ARIoTEDef: Adversarially Robust IoT Early Defense System Based on Self-Evolution against Multi-step Attacks

MENGDIE HUANG, Xidian University, China and Purdue University, United States
HYUNWOO LEE, Korea Institute of Energy Technology, South Korea
ASHISH KUNDU, Cisco Systems Inc, United States
XIAOFENG CHEN, Xidian University, China
ANAND MUDGERIKAR, Purdue University, United States
NINGHUI LI, Purdue University, United States
ELISA BERTINO, Purdue University, United States

Internet of Things (IoT) cyber threats, exemplified by jackware and crypto mining, underscore the vulnerability of IoT devices. Due to the multi-step nature of many attacks, early detection is vital for a swift response and preventing malware propagation. However, accurately detecting early-stage attacks is challenging, as attackers employ stealthy, zero-day, or adversarial machine learning to evade detection. To enhance security, we propose ARIoTEDef, an Adversarially Robust IoT Early Defense system, which identifies early-stage infections and evolves autonomously. It models multi-stage attacks based on a cyber kill chain and maintains stage-specific detectors. When anomalies in the later action stage emerge, the system retroactively analyzes event logs using an attention-based sequence-to-sequence model to identify early infections. Then, the infection detector is updated with information about the identified infections. We have evaluated ARIoTEDef against multi-stage attacks, such as the Mirai botnet. Results show that the infection detector's average F1 score increases from 0.31 to 0.87 after one evolution round. We have also conducted an extensive analysis of ARIoTEDef against adversarial evasion attacks. Our results show that ARIoTEDef is robust and benefits from multiple rounds of evolution.

CCS Concepts: • **Networks** → **Network security**; • **Security and privacy** → **Network security**; • **Computing methodologies** → **Machine learning**;

Additional Key Words and Phrases: IoT, NIDS, multi-step attack, infection identification, Seq2Seq, attention mechanism, adversarial evasion attack

This work was supported by Cisco Research, NSF grants 2112471 and 2229876, Purdue University, and Xidian University.
Authors' Contact Information: Mengdie Huang, School of Cyber Engineering, Xidian University, Xi'an, Shaanxi, China and Department of Computer Science, Purdue University, West Lafayette, Indiana, United States; e-mail: huan1932@purdue.edu; Hyunwoo Lee, Korea Institute of Energy Technology, Naju-si, South Korea; e-mail: hwlee@kentech.ac.kr; Ashish Kundu, Cisco Systems Inc, San Jose, California, United States; e-mail: ashkundu@cisco.com; Xiaofeng Chen, School of Cyber Engineering, Xidian University, Xi'an, Shaanxi, China; e-mail: xfchen@xidian.edu.cn; Anand Mudgerikar, Department of Computer Science, Purdue University, West Lafayette, Indiana, United States; e-mail: amudgeri@purdue.edu; Ninghui Li, Department of Computer Science, Purdue University, West Lafayette, Indiana, United States; e-mail: ninghui@purdue.edu; Elisa Bertino, Department of Computer Science, Purdue University, West Lafayette, Indiana, United States; e-mail: bertino@purdue.edu.

ACM 2577-6207/2023/06-ART15
https://doi.org/10.1145/3660646

ACM Reference Format:
Mengdie Huang, Hyunwoo Lee, Ashish Kundu, Xiaofeng Chen, Anand Mudgerikar, Ninghui Li, and Elisa Bertino. 2024. ARIoTEDef: Adversarially Robust IoT Early Defense System Based on Self-Evolution against Multi-Step Attacks. *ACM Trans. Internet Things* 5, 3, Article 15 (June 2024), 34 pages. https://doi.org/10.1145/3660646

1 INTRODUCTION

The **Internet of Things (IoT)** is a network of physical objects embedded with electronics, software, and network connectivity. It allows physical objects to collect and exchange data, and to be remotely sensed and controlled across the network infrastructure. IoT technologies combine the physical world with computer-based systems, creating countless opportunities for innovative applications. However, since IoT devices often have access to assets such as sensitive data, cyber-physical systems, and user credentials [10], they are valuable targets for attackers. IoT devices commonly suffer from inadequate hardening and infrequent patching. Developing security techniques for IoT devices is critical yet complex, as their resource constraints make defending themselves a challenge.

Attack campaigns aimed at compromising IoT devices often involve multiple steps to establish a foothold in the target system. A collection of these steps is called a *multi-step attack* or *multi-stage attack* [19, 38, 48]. For instance, in botnet campaigns such as Reaper [25] or Mozi [41], the attacker starts by scanning ports for any vulnerable entry points of the target device, then attempts to take it over by performing dictionary attacks [44] or zero-day attacks [3]. Once the attacker establishes a foothold, it can maintain persistence in the system to perform actions such as spreading malware to other devices, exfiltrating confidential data, or stealing credentials. We refer to the steps executed by the attacker to establish a foothold in the targeted system as the *early stages* and the subsequent steps as the *later stages*. Early detection of malicious behavior is critical for spotting potential attacks and preventing the spread of attack effects.

However, it is challenging to detect early-stage threats with both high precision and high recall. In other words, it is difficult to achieve low **false positives (FPs)** and **false negatives (FNs)** at the same time. The reason is that to gain a foothold in the target system, the adversary typically utilizes sophisticated techniques to evade the **network intrusion detection system (NIDS)**, such as performing stealthy attacks to camouflage their activities (e.g., distributed SSH brute-forcing [21]) or exploiting completely unknown device vulnerabilities (e.g. zero-day attacks) [25, 42, 45].

Some existing work makes use of **deep neural networks (DNNs)** that are better at learning complex abnormal patterns for network intrusion detection, such as HAM (Hierarchical Attention Model) [29] and SAAE-DNN [50], which use a stacked autoencoder and a gated recurrent unit to strengthen the traffic feature learning, respectively. However, to detect such attacks and reduce FNs, intrusion detectors that are optimized too strict on the threshold of anomaly class may also falsely identify legitimate changes and noise in the actual network environment as intrusions, thus generating a significant number of FPs and low F1 score [20]. Furthermore, as **machine learning (ML)** models are increasingly used as detectors, an attacker also attempts to evade detection by crafting *adversarial samples* which are typically created by adding *adversarial perturbations* to the original non-adversarial data (*clean samples*) with the intention of causing misclassification [14, 31, 51]. Existing IoT early defense efforts against early-stage attacks rarely consider this new type of threat.

In this article, to address those issues, we propose an ARIoTEDef (Adversarially Robust IoT Early Defense) system against a *cyber kill chain*. A cyber kill chain is a framework to break down a complex cyber attack into mutually non-exclusive stages or layers [52]. We focus on a cyber

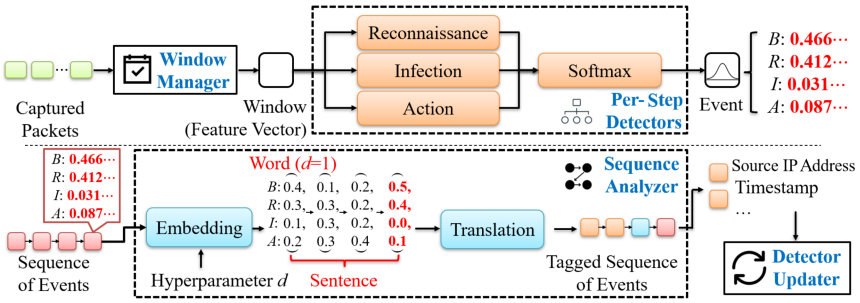

Fig. 1. Architecture of ARIoTEDef.

kill chain consisting of three stages where networking communication is involved: early-stage *reconnaissance*, early-stage *infection*, and later-stage *action*. The ARIoTEDef system is designed to identify more infection anomalies through the following process (Figure 1):

- —First, we use three independent binary classifiers as per-step detectors to detect malicious network traffic in three stages: reconnaissance, infection, and action.
- —Second, we generate several sequences of network events from the probabilities output by per-step detectors. Each event contains four probability features, indicating the probability that the traffic belongs to the four categories of reconnaissance, infection, action, and benign.
- —Hereafter, we use a sequence analyzer to further identify infection events by traversing the log of the events backward when detecting anomalies related to the action stage of the kill chain. The main purpose of this step is to detect early-stage infection events and normal events that were misclassified by the infection detector.
- —Finally, according to our designed self-evolution strategy, we retrain the infection detector with identified events to improve its classification performance on clean samples and robustness to adversarial evasion samples.

Essentially, when the per-step detector first encounters an unknown early-stage infection pattern, it may not achieve high recall. However, once the sequence analyzer recognizes the presence of an early-stage infection through the later-stage action pattern, the corresponding early-stage infection pattern is identified and the infection detector is prompted to learn it. As a result, ARIoTEDef will be capable of recalling such early-stage infection attacks later on. The main challenge is to correlate events in two different stages with potentially long intervals (e.g., correlating a UDP flood in the action stage with several dictionary attack packets present in the infection stage).

To address this problem, we adopt the **sequence-to-sequence (Seq2Seq)** translation model used in language translation tasks. We introduce a novel probability-based embedding method to encode past events into kill chain steps and design an attention-based infection identification algorithm to correlate the encoded events with long-term dependencies in different steps. Experimental results show that our designed Seq2Seq-based analysis algorithm helps to identify infection events leading to malicious action events. Moreover, our proposed self-evolution mechanism based on identified infection events also improves the robustness of the infection detector against evasion attacks.

As ARIoTEDef is an NIDS that deploys pre-trained lightweight ML models capable of rapidly detecting intrusion events, it is friendly to resource-constrained IoT scenarios, and only self-evolution involves a small amount of computation. In particular, we note that, depending on the specific hardware characteristics of the IoT devices, training involved in the evolution can also be

performed on an edge server and not necessarily directly on the IoT devices, as in the case of the device-edge split architecture for IoT host-based intrusion detection [37]. In addition, our systematic and automated method for early detection and self-evolution is beneficial to organizations that perform threat hunting [47]. According to a survey [8], many organizations value threat hunting because it is helpful for early detection and faster repair of vulnerabilities. However, 88% of the respondents say that their current systems for threat hunting are immature in terms of formal processes and automation. It shows the value of ARIoTEDef since it meets such requirements.

In summary, we make the following contributions:

- We propose ARIoTEDef, an adversarial robust NIDS framework, to detect infections in the early stage of multi-step attacks against IoT devices. We are the first IoT early defense work that considers the robustness of the detector against ML-based adversarial infection attacks.
- We design an event sequence analysis algorithm based on Seq2Seq structure and an attention mechanism to identify infection events that already existed by linking past events. It benefits the detection of infection events that were previously misclassified by the infection detector.
- We carry out comprehensive experiments to assess the effectiveness and robustness of ARIoTEDef. Results show that our approach is feasible and effective without loss of generality. The self-evolution strategy also improves the performance when applied to other NIDSes.
- We implement a proof-of-concept of the ARIoTEDef system and publicly release it.

An extended abstract of this article was published in ESORICS [28]. The main differences between this work and the conference version are as follows. First, we added two brand new sections, Sections 7 and 8, focusing on enhancing self-evolution strategies to reduce FNs and validating the robustness of our defense system against ML-based adversarial and non-adversarial evasion attacks by presenting comprehensive experimental results. Detailed computation, storage, time cost analysis, and robustness performance comparison with other robust training solutions have also been provided. Second, we describe the algorithm principle of adversarial evasion attacks in the new Section 3.2. In addition, we present an interpretation of our defense system's ability to be robust against multiple evasion attacks from an architectural design perspective in Section 4.2. Finally, we optimized the entire content, including emphasizing motivation and contributions related to robustness in Section 1, improving preliminaries in Section 3, refining threat models and clarifying system architecture in Section 4, and enhancing algorithm formalization in Section 5. We provide some of the abbreviations and notations used throughout this article in Table 1.

2 RELATED WORK

2.1 NIDS for IoT

Kalis [33] is a self-adapting, knowledge-driven NIDS system, which collects knowledge about the network's features autonomously and selects relevant detection techniques. Fu et al. [12] designed an NIDS that models the steps of a protocol with an automaton. Upon receiving a packet, the automaton corresponding to the packet protocol executes a transition. If there is any deviation in the execution of a protocol, the NIDS raises the alarm. DoT [39] is a federated self-learning anomaly detection system, which builds on device-type-specific communication profiles and raises an alarm upon detecting deviations concerning these profiles. To capture diverse device-type-specific communication profiles, it uses a federated learning approach for aggregating profiles from large numbers of clients. Unlike those systems, the focus of ARIoTEDef is to identify infection events from an attacker's actions, which is independent to the goals of those systems.

Table 1. Abbreviations and Notations

Abbreviation	Meaning	Notation	Meaning
IoT	Internet of Things	x	Clean Sample
ARIoTEDef	Adversarial Robust IoT Early Defense	ϵ	Perturbation Budget
NIDS	Network Intrusion Detection System	$\|\|\delta\|\| \leq \epsilon$	Adversarial Perturbation
ML	Machine Learning	$x^* = x + \delta$	Adversarial Sample
DL	Deep Learning	w	Window
Seq2Seq	Sequence-to-Sequence	e	Event (Class Probability Distribution for the Window)
LSTM	Long Short-Term Memory	$\mathbf{e}$	Event Sequence
TP	True Positive	d	Decimal Places
FN	False Negative	z	Word (Embedded Event)
TN	True Negative	$\mathbf{z}$	Word Sequence
FP	False Positive	y	Tag
FNR	False Negative Rate = $\frac{FN}{TP+FN}$	$\mathbf{y}$	Tag Sequence
FPR	False Positive Rate = $\frac{FP}{TN+FP}$	B	Benign
Recall	Recall = $\frac{TP}{TP+FN}$	R	Reconnaissance
Precision	Precision = $\frac{TP}{TP+FP}$	I	Infection
F1	F1 score = $2 * \frac{Precision*Recall}{Precision+Recall}$	A	Action

The Abbreviation is DNN, corresponding Meaning is Deep Neural Netwok.

