org/10.1145/3372823.

Levenshtein, V.I., et al., 1966. Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Doklady 10 (8), 707–710.

Lin, Y.-D., Wang, Z.-Y., Lin, P.-C., Nguyen, V.-L., Hwang, R.-H., Lai, Y.-C., 2022. Multi-datasource machine learning in intrusion detection: Packet flows, system logs and host statistics. J. Inform. Secur. Appl. 68, 103248.

Liu, J., Simsek, M., Kantarci, B., Bagheri, M., Djukic, P., 2022. Collaborative feature maps of networks and hosts for AI-driven intrusion detection. In: GLOBECOM 2022-2022 IEEE Global Communications Conference. IEEE, pp. 2662–2667.

Meng, W., Liu, Y., Zhang, S., Pei, D., Dong, H., Song, L., Luo, X., 2018. Device-agnostic log anomaly classification with partial labels. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service. IWQoS, IEEE, pp. 1–6.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., et al., 2019. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: IJCAI, vol. 19, (no. 7), pp. 4739–4745.

Operationally Transparent Cyber Dataset, 2019. DARPA. [Online]. Available: https://github.com/FiveDirections/OpTC-data.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Satilmiş, H., Akleylek, S., Tok, Z.Y., 2024. A systematic literature review on host-based intrusion detection systems. IEEE Access 12, 27237–27266.

Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A., 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: International Conference on Information Systems Security and Privacy. [Online]. Available: https://api.semanticscholar.org/CorpusID:4707749.

Singh, S., Sharma, P.K., Moon, S.Y., Moon, D., Park, J.H., 2019. A comprehensive study on APT attacks and countermeasures for future networks and communications: Challenges and solutions. J. Supercomput. 75, 4543–4574.

Strom, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B., 2018. Mitre ATT&CK: Design and Philosophy. Technical Report, The MITRE Corporation.

Sun, P., Yuepeng, E., Li, T., Wu, Y., Ge, J., You, J., Wu, B., 2020. Context-aware learning for anomaly detection with imbalanced log data. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems. HPCC/SmartCity/DSS, IEEE, pp. 449–456.

Sworna, Z.T., Mousavi, Z., Babar, M.A., 2023. NLP methods in host-based intrusion detection systems: A systematic review and future directions. J. Netw. Comput. Appl. 220, 103761. http://dx.doi.org/10.1016/j.jnca.2023.103761.

Toupas, P., Chamou, D., Giannoutakis, K.M., Drosou, A., Tzovaras, D., 2019. An intrusion detection system for multi-class classification based on deep neural networks. In: 2019 18th IEEE International Conference on Machine Learning and Applications. ICMLA, IEEE, pp. 1253–1258.

Wang, W., Yi, P., Jiang, J., Zhang, P., Chen, X., 2024. Transformer-based framework for alert aggregation and attack prediction in a multi-stage attack. Comput. Secur. 136, 103533.

Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y., Han, H., 2022. A systematic literature review of methods and datasets for anomaly-based network intrusion detection. Comput. Secur. 116, 102675.

Yudha, F., 2023. CREMEv2: A toolchain of automatic dataset collection for machine learning in intrusion detection based on MITRE ATT&CK. [Online]. Available: https://github.com/masjohncook/CREMEv2.

Yue, H., Li, T., Wu, D., Zhang, R., Yang, Z., 2024. Detecting APT attacks using an attack intent-driven and sequence-based learning approach. Comput. Secur. 140, 103748.

Zavrak, S., İskefiyeli, M., 2020. Anomaly-based intrusion detection from network flow features using variational autoencoder. IEEE Access 8, 108346–108358.

Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., Yang, A., 2022. Comparative research on network intrusion detection methods based on machine learning. Comput. Secur. 121, 102861.

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 cofounded L7 Networks Inc. in 2002 and O'Prueba Inc. in 2018. His research interests include cybersecurity, wireless communications, network softwarization, and machine learning forcommunications. He is an IEEE Fellow (class of 2013). 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, 2017–2020).

Shin-Yi Yang received his M.S. degree at Institute of Network Engineering of National Yang Ming Chiao Tung University (NYCU) in 2023. He was an associate researcher at High Speed Network Lab, NYCU, in 2021–2023. His research interests include cybersecurity and machine learning.

Didik Sudyana is a PhD candidate at Electrical Engineering and Computer Science (EECS) International Graduate Program of National Yang Ming Chiao Tung University (NYCU), Taiwan. He received his M.S. degree in Informatics from Universitas Islam Indonesia (UII), Indonesia, in 2016. He is a lecturer in Informatics at Universitas Sains & Teknologi Indonesia (USTI), Indonesia. His research interests include cybersecurity, machine learning, and network design and optimization.

Fietyata Yudha received an M.S. degree with a digital forensics specialization from Universitas Islam Indonesia (UII) in 2013. He is a lecturer and researcher at Universitas Islam Indonesia and a member of the Center for Digital Forensic Studies. He is currently pursuing a Ph.D. degree in Electrical Engineering and Computer Science(EECS) International Graduate Program at National Yang Ming Chiao Tung University (NYCU), Taiwan. His research interests include cybersecurity, machine learning, and digital forensics.

Yuan-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.

Ren-Hung Hwang is the Dean of the College of Artificial Intelligence, National Yang Ming Chiao Tung University (NYCU), Taiwan. He received his Ph.D. degree in computer science from the University of Massachusetts, Amherst. Before joining NYCU, he was with National Chung Cheng University, Taiwan, from 1993 to 2022. 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 UbiMedia 2018, IEEE SC2 2017, and IEEE IUCC 2014. His current research interest is in Deep Learning, Wireless Communications, Network Security, and Cloud/Edge/Fog Computing.