An Infection-Identifying and Self-Evolving System for IoT Early Defense from Multi-Step Attacks

Hyunwoo Lee[†], Anand Mudgerikar[†], Ashish Kundu[‡],
Ninghui Li[†], and Elisa Bertino[†]

Purdue University[†], Cisco Research[‡]
[†]{lee3816,amudgeri,ninghui,bertino}@purdue.edu, [‡]ashkundu@cisco.com

Abstract. Internet-of-Things (IoT) cyber threats such as jackware [14] and cryptomining [33] show that insecure IoT devices can be exploited by attackers with different goals. As many such attacks are multi-steps, early detection is critical. Early detection enables early attack containment and response, and prevention of malware propagation. However, it is challenging to detect early-phase attacks with both high precision and high recall as attackers typically attempt to evade the detection systems with stealthy or zero-day attacks. To enhance the security of IoT devices, we propose IoTEDef, a deep learning-based system able to identify the infection events and evolve with the identified infections. IoTEDef understands multi-step attacks based on cyber kill chains and maintains detectors for each step. When it detects anomalies related to a later stage of the kill chain, IoTEDef backtracks the log of events and analyzes these events to identify infection events. Then, IoTEDef updates its infection detector with the identified events. IoTEDef can be used for threat hunting as well as the generation of indicators of compromise and attacks. To show its feasibility, we implement a prototype of the system and evaluate it against the Mirai botnet campaign [2] and the multi-step attack that exploits the Log4j vulnerability [36] to infect the IoT devices. Our results show that the F1-score of our evolved infection detector in IoTEDef, instantiated with long short-term memory (LSTM) and the attention mechanism, increases from 0.31 to 0.87. We also show that existing attention-based NIDSes can benefit from our approach.

Keywords: internet of things, multi-step attacks, infection identification, threat hunting, attention mechanism

1 Introduction

The Internet of Things (IoT) is the network of physical objects (or "things"), embedding electronics, software, and network connectivity, which enable these objects to collect and exchange data. IoT allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer-based systems. IoT technology thus enables many novel applications and business opportunities [1].

However, IoT devices are at higher security risks than conventional computer systems [4]. Such devices often have access to attackers' targets such as sensitive data, cyber-physical systems, and/or user credentials [9]. Coming up with security techniques for IoT devices is challenging because these devices are often resource-constrained and thus hardly able to defend themselves. Often they are not appropriately hardened, nor regularly patched, and not even managed according to security best practices, leading them to become easy targets for attackers.

Very often, attack campaigns aiming at compromising IoT devices include multiple steps to acquire a foothold in a targeted system. For example, recent botnet campaigns, such as Reaper [26] or Mozi [38], scan ports to find any vulnerable entry points of the target device and attempt to take over it by performing telnet dictionary attacks or zero-day attacks. Once such a foothold is established, the attacker maintains persistence in the system, spreading malware to other devices, ex-filtrating confidential data, or stealing credentials. Therefore, detecting attacks at an early stage and identifying infection vectors are critical in order to contain and respond to the attacks, prevent re-infection, and fully remove the attacker's footholds.

However, it is challenging to detect early-phase attacks with both high precision and high recall. The reason is that to acquire a foothold in a target system, an attacker typically attempts to evade the detection systems by performing stealthy attacks (e.g., stealthy, distributed SSH brute-forcing [22]) or exploiting completely unknown device vulnerabilities (i.e., zero-day attacks) [26, 39, 40]. To detect such attacks, the detectors may classify all the suspicious or unknown patterns as anomalies, but it results in a high number of false positives [21].

In this paper, we propose IoTEDef, an anomaly-based network intrusion detection system (NIDS) tailored for IoT devices, which is *kill chain-based*, *infection-identifying*, and *self-evolving*. As IoTEDef is *network-based*, it supports resource-constrained IoT devices without requiring additional computation or networking by these devices. We design IoTEDef to be *anomaly-based* because anomaly detection is able to detect unknown patterns [10] and is effective for IoT networks that have simple communication patterns [18]. Building on the concept of a *cyber kill chain*, which is a framework for understanding multi-step attacks [13, 20, 43], IoTEDef uses several detectors – one for each step, and detects abnormal traffic based on results from these detectors. IoTEDef focuses on the steps of a kill chain where networking communication is involved and models the attack's structure. We refer to the steps executed by the attacker to gain a foothold in the targeted system as *early* stages and the other steps as *later* stages. With such knowledge, IoTEDef backward traverses the log of the events upon detecting anomalies related to a later stage of the kill chain, and analyzes these events to identify infection events. IoTEDef updates the system based on the identified events to improve the performance of its infection detector. Essentially, IoTEDef gives up precise detection of unknown patterns of an early stage attack when IoTEDef faces an unknown attack for the first time. Instead, after IoTEDef recognizes that there has been an early stage attack through known patterns of the later stage attack, IoTEDef identifies the corresponding

early stage attack patterns and makes its early detector learn the patterns. Then, IoTEDef can detect such an early stage attack with high precision later on.