2.2 Detection of Multi-Step Attacks

BotHunter by Gu et al. [16] detects malware infections by tracking communication flows between internal assets and external entities and applying dialog-based correlation. Haas and Fischer et al. [17] proposed a graph-based alert correlation (GAC). They use a graph-based clustering algorithm to cluster alarms based on their similarity. Then, each cluster is labeled considering the communications between attackers and victims within the cluster. Finally, the clusters are correlated based on the labels. Milajerdi et al. [34] proposed Holmes, which models the attacks with a kill chain. From audit logs, they generate a provenance graph, find adversarial activities based on pre-defined rules, and map the activities to the corresponding kill chain step. Han et al. [18] proposed Unicorn, which exploits provenance graphs to detect advanced persistent threats and uses the clustering approach to detect anomalies without prior knowledge of advanced persistent threat patterns. Our work differs from those approaches in three aspects:

— *Goals*: They use correlation algorithms to automatically detect multi-step attacks, whereas our approach aims to automatically identify infection vectors and update NIDS after seeing anomalies in the action step.
— *Logs*: Although correlation algorithms can be used to identify infection windows, these methods primarily rely on host events, such as process-related events. Applying such methods requires scaling IoT devices, which is something we want to avoid.
— *Techniques*: The preceding methods mainly rely on graphs to analyze the causality between events, whereas ARIoTEDef uses an attention mechanism-based Seq2Seq neural network to correlate event windows.

3 PRELIMINARIES

3.1 Cyber Kill Chain

A cyber kill chain refers to the multi-step chain of activities an attacker conducts to establish a persistent and undetected presence in a targeted cyber infrastructure [52]. Although the number and the name of steps vary, these kill chains commonly break down an attack into the following

five steps: reconnaissance, infection, lateral movement, obfuscation, and actions on targets [19, 48]. Since an NIDS can issue alerts on reconnaissance and infection steps with higher priority and can also detect network attack actions such as DDoS, in our work, we model a multi-step attack using a kill chain consisting of the following three steps:

— *Reconnaissance*: In the first stage, the attacker collects information about the target system to identify the target's weaknesses and potential attack opportunities. This may involve searching publicly available information, scanning networks and systems, collecting social engineering data of target employees, and so on.
— *Infection*: In this stage, the attacker selects the appropriate attack tool to weaponize the previous reconnaissance results, such as embedding malware into documents, links, or other carriers, then delivers them to the target system through e-mail attachments, malicious links, infectable external devices, or other means.
— *Action*: In the final stage, the attacker executes attacks based on their ultimate goals, such as data exfiltration, system disruption, and network spreading. The specific actions taken will depend on the attacker's motivations, such as directing bots to perform a DDoS attack with UDP flooding in botnets.

The reconnaissance and infection stages are early stages in a cyber attack, as they occur before the attacker has gained a foothold in the target system. In contrast, the action stage occurs after the attacker has established a foothold and is considered a later stage in the multi-step attack process.

3.2 Adversarial Evasion Attacks

Adversarial evasion attack refers to a type of attack in which attackers try to use adversarial ML techniques to evade detection by security systems. Concerning NIDS, these attacks are designed to make malicious activities or network anomalies appear benign or normal, thus bypassing ML-based network intrusion detection algorithms. Unlike conventional evasion attacks based on techniques such as packet segmentation, encryption, and obfuscation, in an adversarial evasion attack [13, 31, 35, 43], the attacker aims to craft the *adversarial sample* x^* by adding a slight *adversarial perturbation* δ to the original input x, also known as the *clean sample*. Commonly, techniques used for generating adversarial samples in adversarial ML can be divided into two types according to the goal of the adversary:

— *Untargeted attack*: The attacker aims to find a perturbation that maximizes the model's prediction error L without focusing on a particular target class. The optimization objective is shown in Equation (1), where f_θ denotes a DNN model with trainable parameters θ, ϵ denotes the l_p norm-measured maximum value of allowable perturbation δ, y_{true} is the ground truth label of x, and L is the loss function, which is set to cross-entropy loss by default.

$$\max_{x^*} L(f_\theta(x^*), y_{true}) = \max_{||\delta||_p \leq \epsilon} L(f_\theta(x+\delta), y_{true}) \tag{1}$$

— *Targeted attack*: The attacker aims to find a perturbation that leads to the desired misclassification to the target class y_{target}. The optimization objective is shown in Equation (2).

$$\min_{x^*} L(f_\theta(x^*), y_{target}) = \min_{||\delta||_p \leq \epsilon} L(f_\theta(x+\delta), y_{target}) \tag{2}$$

In other domains, such as multi-class image classification, adversarial samples can often be constructed from clean samples from any class. However, for the binary (benign/malicious)

Table 2. Black-Box Adversarial Evasion Attacks against Per-Step Detectors

Input	ϵ	IterNum	QueryNum	Reconnaissance Detector			Infection Detector			Action Detector		
				TP	FN	Recall (%)	TP	FN	Recall (%)	TP	FN	Recall (%)
Clean	–	–	–	3,064	13	99.58	126	192	39.62%	112	11	91.06
Boundary	1.0	500,000	100,000	3,064	13	99.58	126	192	39.62%	112	11	91.06
HopSkipJump	–	500,000	100,000	3,064	13	99.58	126	192	39.62%	112	11	91.06

classification-based NIDS, *adversarial attacks* usually refer to *adversarial evasion attacks*, and adversarial samples used in adversary attacks are usually only constructed from malicious samples, to mislead target detection into predicting them as benign. Thus, the *adversarial evasion sample* $x^* = x + \delta$ can be generated according to the optimization objective of the untargeted attack with the ground truth label y_{true} setting to malicious, or be constructed according to the optimization objective of the targeted attack with the target label y_{target} setting to benign. In this work, we *only* consider adversarial evasion samples and abbreviate them as adversarial samples. For example, we refer to adversarial evasion samples constructed from clean samples originally belonging to the infection category as *adversarial infection samples*.

According to the level of knowledge an adversary has about the target system, the threats faced by the NIDS from adversarial evasion attacks are mainly divided into the following two types:

— *White-box adversarial attack*: The adversary has complete knowledge and access to the target system, including its architecture, parameters, and internal workings. With this information, the attacker can craft highly optimized adversarial samples.
— *Black-box adversarial attack*: The attacker has no knowledge about the target system. Typically, the attacker only has access to the input and output of the model and relies on techniques such as input manipulation, query optimization, and exploration to understand the target model and craft adversarial samples based on the local surrogate model.

In our work, we focus only on white-box adversarial attacks for two reasons. First, a model's resilience to white-box attacks provides a more comprehensive evaluation of its robustness. If a model can withstand white-box attacks, it is more likely to be robust against various other types of attacks as well. Second, our preliminary experiment results show that the per-step detectors, when trained according to a standard approach without self-evolution, are robust against black-box adversarial attacks, such as Boundary [4] and HopSkipJump [6]. Table 2 shows that such black-box attacks are unable to evade the per-step detectors even with large perturbation budgets, iteration rounds, and query times. However, they are not robust against white-box adversarial attacks.

3.3 Long Short-Term Memory Based Seq2Seq Model with the Attention Mechanism

A Seq2Seq model consisting of two main components, an encoder and a decoder, is a **deep learning (DL)** model commonly used for tasks involving sequence data, such as machine translation, text summarization, and speech recognition [49]. In our work, we build both the encoder and decoder structures of the Seq2Seq model on typically **Long Short-Term Memory (LSTM)** units [15] and refer to it as an LSTM-based Seq2Seq model (Figure 2). The two main components of the LSTM-based Seq2Seq model are as follows:

— *Encoder*: In the Seq2Seq model, the unified goal of the encoder is to capture essential information from the entire input (word) sequence and generate a *representation*. In the LSTM-based Seq2Seq model, h_0 is the initial hidden state of the encoder. Each element of the input

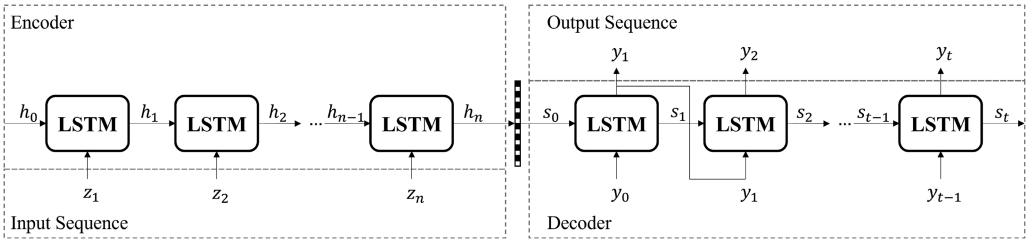

Fig. 2. LSTM-based Seq2Seq architecture.

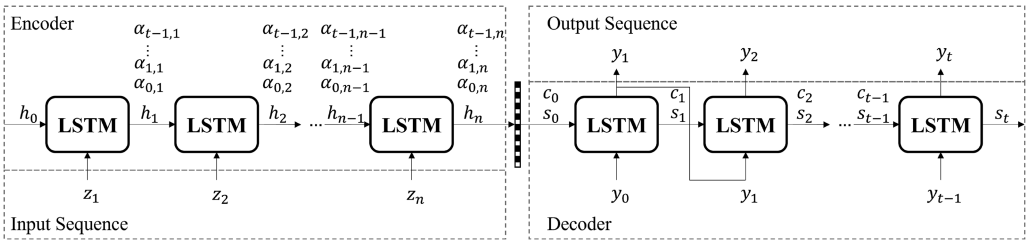

Fig. 3. LSTM-based Seq2Seq architecture with attention.

sequence $\{z_i\}_{i=1}^{n}$ is fed into the LSTM cell one by one to generate encoder hidden states $\{h_i\}_{i=1}^{n}$, and the final hidden state h_n of the LSTM is used as the target representation.

— *Decoder*: In the LSTM-based Seq2Seq model, the decoder uses the representation h_n generated by the encoder as the initial decoder hidden state s_0, and generates the output sequence $\{y_i\}_{i=1}^{t}$ step by step, with each step representing a *timestep*. $\{s_i\}_{i=0}^{t}$ denotes the decoder's hidden states. During training, the decoder is fed with the ground truth output sequence up to the current timestep to predict the next element. However, during inference or testing, the decoder uses its own predictions as input for the next timestep. The decoder generates an element at a timestep until a pre-defined maximum length is reached.

However, Cho et al. [7] have shown that the performance of the model degrades as the length n of the input sequence increases, which is called the *bottleneck problem*. The reason for this problem is that information loss occurs in the representation due to its fixed size. Specifically, it often fails to capture the interdependencies between elements that are far apart in a sequence.

To enhance the performance of the Seq2Seq model, an *attention mechanism* [2, 30] is often incorporated between the encoder and decoder (Figure 3). The motivation for the use of the attention mechanism is to make the decoder focus on different parts of the input sequence according to relevance while generating each element in the output sequence. To this end, the attention mechanism requires the decoder not only to refer to the current hidden state s of the decoder but also to a *context vector* c at each timestep, which is a weighted average of all hidden states $\{h_i\}_{i=1}^{n}$ of the encoder. Then, the context vector c and the hidden state vector s of the decoder are concatenated and given as input to the LSTM cell. With the context vector, the decoder will focus more on certain hidden states of the encoder related to the current state of the decoder.

For example, when the next element to be returned by the decoder is y_k, $k \in \{1, 2, \ldots, t\}$, each hidden state of the encoder $h_i, i \in \{1, 2, \ldots, n\}$, is associated with an *attention weight* $\alpha_{k-1,i}$, which reflects the relevance of the encoder hidden state h_i to the current decoder state s_{k-1}. Attention weight is determined by the alignment score that quantifies the amount of attention. The most widely used scoring function is the dot product, by which the alignment score is obtained by multiplying the hidden states of the encoder with the state of the decoder. Every time the decoder

predicts an element in the output sequence, a set of attention weights is updated according to the latest hidden state of the decoder to obtain a new context vector. Therefore, when the length of the output sequence is t, t context vectors will be calculated according to Equation (3).

$$c_k = \sum_{i=1}^{n} \alpha_{k-1,i} h_i, k = \{1, 2, \ldots, t\} \tag{3}$$

4 ARCHITECTURE OF ARIOTEDEF

In this section, we introduce the design principles and the framework of ARIoTEDef. We first describe the threat model faced by ARIoTEDef and its main properties, then describe the architectural components of ARIoTEDef.

4.1 Design Principles

4.1.1 Threat Model. ARIoTEDef is designed to analyze network packets exchanged between the protected network and the Internet. We assume that ARIoTEDef is not compromised, and therefore it does not manipulate the exchanged packets. We consider two main evasion attack models: the black-box non-adversarial attack model and the white-box adversarial attack model:

— *Black-box non-adversarial attack model*: We assume that attacks are launched from the Internet by using remote network access. In other words, we assume that IoT devices are not compromised when they are initially deployed in the network, and that the attacker does not have physical access to the IoT devices or direct access to the wireless network to which these devices are connected. In this work, we refer to attack samples made based on this kind of threat model as black-box non-adversarial attack samples or clean evasion samples.
— *White-box adversarial attack model*: We assume that the attacker has complete knowledge of the architecture, algorithms, and parameters of the target model. These attacks exploit vulnerabilities in DNN-based classifiers to construct adversarial evasion samples from the clean evasion samples. In this work, we refer to attack samples made based on this kind of threat model as white-box adversarial attack samples.

4.1.2 Main Properties. To be strong against threats, ARIoTEDef is designed to adhere to the following properties:

— *Network based*: It works with network packets and does not require any change to IoT devices, avoiding any computational burden on IoT devices and enabling instant deployment.
— *Anomaly based*: It is capable of detecting unknown patterns and is also appropriate for the simple communication behaviors of IoT devices.
— *Cyber kill chain based*: It understands multi-step attacks based on the cyber kill chain and deploys classifiers specialized for these steps.
— *Infection identifying*: When a later-stage action event of a multi-step attack is detected, it analyzes past events to identify infection events.
— *Self-evolution*: The infection detector is retrained with identified infection events.
— *Adversarial robust*: The evolved infection detector is robust to adversarial evasion attacks using adversarial samples.