Implementing such a strategy is, however, challenging as IoTEDef needs to correlate adversarial and separate events in two different steps (e.g., UDP flooding in the action step and dictionary attack packets in the infection step), where the interval between them can be long. Solving the problem requires mapping diverse networking patterns into kill chain steps and backtracking from later events to earlier events. We note that similar challenges also exist in the area of language translation, and many techniques have been developed to address these challenges [5]. Therefore, we model the problem as a language translation problem by introducing a novel *probability-based embedding* to encode past events into steps that the events belong to and a *attention-based infection identification algorithm* to correlate the encoded events with long-term dependencies in different steps. The algorithm we use helps identify infection events that lead to action events. Finally, we use the identified infection events to improve the performance of the infection detector.

Our systematic and automated method for early detection and self-evolution is beneficial to organizations that perform threat hunting [42]. According to a survey [7], many organizations value threat hunting as it is helpful for early detection and faster repair of vulnerabilities. However, 88% of the respondents say their current systems for threat hunting are immature in terms of formal processes and automation. It shows the value of IoTEDef since it meets such requirements.

To summarize, we make the following contributions:

- We propose IoTEDef, an NIDS that prevents persistent attacks in IoT environments at an early stage.
- We design an attention mechanism-based algorithm to identify infection events from past events.
- We implement a proof-of-concept of IoTEDef and release it on a publicly available repository.
- We carry out comprehensive experiments to assess the accuracy of IoTEDef. Our results show that our approach is feasible and effective.

2 Background

2.1 Cyber Kill Chains

The term 'kill chain' has been coined in the military domain to describe the steps an attacker should complete one by one to achieve its malicious objectives. There are several frameworks [13, 20, 43] proposed to apply this concept to the area of information security. These frameworks understand the structure of attacks as many attacks are multi-steps. Although the number and the name of steps vary, these kill chains commonly break down an attack into the following five steps:

- **Reconnaissance:** the attacker collects information about the target. The attacker may perform social engineering or port scanning.

- **Infection:** the attacker exploits vulnerabilities of the target to take over it and installs malware binaries required to launch attacks. The attacker may launch dictionary attacks or zero-day attacks for this purpose.
- **Lateral movement:** once the attacker has access to the target, the attacker may move laterally to other devices to gain more leverage. The attacker may perform scanning activities or propagate malware in the internal domain.
- **Obfuscation:** the attacker hides its tracks. This step may involve laying false trails and clearing logs.
- **Action:** the attacker launches the attack. For example, in botnets, the attacker directs bots to perform a DDoS attack such as UDP flooding.

We focus on the three steps – *Reconnaissance*, *Infection*, and *Action* – in our NIDS to model the multi-step attacks. The reason is that NIDS can issue the alarms for the reconnaissance and infection steps with the higher priority [32] and can also detect networking attacks, such as DDoS, of the action step.

2.2 Attention Mechanism

The goal of the attention mechanism [3, 30] in deep learning is to pay greater attention to certain factors when classifying data. It was introduced in the natural language processing field to address the problem of *long-term dependencies* in the sequence-to-sequence model that consists of the encoder and the decoder [44]. The encoder compresses a sentence in one language into a fixed-length vector (called a *context vector*). With the vector, the decoder generates a sentence in the other language. Both the encoder and the decoder are recurrent neural networks (RNNs) with long short-term memory (LSTM) units, and the final hidden state of the encoder is provided to the decoder as a context vector. However, Cho *et al.* [6] have shown that the performance of the model degrades as the length of the sentences increases, which is called the bottleneck problem. It is mainly because information loss occurs in the context vector due to its fixed size. Specifically, it often cannot capture interdependence between words far apart in sentences.

With the attention mechanism, the decoder not only refers to the final hidden state of the encoder but also checks all the hidden states of the encoder. In generating a translated sentence, the decoder outputs the next word focusing more on certain hidden states related to the decoder's current state. To this end, each hidden state of the encoder is associated with a weight per state of the decoder, called an *attention weight*. It is determined by the *alignment score* that quantifies the amount of attention. The most widely used scoring function is a *dot product* by which the score is obtained by multiplying the hidden states of the encoder with the state of the decoder.

3 Architecture of IoTEDef

This section presents the design of IoTEDef. We first provide the threat model and the main properties, then describe the architecture of IoTEDef.

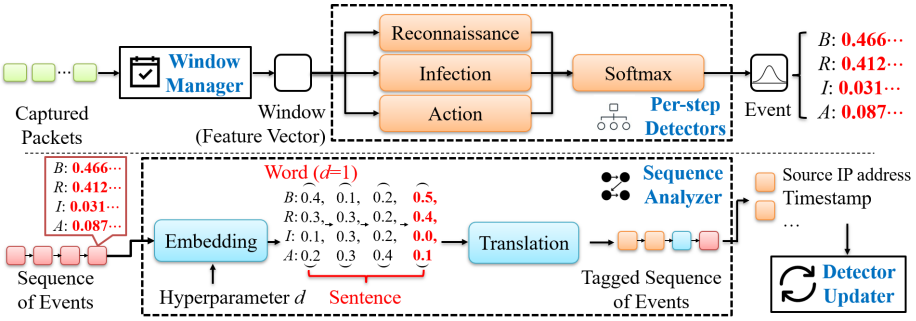

Fig. 1: Architecture of IoTEDef and Infection identification flow.

3.1 Design Principles

Threat model. IoTEDef analyzes network packets exchanged between the network (where it aims to protect) and the Internet. We assume that IoTEDef is not compromised; thus, it does not manipulate the exchanged packets. Also, we assume that an attack is always initiated from the Internet by using remote network access. IoT devices when initially deployed in the network are not compromised; however, they can be compromised later on.

Main properties. IoTEDef is designed to adhere to the following properties:

- **Network-based:** it works with network packets; thus, it does not impose any computation overhead on IoT devices, and is immediately deployable as it does not require any change on IoT devices.
- **Anomaly-based:** it is able to detect unknown patterns and it is also appropriate for the simple communication behavior of IoT devices.
- **Kill chain-based:** it understands multi-step attacks based on a kill chain and deploys classifiers specialized for the steps.
- **Infection-identifying:** it backtracks past events to identify infection events when it detects known events of later stages.
- **Self-evolving:** it updates the infection detector with the identified events.

3.2 Overview

IoTEDef consists of four main components: *window manager*, *per-step detectors*, *sequence analyzer*, and *detector updater* (see Figure 1)

Window manager (packets → window). IoTEDef works on a *flow-based window* where a *flow* is defined as a 5-tuple – the protocol in use, source/destination IP addresses, and source/destination ports. A window manager collects packets per flow and runs a sliding window based on two parameters – a *window output period* and a *window length*. On every window output period, the window manager outputs a *window* in the form of a vector that consists of the 84 flow feature values considered in CICFlowMeter [28], a network traffic flow analyzer. The flow feature values are evaluated from packets within a window length. For instance, let a window output period be 2 and a window length be 5. When a window is

output at time $t = 5$, the window contains the flow feature values from packets captured between $t = 0$ and $t = 5$. The next window is output at $t = 7$ with the values from packets captured between $t = 2$ and $t = 7$.

Per-step detectors (window → event). The main purpose of this component is to map a window to one or more kill chain steps. To this end, IoTEDef has three *per-step detectors* - one for each of the *Reconnaissance*, *Infection*, and *Action* steps, by which NIDS can detect anomalies. They are called the reconnaissance detector, the infection detector, and the action detector, respectively. Each detector has its classifier learned from networking patterns of the corresponding step. Once a window is given to IoTEDef, each per-step detector takes it as input and determines if it contains any anomalous pattern for the corresponding step. If so, IoTEDef labels the window with the name of the corresponding step. For example, we call a given window a reconnaissance window if the reconnaissance detector detects anomalies from the window. This process provides a precedence relation between windows according to the kill chain steps.