4.2 System Architecture

ARIoTEDef consists of four main components (see Figure 1): Window Manager, Per-Step Detectors, Sequence Analyzer, and Detector Updater. The detailed workflow of each module is as follows.

4.2.1 Window Manager (Packets → Window). ARIoTEDef works on a flow-based window, where a network *flow* is defined as a 5-tuple consisting of the used protocol, source/destination IP address, and source/destination port. The window manager is responsible for collecting packets for each flow and sliding the window according to two parameters: window output period and window length. At each window output period, the window manager outputs a *window* vector. The elements of this window vector correspond to the 84 flow features considered by the network traffic analyzer CICFlowMeter [27]. The value of each feature in a window vector is calculated from all packets within the window length.

For example, suppose that the window output period is 2 and the window length is 5. When outputting a window at time $t = 5$, the window vector consists of flow feature values computed from all packets captured between $t = 0$ and $t = 5$. Since the window output period is 2, the next window vector will be output at $t = 7$, where the feature values are computed from all packets captured between $t = 2$ and $t = 7$.

4.2.2 Per-Step Detectors (Window → Event). The main purpose of per-step detectors is to map an arbitrary window to one or more steps in a cyber kill chain. In other words, the packets within a window may only correspond to one step in a multi-step attack, or may correspond to multiple steps at the same time. In this work, we model a multi-step attack as three steps in a cyber kill chain: reconnaissance, infection, and action. To this end, we design three binary classifiers, called *reconnaissance* detector, *infection* detector, and *action* detector, respectively, as per-step detectors in ARIoTEDef to detect whether there is a corresponding anomaly in an arbitrary window. Once a window vector is given to ARIoTEDef, each per-step detector takes it as input and determines whether it contains any anomalous patterns for the corresponding step. If so, ARIoTEDef marks the input window vector with the name of the corresponding step.

For example, we call a given window a reconnaissance window if the reconnaissance detector detects an anomaly from the window vector. If the infection detector also detects an anomaly from this window vector, we call it both a reconnaissance window and an infection window. This process provides a precedence relation between windows according to the kill chain steps.

The rationale behind this modular design is as follows. First, the detectors specialized in different steps are more effective, as they have higher accuracy than a single monolithic detector that detects all events of the steps altogether [46]. Second, ARIoTEDef mainly focuses on preventing infections by identifying infection patterns and improving the infection classifier. The modular approach allows for an efficient evolution of the system as whenever an anomaly at a step is detected, ARIoTEDef can upgrade the infection detector individually without affecting the other detectors. Third, such modular design helps with explainability for attacks. With the detection results, our modular architecture can provide information about the attack step (e.g., Reconnaissance, Infection, or Action) that the adversary is performing, which is helpful to understand the attack in detail and possibly contain/block the attack. Finally, in terms of security, ARIoTEDef is more resilient to attempts aimed at evading detection as the attack needs to avoid detection by all detectors.

Since there may be FPs or FNs in the output of per-step detectors, we let each per-step detector not only output the binary classification result (benign/malicious) but also output the confidence score—that is, the score level of the detection result, to prompt FPs and FNs. In other words, each per-step detector outputs a score for the malicious class and a score for the benign class. ARIoTEDef takes the product of the benign scores output by the three per-step detectors as the global confidence score of the benign category, then applies the softmax function to normalize the confidence scores of the four categories of reconnaissance, infection, action, and benign to produce a probability distribution. We call an output of per-step detectors module an *event* that contains a window, three labels (indicating whether the window belongs to each per step, respectively), and four probabilities (normalized confidence scores for the four categories).

4.2.3 Sequence Analyzer (Sequence of Events → Identified Infection Events). To correct FPs and FNs of the infection detector, we design an event sequence analyzer to identify infection events that lead to detected action events. Our proposed infection identification algorithm is based on a Seq2Seq translation model, which is implemented based on the LSTM cells and an attention mechanism. This algorithm takes a sequence of past events with a certain length as input. Each event is represented as a four-dimensional probability vector consisting of four confidence scores assigned by per-step detectors. Then, the identification algorithm analyzes the input event sequence according to the entire context and predicts a unique step in the kill chain for each event, resulting in an output step sequence with the same length as the input. Finally, ARIoTEDef selects the steps predicted to be infection from the output step sequence and returns the corresponding infection events.

4.2.4 Detector Updater (Identified Infection Events → Updated Infection Detector). The detector updater is responsible for updating the infection detector using the infection events tagged by the sequence analyzer. This module first assigns infection labels to infection windows in the identified infection events. Then, according to the adopted self-evolution strategy, it uses these newly prepared infection window samples and original training samples for the infection detector as the training set to retrain the binary classifier of the infection detector. In addition, the self-evolution of infection detectors also brings sustainable growth in adversarial robustness to the infection detector. On the one hand, benefiting from the introduction of the attention mechanism, the sequence analyzer can combine the long context information in the event sequence to focus on the non-sensitive features of the input sample, thereby identifying more complex malicious traffic patterns that are difficult to be identified by the infection detector, such as adversarial evasion samples. As the infection detector is updated with relabeled adversarial evasion samples, the adversarial robustness of the model is also enhanced in the process of forcing the detector to learn small changes in the input space. On the other hand, since the adversarial samples used by the evolution of the infection detector come from the attacker rather than actively constructed by the defender, it avoids the high time overhead required for adversarial sample generation and enhances the adaptability to constantly updated adversarial attack methods.

5 FORMALIZATION OF ARIOTEDEF

In this section, we describe in detail the algorithms of ARIoTEDef. We first formally define the problem to be solved and then present the solutions we propose through a probability-based embedding algorithm and an attention-based translation algorithm. Finally, we discuss the self-evolution strategy for the infection detector.

5.1 Problem Definitions

In the following definitions, we use the notation $a.b$ to indicate an attribute b of a.

Definition 1 (Tag). Tag refers to the label assigned to the event by the sequence analyzer. We denote the collection of tags that can label events by the set $\mathcal{L} = \{B, R, I, A\}$, where B, R, I, A denote Benign, Reconnaissance, Infection, and Action, respectively.

Definition 2 (Event). An event $e = (w, l, p, t)$ in the event set $\mathcal{E}$ is a tuple with four attributes:

- $e.w$ is a vector representing the window in the event e.
- $e.l = (r, i, a)$ is a tuple with three attributes indicating the label of the window $e.w$. Each attribute indicates the result of the window $e.w$ being predicted by the per-step detector and the attribute values $e.l.r, e.l.i, e.l.a \in \{0, 1\}$. Taking $e.l.r$ as an example, when $e.l.r = 0$,

it means that $e.w$ is predicted as benign by the reconnaissance detector; when $e.l.r = 1$, it means that $e.w$ is predicted as malicious by the reconnaissance detector.

— $e.p = (b, r, i, a)$ is a tuple with four attributes representing the confidence scores of the windows $e.w$ on the four categories. The attribute values $e.p.b, e.p.r, e.p.i, e.p.a$ denote the probability given by the per-step detector that the window $e.w$ belongs to the benign, reconnaissance, infection, and action classes, respectively. Attribute values $e.p.b, e.p.r, e.p.i, e.p.a \in [0, 1]$ and $e.p.b + e.p.r + e.p.i + e.p.a = 1$.

— $e.t$ represents the tag assigned to the event e by the infection identification algorithm, where $e.t \in \mathcal{L} = \{B, R, I, A\}$.

Our goal is to identify infection events in the early stages of the cyber kill chain by backtracking past events from anomalies in known action steps. To this end, we model the backtracking process as an event tagging problem, described as follows.

Problem 1 (Event Tagging Problem). *Let $\mathbf{e} = \{e_1, e_2, \ldots, e_n\} \in \mathcal{E}^*$ be an input event sequence where $\mathcal{E}^*$ is a set of all event sequences over the event set $\mathcal{E}$, and let $\mathbf{y} = \{y_1, y_2, \ldots, y_n\} \in \mathcal{L}^*$ be an output tag sequence where $\mathcal{L}^*$ is a set of all tag sequences over the tag set $\mathcal{L}$. Our goal is to design a function $g : \mathcal{E}^* \rightarrow \mathcal{L}^*$ that takes an input sequence $\mathbf{e}$ and outputs $\mathbf{y}$.*

The main challenge for solving this problem is the long-term dependencies between events. Specifically, infection events always precede action events, but the time interval between these two events can be long. For example, in the Mirai botnet attacks [1], the device, to be used as a bot, is first infected by the attacker using a dictionary attack. Then, the attacker can launch a UDP flood attack toward the victim a long time after the infection event. To this end, ARIoTEDef should be able to correlate distant events. For example, after detecting a UDP flood pattern in the action step, a dictionary attack pattern in the infection step can be identified by backtracking past events.

To address this issue, we use language translation techniques in natural language processing, since even words that appear to be far apart can have significant relationships in natural language. Many techniques have been proposed to capture such dependency [5]. Our idea is that if we can model events as words and event sequences as sentences in the language, then we can address our problem by using language translation techniques. To this end, we split the preceding Problem 1 into two problems to solve one by one:

(1) Model the event as a representation vector that can reflect the correlation information between events, just like the representation vector of words in natural language processing.
(2) Translate the event sequence with a long-distance relationship between events into an equal-length tag sequence—that is, predict a category label for each event in the input sequence.

We formalize the preceding two subproblems as follows.

Subproblem 1 (Embedding Problem). *Let $\mathbf{e} = \{e_1, e_2, \ldots, e_n\} \in \mathcal{E}^*$ be an input event sequence of length n, and let $\mathbf{z} = \{z_1, z_2, \ldots, z_n\} \in \mathcal{Z}^*$ be an input word sequence, where $\mathcal{Z}^*$ is a set of all word sequences over an input word set $\mathcal{Z}$. Our goal is to define an input word set $\mathcal{Z}$ and an embedding function $e : \mathcal{E}^* \rightarrow \mathcal{Z}^*$, which takes $\mathbf{e}$ as input and outputs $\mathbf{z}$.*

Subproblem 2 (Translation Problem). *Let $\mathbf{z} = \{z_1, z_2, \ldots, z_n\} \in \mathcal{Z}^*$ be an input word sequence, and let $\mathbf{y} = \{y_1, y_2, \ldots, y_n\} \in \mathcal{L}^*$ be an output tag sequence. Our goal is to design a translation function $t : \mathcal{Z}^* \rightarrow \mathcal{L}^*$ that takes $\mathbf{z}$ as an input and outputs $\mathbf{y}$, thus labeling all of the embedded windows represented as words in the word sequence $\mathbf{z}$.*

5.2 Solution

Our solution to the preceding problems consists of the following three steps:

- —*Probability distribution assignment*: Each event is assigned a probability distribution $e.p$ indicating the confidence that the window $e.w$ belongs to each class.
- —*Probability-based embedding*: Encodes the probability distribution $e.p$ of an event e into a word in the language.
- —*Attention-based translation*: Translates a sequence of embedding events represented by words into corresponding tags, with each word corresponding to only one tag.

5.2.1 Probability Distribution Assignment. The per-step detectors in ARIoTEDef are responsible for assigning a probability distribution to each event. Recall that the per-step detectors detect reconnaissance, infection, and action patterns in a given window. Each detector is a binary classifier for benign and malicious events, trained with window samples from the corresponding step. For instance, the classifier of the infection detector is trained according to a standard approach on an infection window dataset, which consists of normal window samples and abnormal window samples from telnet dictionary attacks, Log4j attacks, or other attacks aimed at infecting IoT devices. The classifier structure of each per-step detector is customizable. However, note that the performance of the classification algorithm of the per-step detector will affect the performance of ARIoTEDef, because the infection identification algorithm in the sequence analysis module is run over the output of the per-step detectors. Our results show that LSTM-based per-step detectors perform best for classification (see Section 6.4).

Each per-step detector assesses the probability of the window belonging to its corresponding step. For instance, the reconnaissance detector might assign a 0.68 probability to the window being reconnaissance, whereas the infection detector could assign a 0.53 probability to the window indicating infection. We convert the probabilities into one probability distribution using the softmax function and finally output the distribution as an event. For example, an event might carry a probability distribution of $(0.46, 0.31, 0.11, 0.12)$, signifying that the probabilities of the corresponding window being benign, reconnaissance, infection, and action are 0.46, 0.31, 0.11, and 0.12, respectively.

5.2.2 Probability-Based Embedding. To address Subproblem 1, we design a novel probability-based embedding algorithm, as shown in Algorithm 1.

We represent each event e in the input event sequence as a word z in the input word sequence, which is a vector of four probabilities summing to 1. One issue is that the preceding input word set is infinite, which would be inappropriate for a language translation model based on finite input word sets. Thus, we change the input set to be finite. To this end, we introduce a hyper-parameter d, which is the number of decimal places to round off the probabilities. Given $d \in \mathbb{N}$, let $\mathcal{P}$ be $\{p | p = \text{round}(q, d), 0 \leq q \leq 1\}$, where $\text{round}(a, b)$ is a function that rounds off a to b of decimal places. With d, the input set is changed to $\mathcal{Z} = \{(b, r, i, a) | b, r, i, a \in \mathcal{P} \text{ and } b + r + i + a = 1\}$. However, rounding off does not guarantee that the sum of the rounded probabilities is always 1. To avoid the case that the sum is not 1, we first sort the probabilities by the decimal part to be rounded off. Then, we round up the first one or two probabilities or down the last one or two to ensure that the sum of the resulting probabilities is 1. For example, $(0.466\ldots, 0.412\ldots, 0.031\ldots, 0.087\ldots)$ became $(0.5, 0.4, 0.0, 0.1)$ when $d = 1$ (see the boldface numbers in Figure 1). Note that d determines the number of input words in the word set.