A window may belong to multiple steps. For example, the window can be classified as *Reconnaissance* and *Infection* by the reconnaissance and the infection detectors. We call such a window both a reconnaissance window and an infection window. As the results of per-step detectors can be false positives, we make our per-step detectors return confidence scores as well as the results. IoTEDef applies the softmax function to normalize the confidence scores from per-step detectors, resulting in a probability distribution. The probability distribution is used to correct false positives by the infection identification algorithm. We call an output of per-step detectors an *event* that contains a window, three labels (i.e., whether the window belongs to each step respectively), and four probabilities (i.e., normalized confidence scores).

Sequence analyzer (sequence of events → identified infection events). This module runs the infection identification algorithm to find the infection events that lead to the action event. The algorithm takes a sequence of past events, each of which has a probability distribution assigned by per-step detectors. Then, the algorithm analyzes the sequence and determines only one kill chain step for each event according to the entire context. To this end, we develop an identification algorithm based on the attention mechanism in deep learning techniques, which considers all the (hidden) states when producing the next state. Finally, the algorithm returns the infection events from the resulting sequence.

Detector updater. (identified infection events → updated infection detector). The detector updater is responsible for updating the classifier of the infection detector. The module labels the identified infection events as *Infection* and re-trains a new classifier with the training set and the events.

4 Detail of IoTEDef

This section provides IoTEDef in detail. We first formally define the problem to be solved and present our solution with the probability-based embedding and the attention-based translation. Finally, we discuss our update strategies.

4.1 Problem Definition

We begin with the notions of a tag and an event. Our formal definitions include notation $a.b$ to indicate an attribute b of a.

Definition 1 (**tag**). A set $\mathcal{L} = \{B, R, I, A\}$ is a collection of tags that an event can be labeled with, where B, R, I, A denote *Benign*, *Reconnaissance*, *Infection*, and *Action*, respectively.

Definition 2 (**event**). An event $e = (w, l, p, t)$ is a tuple of four attributes. $e.w$ is a window. $e.l = (r, i, a)$ is a tuple of three attributes where $e.l.r, e.l.i, e.l.a \in \{0, 1\}$ and each indicates whether the window $(e.w)$ is labeled by each step detector. $e.p = (b, r, i, a)$ is a tuple of four attributes $(0 \leq e.p.b, e.p.r, e.p.i, e.p.a \leq 1, e.p.b + e.p.r + e.p.i + e.p.a = 1)$, which are the probabilities of the window belonging to the class *Benign*, *Reconnaissance*, *Infection*, and *Action*, respectively. $e.t \in \mathcal{L}$ is a tag that will be finally assigned by the identification algorithm.

In what follows, $\mathcal{E}$ denotes the set of events. Recall that our goal is to identify infection events by backtracking the past events (or an input sequence of events) based on anomalies in the action step. We model the process of backtracking as an event tagging problem described as follows:

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

While solving the problem, we face the challenge of long-term dependency between events. As an example, an infection event always precedes an action event, but the time interval between those two events can be long. For instance, in Mirai [2], a device is infected by an attacker's dictionary attack (an infection step). Then, the device launches the UDP flooding attack (an action step), possibly a long time after the infection event. In this example, the role of IoTEDef is to identify events that contain dictionary attack patterns when IoTEDef detects a UDP flooding pattern. Therefore, IoTEDef should be able to correlate distant events.

To address such an issue, we review language translation techniques in natural language processing as even words that appear far apart can have a significant relationship in natural language. There have been many techniques to capture such dependencies [5]. Our idea is that if we are able to model an event as a word in a language and a sequence as a sentence in a language, we can achieve the goal by using language translation techniques. To this end, we should resolve the two challenges: 1) *how to model an event as a word in the language* and 2) *what technique we use to translate an input sequence to an output sequence.*

Modeling an event as a word in a language requires defining the word sets of input and output sequences. For an output word set, we can use $\mathcal{L}$. However, it is challenging to define an input word set to represent each in a sequence of events.

One important requirement is that our encoding scheme should be able to model long-term dependency. For example, using the one-hot encoding is inappropriate in our scheme since it cannot capture the informative relations between different categorical variables [16] (see Subsection 5.2).

Another challenge is what technique to use for translating an input sequence to an output sequence. There are traditional sequence labeling or decoding algorithms from observations, such as an episode-tree-based model [41] or the Viterbi algorithm [11]. However, they are ineffective as they cannot model long-term dependency (see Subsection 5.4).

Therefore, we divide Problem 1 into the following two subproblems.