5.2.3 Attention-Based Translation. To address Subproblem 2, we apply the attention mechanism to the LSTM-based sequence analyzer to label the events in the input event sequence one by one. The flow of infection event identification is as follows (see Algorithm 2):

ALGORITHM 1: Probability-Based Embedding e

Input: Sequence of events $\mathbf{s} = (e_1, \ldots, e_n)$ and d
Output: Sequence of words (Sentence) $\mathbf{z} = (z_1, \ldots, z_n)$
Initialize: $\mathbf{z}[:] = []$

for $k = 1, 2, \ldots, n$ **do**
  sort $e_k.p.b, e_k.p.r, e_k.p.i, e_k.p.a$ by the decimal part to be rounded off in descending order
  if more than two decimal parts are larger than 5 **then**
    Round down the last one or two probabilities to ensure $sum = 1$
    Round off the rest of the probabilities
  else if more than two decimal parts are smaller than 5 **then**
    Round up the first one or two probabilities to ensure $sum = 1$
    Round off the rest of the probabilities
  else
    Round off the probabilities
  end if
  $\triangleright$ $\mathtt{r}(a, b)$: the result of rounding up/down/off a to b of decimal places
  $z_k = (\mathtt{r}(e_k.p.b, d), \mathtt{r}(e_k.p.r, d), \mathtt{r}(e_k.p.i, d), \mathtt{r}(e_k.p.a, d))$
  Add a word z_k to Sequence $\mathbf{z}$
end for

ALGORITHM 2: Attention-Based Translation t

Input: Sequence $s_i = (e_1, \ldots, e_n)$, Decimal Place d
Output: Sequence s_o
Initialize: $s_o[:] = 0$

$s_e = \mathtt{ProbabilityBasedEmbedding}(s_i, d)$
$r_{lstm} = \mathtt{LSTM}(s_e)$
$r_{attention} = \mathtt{Attention}(r_{lstm})$
$r_{ff} = \mathtt{Feedforward}(r_{attention})$
$r_{sm} = \mathtt{Softmax}(r_{ff})$
$\triangleright$ $r_{sm} = (p_{b_1}, p_{r_1}, p_{i_1}, p_{a_1}), \ldots, (p_{b_n}, p_{r_n}, p_{i_n}, p_{a_n})$
for $k = 1, 2, \ldots, n$ **do**
  $e_k.t = \mathtt{argmax}((p_{b_k}, p_{r_k}, p_{i_k}, p_{a_k}))$
  Add e_k to sequence s_o
end for

— *Input*: The input sequence consists of a series of events, each of which has a probability distribution assigned by the per-step detectors.
— *Embedding*: Events in the input sequence are converted into words one by one, and each word consists of four probabilities.
— *LSTM-based encoder*: After embedding the event sequence as a word sequence, the word sequence is input to the LSTM-based encoder, which outputs a hidden representation that encodes the information of the entire input sequence.
— *Attention*: After generating the final hidden state of the encoder, all hidden states of the LSTM-based encoder and the current hidden state of the decoder are fed into an attention layer. The attention layer first calculates a set of attention weights, reflecting the relevance of each hidden state of the encoder to the next output prediction of the decoder. Then, all hidden states of the encoder are weighted and averaged according to the attention weights, and the result is output as a context vector.

— *LSTM-based decoder*: The LSTM layer of the decoder takes the current decoder hidden state, the context vector, and the previous decode word as inputs, and outputs a decoded word along with the updated decoder hidden state. When the decoder prepares to output a translated word at each timestep, it needs to recompute its associated context vector until the output sequence reaches the upper limit.
— *Feedforward and softmax*: We add a feedforward layer and a softmax layer. They output four probabilities for each input word. Each probability represents the degree to which an input word is translated into an output word.
— *Output*: Finally, each input word representing an embedded event in the input sequence is translated into the output word representing the tag category with the highest probability.

With the attention mechanism, ARIoTEDef analyzes a given sequence in the context between events. As words of the same form may have different meanings in the context of the sentence, events with the same distribution may also belong to different steps in the context of the sequence. For example, some events with the same distribution may belong to Benign or Infection, depending on whether an action event is in the sequence or not. In the attention mechanism, the attention weights are evaluated concerning different steps and positions in sequences. Thus, it helps to distinguish the differences between the events that have the same distribution but belong to different steps and identify infection events related to an action event in the sequence.

5.3 Self-Evolution Strategies

The self-evolution strategies are used to guide the detector updater to retrain the binary classifier of the infection detector with the events identified by the sequence analyzer.

Definition 3 (Evolution Set). Let $\mathcal{A}$ be a set of events tagged as infection by the infection detector. Let $\mathcal{B}$ be a set of events tagged as infection by both the infection detector and the LSTM-based translation model with the attention mechanism. Let C be a set of events tagged as infection by the infection detector but tagged as benign by the LSTM-based translation model with the attention mechanism. In general, $\mathcal{B}$ and C are two subsets of $\mathcal{A}$ that satisfy $\mathcal{A} = \mathcal{B} \bigcup C$. We label all events in set $\mathcal{B}$ as infection and all events in set C as benign.

We consider three different self-evolution strategies. Let $\mathcal{D}_{inf-tra}$ be the original training set for training the infection detector:

— *Strategy 1*: The binary classifier of the infection detector is retrained over a retraining set, consisting of windows in all events in $\mathcal{B}$, windows in all events in C, and $\mathcal{D}_{inf-tra}$. For example, suppose that there are five events $e_1, e_2, e_3, e_4, e_5 \in \mathcal{A}$ (i.e., $e_i.l.i = 1$ for $i = \{1, 2, 3, 4, 5\}$) and the attention-based infection identification algorithm provides the information that three events $e_1, e_2, e_4 \in \mathcal{B} \subseteq \mathcal{A}$ are infection events (i.e., $e_1.t = e_2.t = e_4.t = I$). Then, ARIoTEDef updates the infection detector with $e_1.w, e_2.w, e_4.w$ labeled as Infection, $e_3.w, e_5.w$ labeled as Benign, and $\mathcal{D}_{inf-tra}$.
— *Strategy 2*: This strategy is similar to Strategy 1 except that it only uses windows in all events in $\mathcal{B}$ and $\mathcal{D}_{inf-tra}$, which adds the information about the **true positives (TPs)** to the updated model. In the preceding example, $e_1.w, e_2.w, e_4.w$ labeled as Infection and $\mathcal{D}_{inf-tra}$ are used to retrain the infection detector.
— *Strategy 3*: This strategy is similar to Strategy 1 except that it only uses windows in all events in C and $\mathcal{D}_{inf-tra}$, which aims to reduce the FPs of the updated model. In the preceding example, $e_3.w, e_5.w$ labeled as Benign and $\mathcal{D}_{inf-tra}$ are used to retrain the infection detector.

The main purpose of this definition is not only to strengthen the knowledge of infection detectors on known infection patterns but also to identify benign events that are misclassified as infection by the infection detector so that they can be used to reduce the FPs of the evolved infection detector.

6 EXPERIMENTS IN THE REGULAR SETTING

This section presents the experimental analysis of ARIoTEDef on the clean dataset without DL-based adversarial evasion samples. We implement a proof-of-concept prototype and build a testbed for evaluation using a dataset related to the Mirai botnet [1] and the Log4j attack [36].

6.1 Experimental Setup

6.1.1 Implementation. We use the pcapy library [9] to capture packets and the Keras library [23] to implement neural networks and other ML-related functions.

6.1.2 Model. The LSTM layer consists of 100 units for our attention-based neural network with 0.5 for the dropout rate and 0.2 for the recurrent dropout rate. The subsequent attention layer uses a dot product as a scoring function. Finally, the feedforward layer consists of 64 units. We use sparse categorical cross entropy as the loss function.

6.1.3 Dataset. We generate datasets considering the following two scenarios:

— *Mirai botnet campaign* [1]: This includes the telnet dictionary attack as an infection activity. We use a publicly available IoT intrusion dataset from academia [22]. It contains captured packets from the real world and consists of diverse types of packets including benign packets, port scanning packets, telnet dictionary attack packets, and flooding packets, as separate files. We extract packets from the files and combine them into one dataset. We label port scanning packets to reconnaissance, the telnet dictionary attack packets to infection, and the flooding packets to action. We add benign telnet login packets to degrade the performance of the infection detector, because benign telnet login packets are similar to the telnet dictionary attack packets. By adding the benign telnet login packets, the FPs of the infection detector may increase. The detail of making a different Mirai botnet dataset based on the work of Kang et al. [22] and implemented script is described in Section 6.2.
— *Log4j attack* [36]: This includes the Log4j attack as an adversary's infection activity. We build our testbed based on mininet [26], run multi-step attacks, and capture the packets. The resulting dataset includes the port scanning packets as reconnaissance, the Log4j attack packets as infection, and the flooding packets as action. The dataset also contains benign HTTP POST packets from which the Log4Shell attack packets are difficult to discern.

6.1.4 Testbed. We perform our experiments on one machine with an i7-4700 CPU @ 3.60-GHz eight-core processors and 16 GB of RAM. To evaluate the performance of ARIoTEDef with the practical scenarios, we replay the packets from the preceding dataset with Tcpreplay [24]. The generated packets are captured by ARIoTEDef.

6.1.5 Experiments. We measure the performance for the following three cases for a given test set. We report averaged results of 30 trials per scenario:

— *Baseline*: We see how many infection events can be correctly detected by the infection detector learned only with the training set.
— *Attention*: We evaluate how well our attention-based infection identification algorithm works over a sequence of events.

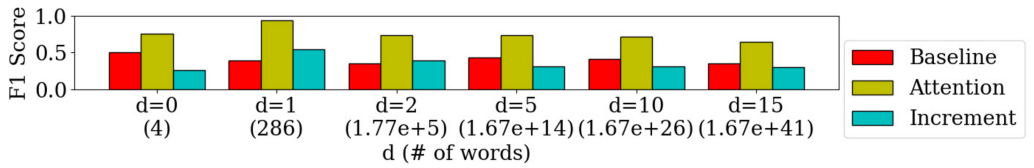

Fig. 4. Impact of the probability-based embedding.

— *Update*: We measure the performance of the infection detector evolved with the identified infection events on a different test set.

6.2 Dataset Generation

In our experiment, we use the dataset from Kang et al. [22] as discussed in Section 6.1.3. It consists of several files that capture packets related to the Mirai botnet. In detail, it includes the ARP spoofing packets, host discovery packets, or other flooding packets. Among them, we use the following packets in our experiments:

— *Benign*: These packets are normal packets exchanged between benign entities.
— *Port scanning*: These packets are simple SYN packets to scan open ports at a targeted device. These packets are labeled as reconnaissance.
— *Brute force*: These packets are used to perform dictionary attacks with pre-defined credentials to infiltrate a target device. We label these packets as infections.
— *Flooding*: These packets are SYN/ACK/HTTP/UDP flooding packets to cause a DoS condition on a victim. These packets are tagged as action.

Due to the limited number of datasets, we manipulate the existing dataset to create new diverse pcap files, which contain different attack patterns (e.g., types, order, and timing). For example, we want to generate a dataset with a specified number of infection packets at a certain time and several UDP flooding packets for a particular time. To this end, we implement a data manipulation script, which works as follows:

(1) A new scenario file is created. The starting time of the scenario is 0.
(2) A list of files that contain interesting packets is specified with the starting time and the duration. In detail, the list consists of several pairs (<*file name*> <*starting time*> <*duration*>), which means that the packets are randomly extracted from <*file name*> and inserted into the new scenario file at time <*starting time*> for <*duration*>. For example, $bruteforce.pcap$, 10, 2 means that the packets from $bruteforce.pcap$ are inserted into the new scenario at time 10 for 2 seconds.
(3) All packets are extracted from the files in the list and put into the new scenario file appropriately. We allow overlaps between different packets.
(4) Finally, the IP addresses of the packets are modified to the loopback addresses.

6.3 Impact of Probability-Based Embedding

We assess the impact of our probability-based embedding on the performance of our attention-based translation by varying the value of the hyper-parameter d (Figure 4), which is the number of decimal places to round off the probabilities.

Overall, the F1 score increases from the Baseline when Attention is used. Furthermore, we see that Attention works best with $d = 1$. Note that the larger the value of d is, the higher the number of elements in the set. For $d > 1$, the number of elements is higher than 10^5, which we believe is too

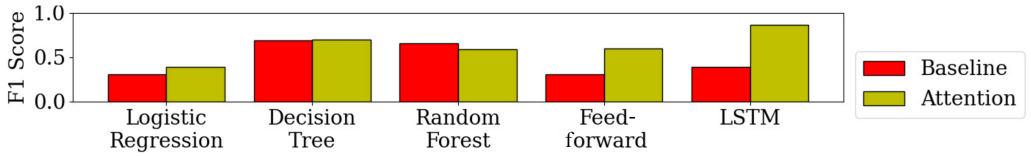

Fig. 5. Impact of classifiers of per-step detectors.

large for mapping to only four variables in the output word set. The worst increment is at $d = 0$, where the one-hot encoding is used. The result shows that the one-hot encoding is ineffective for Attention, as it cannot capture dependency between words. Hereafter, we *fix* $d = 1$ in the other experiments.

6.4 Impact of Classifiers of Per-Step Detectors

As Attention relies on probabilities assigned by per-step detectors (see Figure 1), we carry out some experiments to understand the impact of different types of classifiers for the per-step detectors on the performance of Attention. As classifiers, we use logistic regression, decision tree, random forest, the feedforward neural network, and LSTM. Note that LSTM here is the classifier architecture of the infection detector, not the encoder layer before the attention layer in Attention. We evaluate each classifier with and without the Attention and calculate the F1 score.

We find that the neural networks are compatible with Attention (Figure 5). The increments of the F1 score for both neural network algorithms are 0.29 (Feedforward) and 0.48 (LSTM), respectively, whereas for other algorithms it is less than 0.08. Notably, LSTM is the one classifier that works best with Attention. Compared with other algorithms, LSTM is the only algorithm that considers the context of windows, which explains the result. After breaking down the result of the neural networks, we find that Attention contributes to increasing precision while maintaining recall. This result shows that the attention mechanism assigns higher weights to features that are useful for finding FP results in the detectors.

The reason non-neural network algorithms show worse performance is because of their assumption. Logistic regression shows poor performance due to its linear boundary assumption. The decision tree shows high precision with low recall, which means that it is over-fitted. Furthermore, the difference in F1 score between the decision tree with and without Attention is only 0.01. The reason is that the decision tree does not produce a probability and thus is not compatible with our embedding scheme. In addition, the decision tree has high variance and is very sensitive to small changes in the input, which makes it highly deterministic. It results in loss of information when encoding different steps of the attack. Although the problem is alleviated by using the random forest, we find that the random forest also does not perform well with Attention for similar reasons. Therefore, we use *LSTM* for our classifiers of the per-step detectors hereafter.

6.5 Comparison with Other Identification Algorithms

We compare Attention with other traditional identification algorithms for sequences. We consider the following three algorithms:

— *Highest probability*: Highest probability tags a window to the step with the highest probability assigned by the per-step detectors. If the probabilities are identical for a window, the algorithm labels the window in the order of Benign, Action, Reconnaissance, and Infection. The order is based on the number of samples in our dataset.
— *Viterbi*: Viterbi [11] is based on a hidden Markov model and estimates a sequence of hidden states from an observed sequence with memory-less noise.

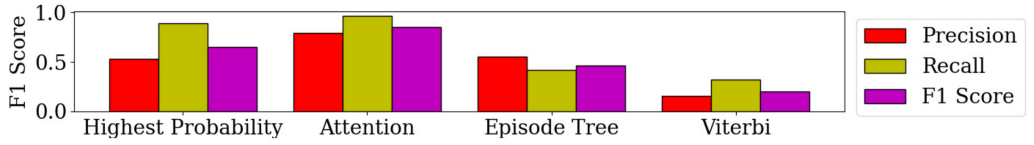

Fig. 6. Comparison with other identification algorithms.

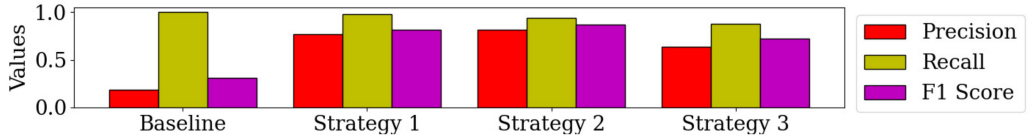

Fig. 7. Impact of self-evolution strategies.

— *Episode tree*: An episode tree is a collection of window sequences. Based on the training set, we build an episode tree using the tree generation algorithm by Mannila et al. [32]. The episode tree identifies infection windows if a given window sequence matches a branch of the episode tree, which contains infection windows.

The results (Figure 6) show that the attention-based algorithm outperforms the other algorithms. The F1 score of the attention-based algorithm is 0.85. The performance of the episode tree (0.46) and Viterbi (0.20) is even worse than the highest probability (0.65).

The episode tree is a simple pattern-matching algorithm; thus, it depends on how many patterns are captured from the training set. Therefore, the episode tree can be easily over-fitted, which accounts for high precision and low recall of its result. Viterbi shows the worst performance due to its memory-less assumption which makes the algorithm unable to capture long-term dependencies.

6.6 Impact of Self-Evolution Strategies

We carry out an experiment to assess the impact of the three strategies discussed in Section 5.3. We compare the performance of the baseline with that of the infection detector updated according to the three strategies. Strategy 1 is the strategy that uses identified infection events, identified benign events, and the original training set for retraining. Strategy 2 is the strategy that only uses identified infection events and the original training set for retraining. Strategy 3 is the strategy that only uses identified benign events and the original training set for retraining. We use those three strategies to evolve the infection detector for one round—that is, the infection detector was only updated once.

The results (Figure 7) show that all updated infection detectors outperform the baselines in terms of precision and F1, regardless of the strategy used. Compared with other strategies, Strategy 2 works best (highest F1 score of 0.87) with the highest precision (0.82). Strategy 1 has an F1 score of 0.82 and a precision of 0.77, and Strategy 3 has an F1 score of 0.72 and a precision of 0.64. We conclude that the self-evolution strategy affects the performance of the infection detector. The results show that many benign samples are classified as malicious under Strategy 1 and Strategy 3 (more FPs), resulting in worse performance on precision and F1 score compared to Strategy 2.

6.7 Comparison with Other Attention-Based NIDSes

We compare our NIDS via the LSTM-based Seq2Seq model with an attention mechanism with existing attention-based NIDSes concerning two aspects. First, we assess whether the performance of LSTM after being evolved is comparable to the performance of those NIDSes. Second, we assess

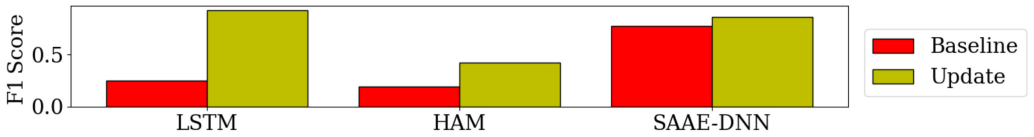

Fig. 8. Comparison with other attention-based NIDSes.

whether our attention-based identification is also beneficial to them. In our analysis, the following two approaches are considered:

- *HAM*: HAM [29] employs two attention layers: the feature-based attention layer and the slice-based attention layer. The former weighs the features and the latter calculates an attention score for a time window considering a specific number of previous windows. Finally, the NIDS predicts the next window with the neural network.
- *SAAE-DNN*: SAAE-DNN [50] is based on a stacked autoencoder with the attention mechanism. It consists of two autoencoders. In between an encoder layer and a latent layer of each autoencoder, an attention layer is inserted. The latent nodes of the second autoencoder are connected to the four-layer neural network, which finally outputs the classification result.

In the experiments, we use two test sets (referred to as set_1 and set_2) with different networking patterns. First, we evaluate the F1 score of LSTM, HAM, and SAAE-DNN on both set_1 and set_2. Then, we apply our self-evolution Strategy 2 to update the models based on the result on set_1. We refer to the evolved models as Updated LSTM, Updated HAM, and Updated SAAE-DNN, respectively. Finally, we measure the F1 score of the updated models on set_2. We compare the results of LSTM, HAM, SAAE-DNN, and their updated models on set_2 (Figure 8).

Our conclusions are as follows. First, our Updated LSTM outperforms HAM and SAAE-DNN. The F1 score of the original LSTM (standard trained infection detector) is only 0.14, which is lower than the scores of HAM (0.19) and SAAE-DNN (0.77). However, the F1 score of LSTM becomes the highest (0.92) after being evolved. These results show that our self-evolution strategy can achieve a classifier performance comparable to the one of existing NIDSes. Second, existing NIDSes can benefit from our approach. After being evolved, the F1 scores of HAM and SAAE-DNN increase from 0.19 to 0.42 and from 0.77 to 0.86, respectively.

7 ENHANCED SELF-EVOLUTION STRATEGIES

Although the self-evolution strategy based on Definition 3 is effective in reducing the FPs of the infection detector, it does not take into account those real malicious events that may be misclassified as benign by the infection detector. Especially when there are adversarial evasion attacks against DNN-based detectors, both these adversarial infection windows and non-adversarial infection windows that are easy to escape detection by infection detectors also need to be included in the relabeling range of our sequence analyzer. To this end, we further redefine the evolution set and improve the previous self-evolution strategies.

Definition 4 (Enhanced Evolution Set). The definitions of set $\mathcal{A}$ and C are consistent with Definition 3, except that set $\mathcal{B}$ is defined here as a set of events tagged as infection by the LSTM-based translation model with the attention mechanism. As a result, set C is the only subset of $\mathcal{A}$. We label all events in set $\mathcal{B}$ as infection and all events in set C as benign.

Compared with the definition of set $\mathcal{B}$ in Definition 3, in Definition 4, set $\mathcal{B}$ not only contains samples that are predicted as infection by both the sequence analyzer and the infection detector at the same time but also contains samples that are only predicted as infection by the sequence

analyzer. Since the definition of set $\mathcal{B}$ only affects Strategy 1 and Strategy 2, we describe them under the new definition as follows, and we let $\mathcal{D}_{inf-tra}$ be the original windows training set for training the infection detector:

- *Strategy 1*: The binary classifier of the infection detector is retrained over a retraining set, consisting of windows in all events in $\mathcal{B}$, windows in all events in C, and $\mathcal{D}_{inf-tra}$. For example, suppose that there are five events $e_1, e_2, e_3, e_4, e_5 \in \mathcal{A}$ (i.e., $e_i.l.i = 1$ for $i = \{1, 2, 3, 4, 5\}$) and the attention-based infection identification algorithm provides the information that five events $e_1, e_2, e_4, e_6, e_7 \in \mathcal{B}$ are infection events (i.e., $e_1.t = e_2.t = e_4.t = e_6.t = e_7.t = I$). Then, ARIoTEDef updates the infection detector with $e_1.w, e_2.w, e_4.w, e_6.w, e_7.w$ labeled as Infection, $e_3.w, e_5.w$ labeled as Benign, and $\mathcal{D}_{inf-tra}$.
- *Strategy 2*: This strategy is similar to Strategy 1 except that it only uses windows in all events in $\mathcal{B}$ and $\mathcal{D}_{inf-tra}$, which adds the information about the TPs to the updated model. In the preceding example, $e_1.w, e_2.w, e_4.w, e_6.w, e_7.w$ labeled as Infection and $\mathcal{D}_{inf-tra}$ are used to retrain the infection detector.

The main purpose of the new definition is not only to identify actual benign events but also to identify actual malicious events that are misclassified as benign by the infection detector so that they can be used to reduce the FNs of the infection detector after evolving. In addition, those adversarial evasion samples that are easily misclassified as benign by the infection detector will also be relabeled and used to retrain the infection detector under the enhanced self-evolution strategy.

Due to the utilization of the attention mechanism, the event sequence analyzer based on the Seq2Seq translation model can track the actual infection event by combining the reconnaissance event and action event that occurred before and after the infection event, so it has sharper identification capabilities than the per-step infection detector on complex adversarial infection behaviors. Therefore, evolving the infection detector using the prediction results of the sequence analyzer as supervisory information can improve its performance in identifying both conventional (non-adversarial) and ML-based adversarial infection behaviors.

8 EXPERIMENTS IN THE ADVERSARIAL SETTING

This section provides the experimental analysis on clean datasets and adversarial datasets consisting of DL-based adversarial evasion samples.

8.1 Experimental Setup

We maintain the same model setting as that in Section 6. However, updates have been made in terms of the testbed, adversarial attack dataset, and evaluation content.

8.1.1 Testbed. We conducted the experiments on an NVIDIA GeForce P8 GPU and CUDA V12.0. We implemented the adversarial evasion attacks using the Adversarial Robustness Toolbox [40].

8.1.2 Dataset. We use the reconnaissance dataset, infection dataset, and action dataset generated based on the Mirai dataset [1] with the method described in Section 6. Then, we use a Standard Scaler normalization function to scale the feature values of each window sample in the dataset to a normal distribution with a mean of 0 and a standard deviation of 1. Information about the normalized dataset is shown in Table 3. The value range (MinVal, MaxVal) of the normalized input samples will be passed to the adversarial sample generation algorithm as a parameter.

8.1.3 Adversarial Evasion Samples Generation. To evaluate the robustness of ARIoTEDef against white-box attacks (see Section 3.2), we consider the following two well-known adversarial example generation algorithms:

Table 3. Normalized Dataset

Windows Dataset	Training Set					Test Set				
	Total	BenNum	MalNum	MinVal	MaxVal	Total	BenNum	MalNum	MinVal	MaxVal
Reconnaissance	14,410	7,205	7,205	−2.44	96.18	4,233	1,156	3,077	−3.30	21.12
Infection	19,818	9,909	9,909	−2.37	140.73	4,233	3,915	318	−3.30	21.12
Action	19,152	9,576	9,576	−3.04	138.32	4,233	4,110	123	−3.30	21.12

— *FGSM*: FGSM (Fast Gradient Sign Method) [13] is a one-step adversarial attack that perturbs input by adding perturbation based on the sign of the gradients of the loss function w.r.t. the input. However, FGSM may produce less effective perturbations in complex DNNs.
— *PGD*: PGD (Projected Gradient Descent) [31] is an iterative version of FGSM. It applies FGSM multiple times with small per-step budget sizes and then projects the perturbations back to a pre-defined ϵ ball to ensure that the perturbations remain within a reasonable range. PGD is effective against a broader range of models.

We input the clean samples one by one in the original dataset to the adversarial example generation algorithm to obtain corresponding adversarial samples. According to the terminology used in the literature, the collection of clean samples is referred to as *clean dataset*, and the collection of adversarial samples is referred to as *adversarial dataset*. Thus, we refer to the original infection test set shown in Table 3 as the *clean infection test set* and the set of adversarial evasion samples generated from the infection samples in the clean infection test set as the *adversarial infection test set*. Note that the clean infection test set contains clean benign and clean infection samples, whereas the adversarial infection test set only contains adversarial infection samples. When generating adversarial samples, we need to set the following budget parameters:

— *Perturbation budget* ϵ: This parameter determines the maximum allowable adversarial perturbation on the clean sample x. The value range of $(\epsilon + x)$ is (MinVal, MaxVal).
— *Step size* ϵ_{ite}: This parameter determines the per-step maximum allowable perturbation.
— *Maximum number of iterations* n_{ite}: This parameter determines the maximum number of iterations a clean sample is allowed to be perturbed when making an adversarial sample.

8.2 Impact of Budget Settings of Adversarial Evasion Attacks

We first train the reconnaissance detector, infection detector, and action detector using the clean training sets. After training for 40 epochs with a batch size of 32, we obtain a baseline reconnaissance detector with a *clean accuracy* (accuracy on the clean test set) of 96.48%, a baseline infection detector with a clean accuracy of 94.80%, and a baseline action detector with a clean accuracy of 97.73%. Then, we conduct the robustness evaluation experiments on two early-stage baseline detectors, a reconnaissance detector and an infection detector, using the white-box FGSM and PGD attacks. We employ optimization methods for untargeted attacks and targeted attacks (see Section 3.2) to construct adversarial reconnaissance samples and adversarial infection samples, respectively. In addition, we explore different perturbation budget settings.