Subproblem 1 (embedding). Let $\mathbf{e} = \{e_1, e_2, \cdots, e_n\}$ be an input sequence of length n that belongs to $\mathcal{E}^*$ and let $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$ be an *input sentence* that belongs to $\mathcal{X}^*$, where $\mathcal{X}^*$ is a set of all sentences over an input word set $\mathcal{X}$. Our goal is to define an input word set $\mathcal{X}$ and an embedding function $e : \mathcal{E} \to \mathcal{X}$ that takes an event e as an input and outputs an input word x to finally convert $\mathbf{e}$ to $\mathbf{x}$.

Subproblem 2 (translation). Let $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$ be an input sentence that belongs to $\mathcal{X}^*$ and let $\mathbf{y} = \{y_1, y_2, \cdots, y_n\}$ be an output sentence that belongs to $\mathcal{L}^*$. Our goal is to design a translation function $t : \mathcal{X}^* \to \mathcal{L}^*$ that takes $\mathbf{x}$ as an input, tags all the embedded windows in $\mathbf{x}$, and outputs $\mathbf{y} = \{y_1, y_2, \cdots, y_n\}$.

Our solution to the problem consists of the following three steps (see Figure 1):
1. Probability distribution assignment: an event is assigned a probability distribution ($e.p$) that shows how much a window ($e.w$) belongs to a class.
2. Probability-based embedding: the probability distribution of an event is encoded into a word in a language.
3. Attention-based translation. A series of embedded words are translated into tagged ones each of which has only one label.

4.2 Probability Distribution Assignment

Per-step detectors are responsible for this step. Recall that the per-step detectors detect reconnaissance, infection, and action patterns in a given window. Each detector has its own classifier trained with packets of the corresponding step and detects anomalies according to the classification results. For example, the classifier of the infection detector is generated from the network patterns of the telnet dictionary attack, the Log4j attack, or other attacks aiming to infect IoT devices. For a classifier of each per-step detector, any algorithm can be used. However, the performance of IoTEDef depends on the characteristics of the classification algorithm as the infection identification algorithm runs over the detection result. Our result shows that the class of recurrent neural networks (RNN) with long short-term memory (LSTM) units is most effective (Subsection 5.3).

Each detector evaluates the probability of the window belonging to its corresponding step. For instance, the reconnaissance detector may label the window as *Reconnaissance* with a probability of 0.68 and the infection detector may classify

Algorithm 1 Probability-based Embedding e

Input: Sequence of events $\mathbf{s} = (e_1, \cdots, e_n)$ and d
Output: Sequence of words (Sentence) $\mathbf{x} = (x_1, \cdots, x_n)$
Initialize: $\mathbf{x}[:] = []$
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
  ▷ $\mathtt{r}(a, b)$: the result of rounding up/down/off a to b of decimal places
  $x_k = (\mathtt{r}(e_k.p.b, d), \mathtt{r}(e_k.p.r, d), \mathtt{r}(e_k.p.i, d), \mathtt{r}(e_k.p.a, d))$
  Add a word x_k to Sequence $\mathbf{x}$
end for

the window as *Infection* with a probability of 0.53. We convert the probabilities into one probability distribution using the softmax function and finally output the distribution as an event. As an example, an event may be associated with a probability distribution $(0.46, 0.31, 0.11, 0.12)$, which means that the probability that the corresponding window belongs to *Benign* is 0.46, *Reconnaissance* is 0.31, *Infection* is 0.11, and *Action* is 0.12.

4.3 Probability-based Embedding

To solve Subproblem 1, we design a novel probability-based embedding algorithm (see Algorithm 1). We represent input words as a vector of four probabilities (for each step) - the sum of which is 1.

Hyperparameter *d*. One issue is that the above 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 hyperparameter d, which is the number of decimal places to round off the probabilities. Given $d \in \mathbb{N}$, let $\mathcal{P}$ be $\{p | p = \mathsf{round}(q, d), 0 \leq q \leq 1\}$, where $\mathsf{round}(a, b)$ is a function that rounds off a to b of decimal places. With d, the input set is changed to $\mathcal{X} = \{(b, r, i, a) | b, r, i, a \in \mathcal{P} \text{ and } b + r + i + a = 1\}$. However, rounding off does not guarantee that the sum of the rounded probabilities is always one. To avoid the case that the sum is not one, 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 the sum of the resulting probabilities is one. For example, $(0.466\cdots, 0.412\cdots, 0.031\cdots, 0.087\cdots)$ became $(0.5, 0.4, 0.0, 0.1)$ when $d = 1$ (see bolded numbers in Figure 1). Note that d determines the number of input words in the word set.