Results in Table 4 show that under the optimization approach for untargeted attacks, with the perturbation budget ϵ set to 1.0, step size ϵ_{ite} set to 0.5, and the number of iterations n_{ite} set to 20, PGD most frequently achieves the highest attack success rate. We also conduct white-box adversarial attacks on the action detector, but the success rate is almost 0. Therefore, in all of the following experiments, we use the *strongest PGD* adversarial samples generated in the preceding setting (ϵ =1.0, ϵ_{ite} =0.5, n_{ite} =20) to evaluate the robustness of the infection detector.

Table 4. Performance of Pre-Evolved Detectors on White-Box Adversarial Samples

Optimize	Input				Reconnaissance Detector				Infection Detector			
	Name	ϵ	ϵ_{ite}	n_{ite}	TP	FN	FNR (%)	Recall (%)	TP	FN	FNR (%)	Recall (%)
–	Clean	–	–	–	3,064	13	0.42	99.58	126	192	60.38	39.62
Untargeted	FGSM	0.5	–	–	66	3,011	97.86	2.14	5	313	98.43	1.57
	FGSM	1.0	–	–	48	3,029	98.44	1.56	0	318	**100.00**	**0.00**
	PGD	0.5	0.25	20	48	3,029	98.44	1.56	5	313	98.43	1.57
	PGD	1.0	0.5	20	29	3,048	**99.06**	**0.94**	0	318	**100.00**	**0.00**
Targeted	FGSM	0.5	–	–	67	3,010	97.82	2.18	110	208	65.41	34.59
	FGSM	1.0	–	–	52	3,025	**98.31**	**1.69**	96	222	69.81	30.19
	PGD	0.5	0.25	20	89	2,988	97.11	2.89	100	218	68.55	31.45
	PGD	1.0	0.5	20	239	2,838	92.23	7.77	4	314	**98.74**	**1.26**

8.3 Impact of the Threshold of the Sequence Analyzer

To analyze event sequences and identify infection events, we train a Seq2Seq translation model based on the LSTM cells and the attention mechanism according to the standard training strategy. We set the sequence length to 10—that is, every 10 events form an event sequence. Since ARIoTEDef uses equal-length sequence translation, the output is a tag sequence with a length of 10 as well. The event samples used for training and testing the sequence analyzer are generated by three per-step detectors from the infection window dataset.

We can see from Table 3 that the number of benign samples in the infection test set is much larger than the number of infection samples. Thus, for the sequences of events, the benign events in a sequence will appear much more frequently than the infections. Since the attention-based sequence analyzer refers to contextual events when predicting each event, under such a mechanism, benign events can easily have benign scores close to 0, whereas malicious events cannot easily have malicious scores close to 1. As a result, the score threshold between benign and malicious categories has a key impact on the performance of the sequence analyzer.

To identify as many TP and **true negative (TN)** samples as possible during the self-evolution of ARIoTEDef, we train the sequence analyzer with different thresholds and random seeds and evaluate its performance on the clean infection test set. The experimental results are shown in Table 5. The results show that when using the commonly used score threshold of 0.5, the Seq2Seq model exhibits a very low **false-positive rate (FPR)** but has a high **false-negative rate (FNR)**, which shows that many malicious events are mislabeled as benign. After we reduce the score threshold to 0.1, there is a clear improvement in the FNR. This shows that some of the malicious samples that were mislabeled in the previous threshold setting are correctly tagged under the current setting. In addition, the FNR increases but to a limited extent. Through multiple tries, we found that when the threshold is set to 0.01, an ideal balance can be achieved between precision and recall. More intuitively, such a threshold results in the highest average F1 score. Therefore, in all of the following experiments, we use a threshold of 0.01 for the sequence analyzer.

8.4 Robustness of the Evolved Infection Detector in Adversarial Setting I

8.4.1 Adversarial Attack and Defense Setting I. Assume that the attacker launches a total of N rounds of adversarial evasion attacks against ARIoTEDef. The clean infection test set (see Section 8.1.3) is denoted as X, and the pre-evolved infection detector is denoted as M_0. In Adversarial Setting I, the attacker constructs adversarial infection samples based on the same clean infection test set and the latest evolved target infection detector each time to launch an adversarial evasion attack. Note that since the adversarial infection samples used in each attack round are crafted

Table 5. Performance of the Sequence Analyzer on a Clean Test Set

Threshold	Seed	TP	FN	TN	FP	FNR (%)	FPR (%)	Precision (%)	Recall (%)	F1 (%)	Accuracy (%)
	0	75	277	3,869	3	78.69	0.08	96.15	21.31	34.88	93.37
	1	93	259	3,870	2	73.58	0.05	97.89	26.42	41.61	93.82
0.5	2	62	290	**3,872**	**0**	82.39	**0.00**	**100.00**	17.61	29.95	93.13
	3	60	292	3,871	1	82.95	0.03	98.36	17.05	29.06	93.06
	Average	73	280	3,871	2	79.40	0.04	98.10	20.60	33.88	93.35
	0	117	235	3,869	3	66.76	0.08	97.50	33.24	49.58	**94.37**
	1	126	226	3,857	15	64.20	0.39	89.36	35.80	51.12	94.29
0.1	2	79	273	3,845	27	77.56	0.70	74.53	22.44	34.50	92.90
	3	76	276	3,863	9	78.41	0.23	89.41	21.59	34.78	93.25
	Average	100	253	3,859	14	71.73	0.35	87.70	28.27	42.49	93.70
	0	138	214	3,841	31	60.80	0.80	81.66	39.20	**52.98**	94.20
	1	**163**	**189**	3,771	101	**53.69**	2.61	61.74	**46.31**	52.92	93.13
0.01	2	158	194	3,695	177	55.11	4.57	47.16	44.89	46.00	91.22
	3	154	198	3,782	90	56.25	2.32	63.11	43.75	51.68	93.18
	Average	153	199	3,772	100	56.46	2.58	63.42	43.54	50.89	92.93

based on different target models, they belong to different adversarial data distributions. The specific process of attack and defense is as follows:

- —In the 1st round of attack, the attacker constructs an adversarial dataset $AdvX_1$ based on X and M_0, then sends it to the target NIDS. ARIoTEDef relabels $AdvX_1$ according to the improved evolution strategy described in Section 7, and then uses relabeled $AdvX_1$ for retraining M_0 to obtain the 1st updated infection detector M_1.
- —In the 2nd round of attack, the attacker constructs $AdvX_2$ based on X and M_1, then sends it to the target NIDS. ARIoTEDef continues to relabel $AdvX_2$, and then uses the relabeled $AdvX_2$ to retrain M_1 to obtain the 2nd updated detector M_2.
- —The attacker continues with successive rounds of attack, following the strategies from prior rounds, and ARIoTEDef performs relabeling and retraining, as in the previous rounds.
- —In the last round of attack, the attacker constructs an adversarial dataset $AdvX_N$ based on X and M_{N-1}, then sends it to the target NIDS. ARIoTEDef relabels $AdvX_N$, and then uses relabeled $AdvX_N$ to retrain M_{N-1} to obtain the Nth updated infection detector M_N.

For the evaluation of ARIoTEDef in the Adversarial Setting I, for the evolved detector M_N, we use the adversarial test set $AdvX_{N+1}$ constructed based on X and M_N to evaluate the white-box robustness of M_N and use X to evaluate the performance of M_N on the clean test set.

8.4.2 Performance of the Evolved Infection Detector on Adversarial Samples. We answered the following research questions based on experimental data:

- —Does the FN of the infection detector on white-box adversarial samples decrease after multiple rounds of evolution?
- —How many rounds of evolution can the detector be robust to white-box adversarial attacks?

We conducted experiments with different numbers of self-evolution rounds and repeated the experiments with multiple random seeds. According to Table 3, the clean infection test set contains 3,915 clean benign samples and 318 clean malicious samples. In each round of adversarial evasion attack, we construct adversarial infection samples based on these clean infection samples and the

Table 6. Performance of the Evolved Infection Detector on White-Box Adversarial Samples (AdvSetting I)

Evolution Round	0	20				40				80			
Seed	–	0	1	2	average	0	1	2	average	0	1	2	average
TP	0	194	198	227	206	232	233	212	226	229	**305**	250	261
FN	318	124	120	91	112	86	85	106	92	89	**13**	68	57
FNR (%)	100.00	38.99	37.74	28.62	35.12	27.04	26.73	33.33	29.04	27.99	**4.09**	21.38	17.82
Recall (%)	0.00	61.01	62.26	71.38	64.88	72.96	73.27	66.67	70.96	72.01	**95.91**	78.62	82.18

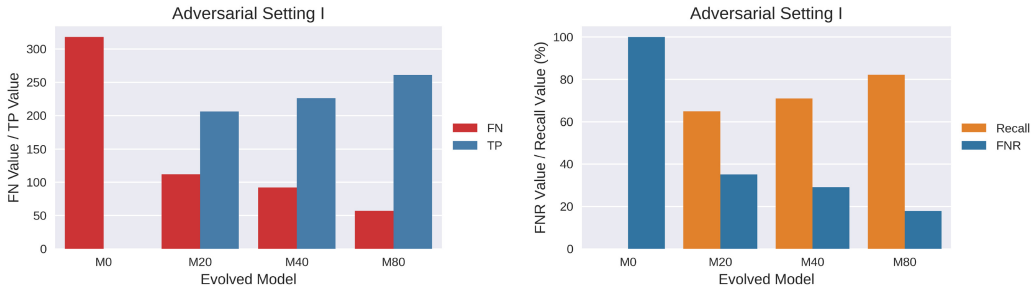

Fig. 9. Average performance of the evolved infection detector on adversarial samples (AdvSetting I).

updated infection detector in the previous round and then use them to attack the target system. After each round of evolution, we construct adversarial infection samples based on these 318 clean infection samples and the latest update of the infection detector to form a white box adversarial infection test set. The evaluation results for our proposed self-evolution strategy on the infection detector against white-box adversarial attacks are shown in Table 6.

We observe that the pre-evolved infection detector (evolution round is 0) is hardly robust against the white-box adversarial PGD attack, since all adversarial infection samples are misclassified as benign, resulting in an FNR close to 100%. After self-evolution, the recall of the updated infection detector significantly improves. After 20 rounds, 40 rounds, and 80 rounds of self-evolution, the average recall of the infection detector increases to 64.88%, 70.96%, and 82.18%, respectively. In the best case, the recall of the infection detector even increases from 0% to 95.91%. Such results show that the self-evolution strategy effectively improves the ability of infection detectors to identify more deceptive malicious samples.

From the average trend shown in Figure 9, we can see that the robustness of the detector is proportional to the number of evolution rounds. The reason is that the more adversarial samples the detector has seen during retraining, the more familiar it is with the patterns of adversarial infections, and it is thus better at identifying anomalies on unseen adversarial test samples.

Since the distribution of adversarial data keeps changing over multiple rounds of evolution, the robustness improvement of the detector is also continuous. In general, within 30 rounds of evolution, the accuracy of the infection detector on white-box adversarial samples can increase to more than 50%, and the FNR starts to converge to 20% since round 40, as shown in Figure 10.

8.4.3 Performance of the Evolved Infection Detector on Non-Adversarial Samples. We answered the following research question based on experimental data:

—Does the FP of the infection detector on non-adversarial samples (clean data) increase after multiple rounds of evolution?

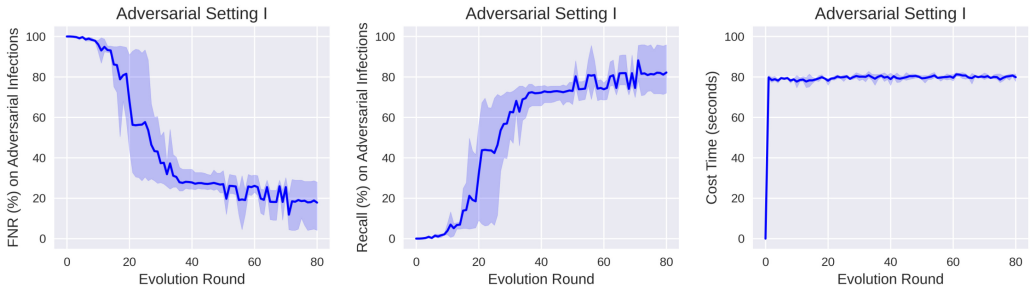

Fig. 10. Impact of the number of evolution rounds on robustness against white-box attack (AdvSetting I).

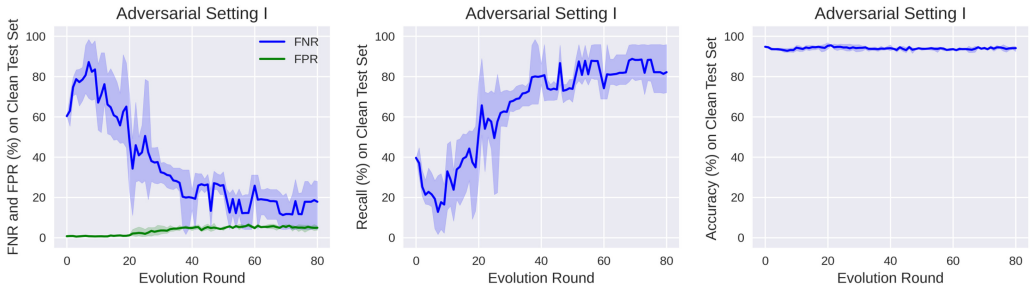

Fig. 11. Impact of the evolution round on performance on the clean test set (AdvSetting I).