Algorithm 2 Attention-based Translation t

Input: Sequence $s_i = (e_1, \cdots, e_n)$, Decimal Place d
Output: Sequence s_o
Initialize: $s_o[:] = 0$
1: $s_e = \texttt{ProbabilityBasedEmbedding}(s_i, d)$
2: $r_{lstm} = \texttt{LSTM}(s_e)$
3: $r_{attention} = \texttt{Attention}(r_{lstm})$
4: $r_{ff} = \texttt{Feedforward}(r_{attention})$
5: $r_{sm} = \texttt{Softmax}(r_{ff})$
6: $\triangleright\ r_{sm} = (p_{b_1}, p_{r_1}, p_{i_1}, p_{a_1}), \cdots, (p_{b_n}, p_{r_n}, p_{i_n}, p_{a_n})$
7: **for** $k = 1, 2, \ldots, n$ **do**
8: $\quad e_k.t = \texttt{argmax}((p_{b_k}, p_{r_k}, p_{i_k}, p_{a_k}))$
9: $\quad$ Add e_k to sequence s_o
10: **end for**

4.4 Attention-based Translation

To solve Subproblem 2, we apply the attention mechanism to our neural network classifier that has been introduced to address the long-term dependency problem in language translation. The flow of infection identification with the multi-classification neural network is as follows (see Algorithm 2):

- **Input:** an input sequence is a series of events, each of which has a probability distribution assigned by the per-step detectors.
- **Embedding:** the events in an input sequence are converted to a sentence of input words, each of which consists of four probabilities.
- **Long short term memory (LSTM):** after the embedding, a sequence of input words passes through an LSTM layer. The layer outputs vectors considering the context of each input word.
- **Attention:** we add an attention layer after the LSTM layer. It calculates correlation scores between events and assigns attention weights.
- **Feedforward & Softmax:** we add a feedforward layer and a softmax layer. They output four probabilities for each input word. Each probability represents the degree that an input word is translated into an output word.
- **Output:** finally, each input word in the input sequence is translated into the output word with the highest probability.

With the attention mechanism, IoTEDef 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 with respect to 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.

4.5 Self-Evolving Strategies

The detector updater updates the classifier of the infection detector with the identified infection events. We consider three different strategies. Let $\mathcal{A}$ be a set of events having the highest probability of *Infection* after passing through the per-step detectors and let $\mathcal{B}$ be a set of infection events tagged by the attention-based translation. We denote the difference between the two sets by $\mathcal{C}$ (i.e., $\mathcal{A} \setminus \mathcal{B}$).

- **Strategy 1:** a new infection classifier is generated over a training set, all the events in $\mathcal{B}$ as *Infection*, and all the events in $\mathcal{C}$ as *Benign*. For example, let say there are 5 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 algorithm provides the information that e_1, e_2, e_4 ($\mathcal{B}$) are infection events (i.e., $e_1.t = e_2.t = e_4.t = I$). Then, IoTEDef updates the classifier with $e_1.w, e_2.w, e_4.w$ labeled as *Infection* and $e_3.w, e_5.w$ labeled as *Benign*.
- **Strategy 2:** this strategy is similar to the first one except that it only uses a training set and all the events in $\mathcal{B}$ (not the events in $\mathcal{C}$). It adds information about the true positives to the updated model. In our example, $e_1.w, e_2.w, e_4.w$ labeled as *Infection* are used to update the model.
- **Strategy 3:** this strategy is similar to the first one, but it uses a training set and all the events in $\mathcal{C}$ (not the events in $\mathcal{B}$). It aims to reduce the false positives of the updated model. In our example, $e_3.w, e_5.w$ labeled as *Benign* are used to update the model.

5 Evaluation

This section provides an experimental analysis of IoTEDef. We implement a proof-of-concept prototype and build a testbed for evaluation using a dataset related to the Mirai botnet [2] and the Log4j attack [36]. We release the dataset, the scripts, and the implementation source codes at https://github.com/iotedef.

5.1 Experimental Setting

Implementation. We use the pcapy library [8] to capture packets and the *keras* library [24] to implement neural networks and other machine learning-related functions. 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 loss function.

Datasets. We generate datasets considering the following two scenarios.

- **Mirai botnet campaign [2]:** it includes the telnet dictionary attack as an infection activity. We use a publicly available IoT intrusion dataset from academia [23]. It contains captured packets from the real-world and consists of diverse types of packets including benign packets, port scanning packets,

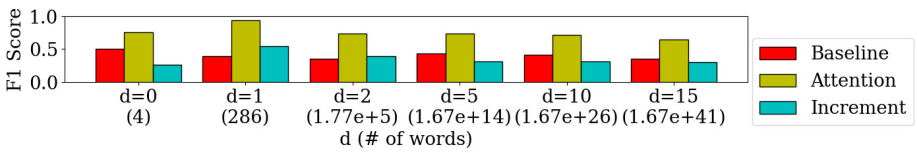

Fig. 2: Impact of the probability-based embedding.

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.

– **Log4j attack [36]:** it includes the Log4j attack as an adversary's infection activity. We build our testbed based on Mininet [27], 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 tell.

We generate several datasets per scenario, each of which has a different number of packets corresponding to the step and time intervals between different events. The detail of the dataset generation algorithm is described in Section A.

Testbed. We perform our experiments in one machine with i7-4700 CPU @ 3.60GHz 8 core processors and 16GB RAM. To evaluate the performance of IoTEDef with the practical scenarios, we replay the packets from the above dataset with *Tcpreplay* [25]. The generated packets are captured by IoTEDef.

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 with the infection detector learned only with the training set.
– **Attention:** We evaluate how well our attention-based infection identification algorithm (Attention) works over a sequence of events.
– **Update:** We measure the performance of the infection detector evolved with the identified infection events on a different test set.

5.2 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 (see Figure 2). Overall, the F1-score increases from 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 is. For $d > 1$, the number of elements is higher than 10^5, which we believe is too large to map 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.

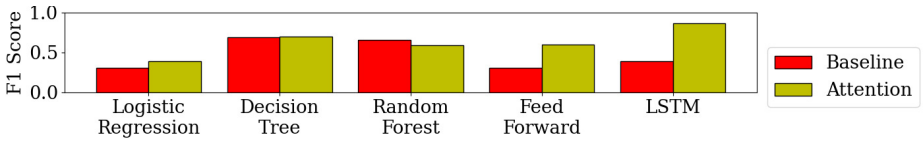

Fig. 3: Compatibility with the detector classifiers.[1]

5.3 Impact of Classifiers of Per-step Detectors

As Attention relies on probabilities assigned by per-step detectors (see Figure 1), we experiment 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. We evaluate each classifier with and without Attention and calculate the F1-score.

We find that the neural networks are compatible with Attention (see Figure 3). The increments of the F1-score for both neural network algorithms are 0.29 (Feedforward) and 0.48 (LSTM), respectively, while for other algorithms 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 to find false positive results in the detectors.

The reason why non-neural network algorithms show worse performance is because of their assumption. Logistic regression shows poor performance due to its linear boundary assumption. Decision tree shows high precision with low recall, which means it is over-fitted. Furthermore, the difference in F1-score between Decision tree with and without Attention is only 0.01. It is because Decision tree does not produce a probability and thus is not compatible with our embedding scheme. Also, 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 Random forest, we find that 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.

5.4 Other Identification Algorithms

We compare Attention with other traditional algorithms for sequences. We consider the following three different 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*,

[1] Note that "LSTM" in the figure is a classifier in the infection detector, not an encoder layer before the attention layer in Attention.

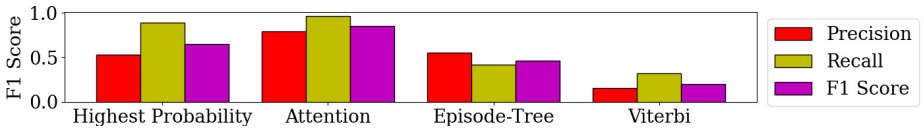

Fig. 4: Comparison with other algorithms.

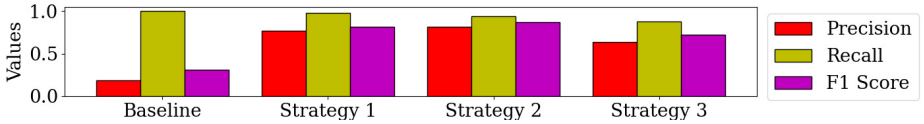

Fig. 5: Self-evolving strategies.

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.