Table 7. Performance of Evolved Infection Detector on a Clean Test Set (AdvSetting I)

Evolution Round	Seed	TP	FN	TN	FP	FNR (%)	FPR (%)	Precision (%)	Recall (%)	F1 (%)	Accuracy (%)
0	–	126	192	**3,887**	**28**	60.38	**0.72**	**81.82**	39.62	53.39	94.80
20	0	215	103	3,793	122	32.39	3.12	63.80	67.61	65.65	94.68
	1	214	104	3,766	149	32.70	3.81	58.95	67.30	62.85	94.02
	2	292	26	3,719	196	8.18	5.01	59.84	91.82	72.46	94.76
	average	240	78	3,759	156	24.42	3.98	60.86	75.58	66.98	94.49
40	0	238	80	3,676	239	25.16	6.10	49.90	74.84	59.87	92.46
	1	234	84	3,671	244	26.42	6.23	48.95	73.58	58.79	92.25
	2	214	104	3,783	132	32.70	3.37	61.85	67.30	64.46	94.42
	average	229	89	3,710	205	28.09	5.24	53.57	71.91	61.04	93.05
80	0	229	89	3,685	230	27.99	5.87	49.89	72.01	58.94	92.46
	1	**305**	**13**	3,712	203	**4.09**	5.19	60.04	**95.91**	**73.85**	94.90
	2	250	68	3,770	145	21.38	3.70	63.29	78.62	70.13	**94.97**
	average	261	57	3,722	193	17.82	4.92	57.74	82.18	67.64	94.11

Since the ideal robustness improvement cannot be at the expense of the performance of the detector when dealing with non-adversarial black-box attacks, we evaluate the performance of the evolved infection detector on the clean test set. We collected performance data for the evolved detectors on the clean test set under different evolution rounds and random seed settings. Note that the samples in the clean test set are not used in retraining. Results are shown in Table 7.

We observe that compared with the pre-evolved infection detector (evolution round is 0), the evolved infection detector significantly improves concerning the recall, F1, and an overall unchanged accuracy on the clean test set. From Figure 11, we can see that, compared with the

sharp drop of average FNR of the evolved infection detector on the clean test set, the FPR increase on the clean test set is very weak. This suggests that the evolved infection detector has a stronger ability to identify truly malicious samples (decreased FNR), including those that are non-adversarial. Note that the detector learns to predict some low-confidence abnormal patterns (such as adversarial malicious patterns) as positive during the retraining process, which leads the detector to judge the negative category more strictly than before evolution. Thus, some normal patterns with low confidence can be easily misclassified as anomalies, resulting in a slightly increased FP. However, we can see from the stable accuracy rate close to 95% and the increasing F1 score that the slight decline in precision is almost negligible compared to the significant increase in recall.

8.5 Robustness of the Evolved Infection Detector in Adversarial Setting II

8.5.1 Adversarial Attack and Defense Setting II. Assume that the attacker launches a total of N rounds of adversarial evasion attacks against ARIoTEDef. The clean infection test set introduced in Section 8.1.3 is denoted as X, and the pre-evolved infection detector is denoted as M_0. In Adversarial Setting II, in each new round of adversarial attack, the adversary will not only craft adversarial samples based on the latest evolved detector, but also use a clean infection test set different from the previous round of attacks. To carry out the experiments, we split the clean infection test set X into N sub-datasets $X_1, X_2, \ldots, X_N, X_{N+1}$.

When training without evolution, the clean infection test set is consistent with that described in Table 3. For N rounds of evolution, we split the original clean test set into $N + 1$ sub-test sets. The number of benign samples and the number of malicious samples in each sub-test set are the same. The specific process of attack and defense is as follows:

- In the 1st round of attack, the attacker constructs an adversarial dataset $AdvX_1$ based on X_1 and M_0, then sends it to the target NIDS. ARIoTEDef relabels $AdvX_1$ according to the improved evolution strategy described in Section 7, then uses relabeled $AdvX_1$ for retraining M_0 to obtain a 1st updated detector M_1.
- In the 2nd round of attack, the attacker constructs $AdvX_2$ based on X_2 and M_1, then sends it to the target NIDS. ARIoTEDef continues to relabel $AdvX_2$, and then uses relabeled $AdvX_2$ to retrain M_1 to obtain a 2nd updated detector M_2.
- The attacker continues with successive rounds of attack, following the strategies from prior rounds, and ARIoTEDef performs relabeling and retraining, as in the previous rounds.
- In the last round of attack, the attacker constructs an adversarial dataset $AdvX_N$ based on X_N and M_{N-1}, then sends it to the target NIDS. ARIoTEDef relabels $AdvX_N$, then uses relabeled $AdvXN$ to retrain M_{N-1} to obtain an Nth updated detector M_N.

In Adversarial Setting II, for the evolved detector M_N, we use the adversarial test set $AdvX_{N+1}$ constructed based on X_{N+1} and M_N to evaluate the white-box robustness of M_N and use X_{N+1} to evaluate the performance of M_N on the clean test set. We simultaneously record the performance of two neighboring evolved detectors on an unseen clean test set (M_0 and M_1 on X_2, M_1 and M_2 on $X_3, \ldots, M_{N-1}$ and M_N on X_{N+1}) to analyze the impact of each (1st, 2nd, ..., Nth) round of evolution on the performance of the detector. The main difference between Adversarial Setting II and Adversarial Setting I is that a new clean dataset is used to construct adversarial samples in each attack round.

8.5.2 Performance of the Evolved Infection Detector on Adversarial Samples. We let the infection detector evolve for 5 rounds, 10 rounds, and 20 rounds sequentially. According to the number of evolution rounds, the original infection test set is divided into several sub-datasets (Table 8).

Table 8. Clean Test Sets Used in Adversarial Setting II

Evolution Round	Test Set Num	Test Set Size	Benign Num in Each Test Set	Malicious Num in Each Test Set
0	1	4,233	3,915	318
5	6	705	652	53
10	11	383	355	28
20	21	201	186	15

Table 9. Performance of the Evolved Infection Detector on White-Box Adversarial Samples (AdvSetting II)

AdvSet Size	53					28					15				
Evolution Round	0	5				0	10				0	20			
Seed	–	0	1	2	average	–	0	1	2	average	–	0	1	2	average
TP	0	3	17	3	8	0	6	6	0	4	0	1	1	1	1
FN	53	50	36	50	45	28	22	27	28	24	15	14	14	14	14
FNR (%)	100.00	94.34	**67.92**	94.34	85.53	100.00	78.57	78.57	100.00	85.71	100.00	93.33	93.33	93.33	93.33
Recall (%)	0.00	5.66	**32.08**	5.66	14.47	0.00	21.43	21.43	0.00	14.29	0.00	6.67	6.67	6.67	6.67

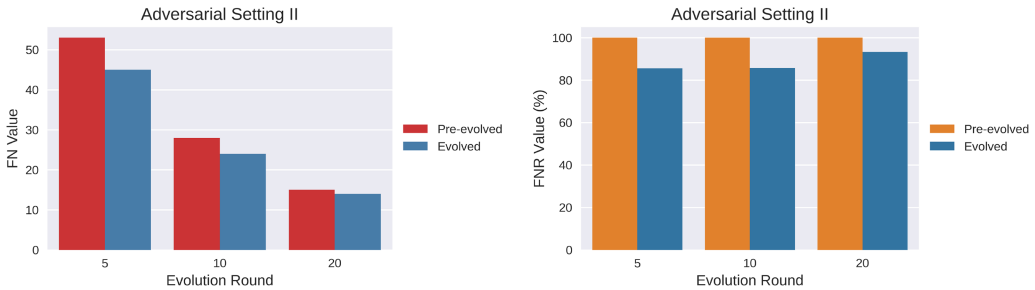

Fig. 12. Average performance of the evolved infection detector on adversarial samples (AdvSetting II).

We answered the following questions based on experimental data:

—Does the FN of the infection detector on white-box adversarial samples decrease after multiple rounds of evolution?
—Does the number of adversarial samples used in each attack round affect the effectiveness of the self-evolution strategy for robustness improvement?

To ensure the fairness of the experiment, when evaluating the pre-evolved detector (baseline model), we also used an adversarial test set of the same size as that for evaluating the evolved detector. We set three random seeds for multiple experiments. The performance of the detector before and after evolution on the white-box adversarial test set is shown in Table 9.

We observe that in tAdversarial Setting II, the pre-evolved infection detectors all have a recall of 0 when evaluated on adversarial test sets of different sizes. This suggests that the pre-evolved detector has difficulty in identifying adversarial infection samples designed to masquerade as benign. After 5, 10, and 20 rounds of evolution, the average recall of the infection detector increased to 14.47%, 14.29%, and 6.67%, respectively, showing that evolution helps to reduce FNs and increase TPs. In addition, we can see from Figure 12 that the number of adversarial samples in each attack and defense round does not essentially affect the effectiveness of the self-evolution strategy. For example, in 20 rounds of evolution, even when there are only 15 adversarial infection samples in

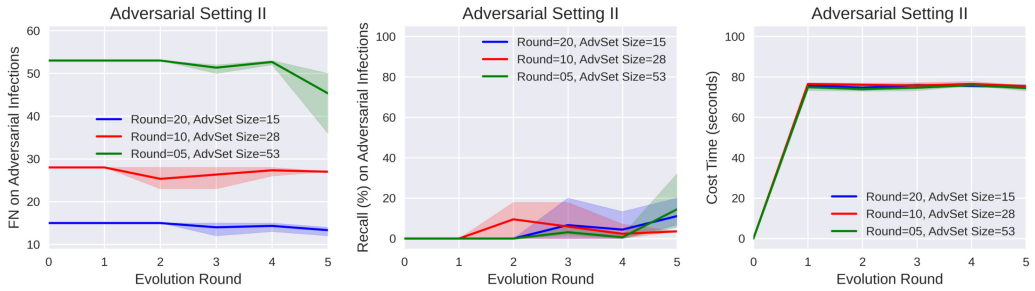

Fig. 13. Impact of the number of adversarial attack samples on the performance (AdvSetting II).

each round of attack, the FNR of the evolved detector is still lower than that before evolution, indicating that the white-box robustness of the detector resulting from the evolution increases.

In general, the more adversarial samples used in each round of attack, the more beneficial the self-evolution strategy is to the improvement of robustness, because more adversarial samples mean that the model will be trained on a wider variety of situations. If only a small number of adversarial samples are used, the adversarial perturbation patterns that the model can learn are very limited, making it difficult to generalize to other unknown perturbation patterns. However, thanks to the multi-step detection approach and the attention mechanism, if adversarial samples are generated based on unknown patterns, these samples can be detected as malicious with high probability, and thus the infection detector can be retrained properly.

To assess the impact of the size of the adversarial dataset on the improvement of robustness, we collected performance data for the first 5 rounds in 10-round evolution and 20-round evolution and compared them with the performance of 5-round evolution. The results are shown in Figure 13. Although not every round of adversarial samples with a larger size can win the race with the infection detector (because the number of adversarial samples used for evolution also depends on the prediction by the sequence analyzer), we can conclude that the higher the number of attack samples, the more the number of FNs decreases. The number of attack samples depends on the attacker's behavior, whether the attacker is fast and sends a lot of malicious samples in a short time or is slow and sends malicious samples over long periods. The more attack samples sent, the more beneficial the self-evolution strategy is to improve the robustness of the system. However, for both types of behavior, ARIoTEDef is robust because of the multi-detector approach.

8.5.3 Performance of the Evolved Infection Detector on Non-Adversarial Samples. We answer the following question based on experimental data:

—Does the FP of the infection detector on non-adversarial samples (clean data) increase after multiple rounds of evolution?

We performed 5 rounds, 10 rounds, and 20 rounds of evolution on the infection detector, and evaluated the performance of the detector on the clean test set before and after each round of evolution. It is worth emphasizing that the test set used for the evaluation before and after any round of evolution is the same, but the test sets between different rounds of evolution are various. The purpose of this design is to ensure that the test samples never participate in the retraining of the detector. The specific construction method of the test set is described in Section 8.5.1. We repeated the experiment using three random seeds and show the results in Figure 14.

The left column graph and right column graph of each x-axis coordinate point show the performance of the detector on the clean test set before and after the corresponding round of evolution, respectively. We observe that the self-evolution strategy results in only a very slight increase in

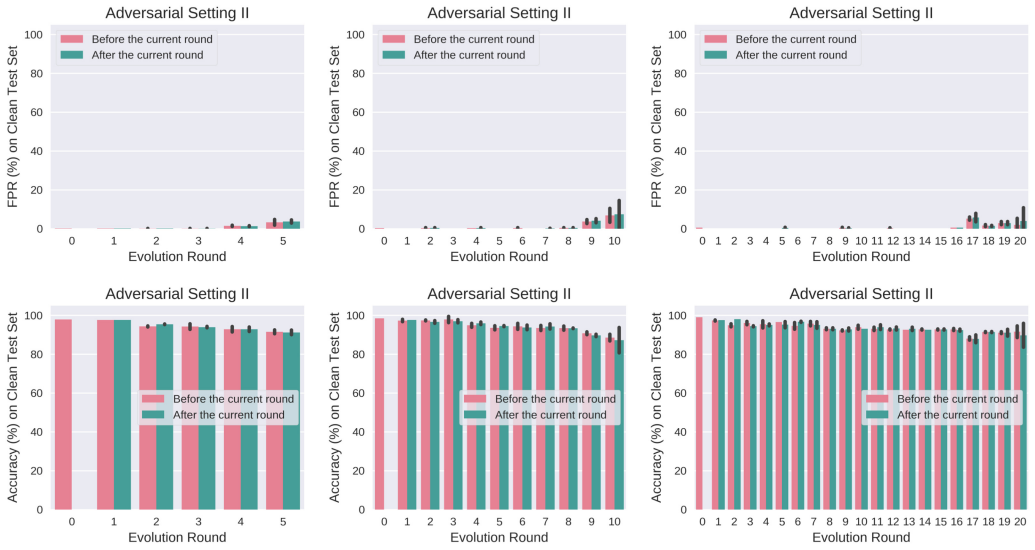

Fig. 14. Performance of the evolved infection detector on the clean test set (AdvSetting II).

the FPR under different numbers of retraining rounds. Combined with the accuracy performance on the clean test set, the detector after multiple rounds of evolution is not only more robust against strong white-box adversarial evasion attacks but also remains stable on non-adversarial samples.

8.6 Computational, Storage, and Time Overhead

In IoT settings, rapid data processing is crucial, requiring DNN-based NIDS to swiftly predict and respond to potential intrusions. Our solution prioritizes lightweight models suitable for IoT scenarios with limited computing and storage. To assess the applicability of our solution, we conducted a comprehensive evaluation from the perspective of the number of model parameters, the storage space required for deployment, and time costs in each stage.

8.6.1 Computational and Storage Overhead. We separately counted the computational overhead of the model during training and inference. Training, typically on powerful servers without affecting IoT devices that only need to deploy the trained model, incurs a total of 218,753 neuron parameters for each per-step detector (Reconnaissance, Infection, Action) and 200,705 for the sequence analyzer. Maximum GPU power consumption during training is 115W for per-step detectors and 152W for the sequence analyzer. In the inference phase, with the architecture, weights, optimizer status, and other information of the model stored in HDF5 format, each per-step detector and sequence analyzer has a size of 2.6M and 2.4M, respectively.

8.6.2 Time Overhead. We also independently analyzed the time overhead of different modules in ARIoTEDef during training and inference, detailed in Figure 15. In the training phase, per-step detectors (Reconnaissance, Infection, Action) converge within 40 epochs, whereas each epoch takes 0.75s, 1.05s, and 1.0s, respectively. The sequence analyzer, due to the need to learn more complex event sequence patterns, takes an average of 12.25s per epoch.

In the inference phase, when the trained models are used for prediction, the per-step detector (represented by the infection detector) only takes 0.007s to predict a window, and the sequence analyzer tags an event sequence in 0.065s. In addition, assuming that the attacker sends 318 adversarial samples in each round, the time overhead of evolution will be related to the number of

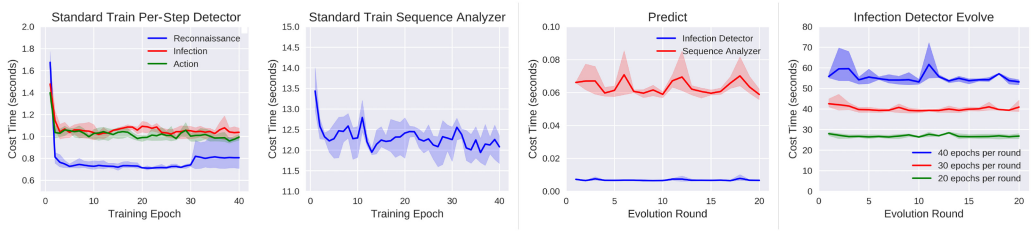

Fig. 15. Time cost in the training and inference phases.

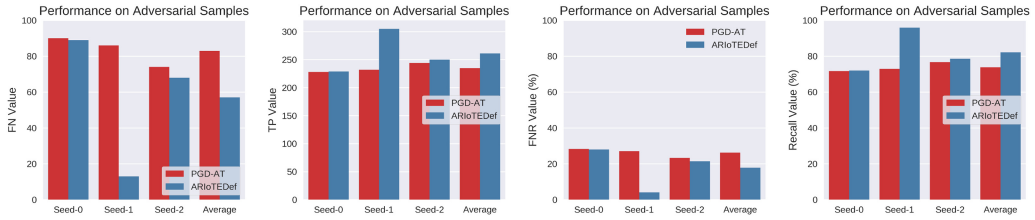

Fig. 16. Adversarial robustness performance comparison on white-box adversarial samples.

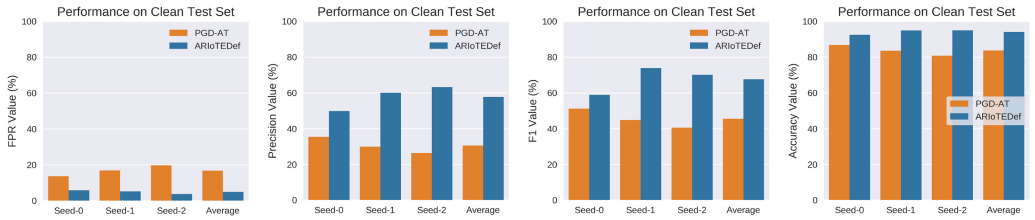

Fig. 17. Regular performance comparison on the clean test set.

epochs required for retraining the infection detector in each round. Under the default setting of 40 epochs, it takes about 55s for per-round evolution.

8.7 Robustness Performance Comparison with Existing Work

To fully demonstrate the robustness performance of ARIoTEDef in adversarial environments, we also compare it with the powerful **PGD-based Adversarial Training (PGD-AT)** [31], which is most often chosen as a strong baseline for robustness comparisons. It involves creating adversarial samples based on the original training samples and incorporating them into the training set to optimize the model. However, different from PGD-AT, which assumes that the defender has prior knowledge of adversarial attack methods and can actively construct adversarial samples, we do not generate adversarial samples but only capture and use advanced sequence analyzers to tag adversarial samples. Therefore, we not only relax the assumptions about the defender's capabilities but also avoid expensive generation overhead. As can be seen from Figure 16, after three experiments, the recall of the 80-round evolved infection detector on white-box adversarial samples is 8.38% higher than that of PGD-AT trained detector on average. This suggests that more adversarial infection samples are correctly identified. In addition, as shown in Figure 17, our evolved model also has higher precision (27.13% higher), F1 score (22.11% higher), accuracy (10.39% higher), and a lower FPR (11.82% lower) on non-adversarial infection and benign samples than the PGD-AT trained detector on average. This indicates that fewer actual normal samples are misclassified as infection.

9 CONCLUSION

In this article, we introduced *ARIoTEDef*, a kill chain-based approach for early detection of persistent attacks against IoT devices. To improve the robustness of the infection detector, ARIoTEDef adopts a feedback strategy that backtracks past events to identify infection events when anomalies at the later steps are detected. We showed that ARIoTEDef is not only effective in detecting non-adversarial black-box infection attacks but also improves the robustness of the infection detector against DL-based adversarial white-box attacks. In the case of PGD, the most powerful white-box adversarial attacks to our knowledge, our experiments showed that after 20 rounds, 40 rounds, and 80 rounds of self-evolution, the average recall of the infection detector on the evasion attack improves from 0% to 64.88%, 70.96%, and 82.18%, respectively. We plan to enhance our approach with host information, such as system calls and CPU/memory and resource usage, and more steps of the cyber kill chain, such as lateral movement and obfuscation, as part of future work. The implementation of the source codes is released at https://github.com/ariotedef.

REFERENCES

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. 2017. Understanding the Mirai botnet. In *Proceedings of the USENIX Security Symposium*. 1093–1110.

[2] D. Bahdanau, K. H. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of International Conference on Learning Representations (ICLR'15)*. 1–15.

[3] L. Bilge and T. Dumitras. 2012. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, USA, 833–844.

[4] W. Brendel, J. Rauber, and M. Bethge. 2018. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*. 1–12.

[5] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath. 2021. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology* 12, 5 (2021), 1–32.

[6] J. Chen, M. I. Jordan, and M. J. Wainwright. 2020. HopSkipJumpAttack: A query-efficient decision-based attack. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'20)*. IEEE, 1277–1294.

[7] K. Cho, B. Van Merrienboer, D. Bahdanau, and Y. Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the 8th Workshop on Syntax, Semantics, and Structure in Statistical Translation (SSST'14)*. 103–112.

[8] E. Cole. 2016. Threat hunting: Open season on the adversary. *SANS Institute Information Reading Room* 1, 1 (2016), 1–23.

[9] CoreSecurity. 2014. Pcapy. Retrieved April 24, 2024 from https://github.com/helpsystems/pcapy

[10] D. Dingee. 2019. IoT, Not People, Now the Weakest Link in Security. Retrieved April 24, 2024 from https://devops.com/iot-not-people-now-the-weakest-link-in-security/

[11] G. D. Forney. 1973. The Viterbi algorithm. *Proceedings of the IEEE* 61, 3 (1973), 268–278.

[12] Y. Fu, Z. Yan, J. Cao, O. Koné, and X. Cao. 2017. An automata-based intrusion detection method for Internet of Things. *Mobile Information Systems* 1, 1 (2017), 1–13.

[13] I. J. Goodfellow, J. Shlens, and C. Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*. 1–11.

[14] A. Goodge, B. Hooi, S. K. Ng, and W. S. Ng. 2020. Robustness of autoencoders for anomaly detection under adversarial impact. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'20)*. 1244–1250.

[15] A. Graves. 2012. Long short-term memory. In *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence, Vol. 385. Springer, 37–45.

[16] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee. 2007. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proceedings of the USENIX Security Symposium*. 1–16.

[17] S. Haas and M. Fischer. 2018. GAC: Graph-based alert correlation for the detection of distributed multi-step attacks. In *Proceedings of the Annual ACM Symposium on Applied Computing (SAC'18)*. ACM, New York, NY, USA, 979–988.

[18] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer. 2020. UNICORN: Runtime provenance-based detector for advanced persistent threats. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'20)*. 1–18.

[19] E. Hutchins, M. Cloppert, and R. Amin. 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Information Warfare & Security Research* 1, 1 (2011), 80.
[20] K. A. Jallad, M. Aljnidi, and M. S. Desouki. 2020. Anomaly detection optimization using big data and deep learning to reduce false-positive. *Journal of Big Data* 7, 1 (2020), 1–12.
[21] M. Javed and V. Paxson. 2013. Detecting stealthy, distributed SSH brute-forcing. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. ACM, New York, NY, USA, 85–95.
[22] H. Kang, D. Ahn, G. Lee, J. Yoo, K. Park, and H. Kim. 2019. IoT Network Intrusion Dataset. Retrieved April 24, 2024 from https://ieee-dataport.org/open-access/iot-network-intrusion-dataset
[23] Keras. 2016. Home Page. Retrieved April 24, 2024 from https://keras.io/
[24] F. Klassen and AppNeta. 2018. Tcpreplay. Retrieved April 24, 2024 from https://tcpreplay.appneta.com/
[25] B. Krebs. 2017. Reaper: Calm Before the IoT Security Storm. Retrieved April 24, 2024 from https://krebsonsecurity.com/2017/10/reaper-calm-before-the-iot-security-storm/
[26] B. Lantz, B. Heller, and N. McKeown. 2010. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets'10)*. ACM, New York, NY, USA, 1–6.
[27] A. H. Lashkari. 2018. CICFlowMeter Features. Retrieved April 24, 2024 from https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt
[28] H. Lee, A. Mudgerikar, A. Kundu, N. Li, and E. Bertino. 2022. An infection-identifying and self-evolving system for IoT early defense from multi-step attacks. In *Proceedings of the European Symposium on Research in Computer Security*. 549–568.
[29] C. Liu, Y. Liu, Y. Yan, and J. Wang. 2020. An intrusion detection model with hierarchical attention mechanism. *IEEE Access* 8 (2020), 67542–67554.
[30] M. T. Luong, H. Pham, and C. D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'15)*. 1412–1421.
[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*. 1–28.
[32] Heikki Mannila, Hannu Toivonen, and A. I. Verkamo. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* 1, 3 (1997), 259–289.
[33] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino. 2017. Kalis: A system for knowledge-driven adaptable intrusion detection for Internet of Things. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'17)*. IEEE, 656–666.
[34] S. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. 2019. Holmes: Real-time apt detection through correlation of suspicious information flows. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'19)*. IEEE, 1137–1152.
[35] S. M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. DeepFool: A simple and accurate method to fool deep neural networks. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 2574–2582.
[36] Msehgal. 2021. Protect Your IoT Devices from Log4j 2 Vulnerability. Retrieved April 24, 2024 from https://live.paloaltonetworks.com/t5/community-blogs/protect-your-iot-devices-from-log4j-2-vulnerability/ba-p/453381
[37] A. Mudgerikar, P. Sharma, and E. Bertino. 2019. E-Spion: A system-level intrusion detection system for IoT devices. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*. ACM, New York, NY, 493–500.
[38] J. Navarro, A. Deruyver, and P. Parrend. 2018. A systematic survey on multi-step attack detection. *Computers & Security* 76 (2018), 214–249.
[39] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A. R. Sadeghi. 2019. DÏoT: A federated self-learning anomaly detection system for IoT. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'19)*. IEEE, 756–767.
[40] M. I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, Ian M. Molloy, and Ben Edwards. 2018. Adversarial Robustness Toolbox v1.0.0. *arXiv preprint arXiv:1807.01069* (2018).
[41] C. Osborne. 2021. This Is Why the Mozi Botnet Will Linger On. Retrieved April 24, 2024 from https://www.zdnet.com/article/this-is-why-the-mozi-botnet-will-linger-on/
[42] D. Palmer. 2022. This Sneaky Hacking Group Hid Inside Networks for 18 Months without Being Detected. Retrieved April 24, 2024 from https://www.zdnet.com/article/this-sneaky-hacking-group-hid-inside-networks-for-18-months-without-being-detected/
[43] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. 2016. The limitations of deep learning in adversarial settings. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P'16)*. IEEE, 372–387.

[44] B. Pinkas and T. Sander. 2002. Securing passwords against dictionary attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'02)*. ACM, New York, NY, USA, 161–170.
[45] Check Point Research. 2017. IoTroop Botnet: The Full Investigation. Retrieved April 24, 2024 from https://research.checkpoint.com/2017/iotroop-botnet-full-investigation/
[46] M. Sarkar. 2000. Modular pattern classifiers: A brief survey. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'00)*. IEEE, 2878–2883.
[47] Sqrrl. 2018. A Framework for Cyber Threat Hunting. Retrieved April 24, 2024 from https://www.threathunting.net/files/framework-for-threat-hunting-whitepaper.pdf
[48] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas. 2018. *MITRE ATT&CK: Design and Philosophy. Technical Report.* MITRE.
[49] I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS'14)*. 1–9.
[50] C. Tang, N. Luktarhan, and Y. Zhao. 2020. SAAE-DNN: Deep learning method on intrusion detection. *Symmetry* 12, 10 (2020), 1695.
[51] N. Wang, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou. 2021. MANDA: On adversarial example detection for network intrusion detection system. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'21)*. IEEE, 1–10.
[52] T. Yadav and A. M. Rao. 2015. Technical aspects of cyber kill chain. In *Proceedings of the International Symposium on Security in Computing and Communication (SSCC'15)*. 438–452.

Received 22 August 2023; revised 2 February 2024; accepted 24 March 2024