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.* [31]. EPISODE-TREE identifies infection windows if a given window sequence matches a branch of the episode-tree, which contains infection windows.

The result (see Figure 4) shows that the attention-based algorithm outperforms the other algorithms. The F1-score of the attention-based algorithm is 0.85. The performance of EPISODE-TREE (0.46) and VITERBI (0.20) are even worse than HIGHEST PROBABILITY (0.65). EPISODE-TREE is a simple pattern matching algorithm; thus, it depends on how many patterns are captured from the training set. Therefore, 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 that makes the algorithm unable to capture long-term dependencies.

5.5 Self-Evolving Strategies

We carry out an experiment to assess the impact of the three strategies discussed in Section 4. We compare the performance of the baseline with the performance of the updated model according to the strategies. STRATEGY 1 is the strategy that uses both identified infection and benign events, STRATEGY 2 is the strategy that only uses identified infection events, and STRATEGY 3 is the strategy that only uses benign events.

Our results (see Figure 5) show that all the updated models outperform the baseline regardless of the strategy. Compared with other strategies, STRATEGY 2 works best (an 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-evolving strategy affects the performance of the models. The results show that many anomalous samples are classified as *Benign* under STRATEGY 1 and STRATEGY 3, resulting in worse performance compared to STRATEGY 2.

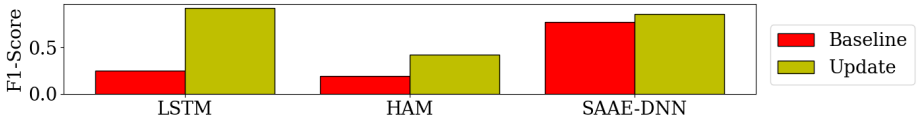

Fig. 6: Comparison with attention-based NIDSes.

5.6 Comparison with Attention-based NIDSes

Although our approach is orthogonal to the detection algorithm, we compare Attention with existing attention-based NIDSes with respect to two aspects. First, we see if the performance of LSTM after being evolved is comparable to the performance of those IDSes. Second, we assess whether Attention is also beneficial to them. In our analysis, the following two systems are considered:

Hierarchical Attention Model (HAM). HAM [29] is based on two attention layers, namely 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 [45] 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 experiment, 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-evolving Strategy 2 to update the models based on the result on set_1. We refer to the updated 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 (see Figure 6).

Our conclusions are as follows. First, Updated LSTM outperforms HAM and SAAE-DNN. The F1-score of the original LSTM 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 became the highest (0.92) after being evolved. It shows that our self-evolving strategy can make the performance of the classifier comparable to existing systems. Second, existing NIDSes can benefit from our approach. After being evolved, the F1-scores of HAM and SAAE-DNN increases from 0.19 to 0.42 and from 0.77 to 0.86 respectively.

6 Related Work

Network intrusion detection systems for IoT. Kalis by Midi *et al.* [34] is a self-adapting, knowledge-driven IDS system. It collects knowledge about the network's features autonomously and selects relevant detection techniques. Fu *et al.* [12] designed an IDS 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 IDS raises the alarm. DÏoT by Nguyen *et al.* [37] is a federated self-learning anomaly detection system. It builds on device-type-specific communication profiles and raises an alarm upon detecting deviations with respect to 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 the above systems, the focus of IoTEDef is to identify infection events from an attacker's actions, which is orthogonal to the goals of those systems.

Multi-step attack detection. Gu *et al.* [15] present BotHunter that detects successful malware infection by tracking communication flows between internal assets and external entities, and applying their dialog-based correlation. Haas and Fischer *et al.* [17] propose GAC, a graph-based approach for correlation. They apply the graph-based clustering algorithm to the alarms to cluster them 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. Sadegh *et al.* [35] suggest Holmes that models the attacks with a kill chain. From audit logs, they generate a provenance graph, find adversarial activities based on predefined rules, and map the activities to the corresponding kill chain step. Xueyuan *et al.* [19] propose Unicorn that detects the APT attacks by leveraging provenance graphs to detect anomalies with no prior knowledge of the APT attack patterns by using the clustering approach.

Our work differs from those approaches in three aspects: (1) *Goals:* they use correlation algorithms to *automatically detect* multi-step attacks. On the other hand, our approach aims to *automatically identify* the infection vectors and *update* the IDS after seeing anomalies in the action step (or other steps). (2) *Logs:* although one can identify infection windows using their correlation algorithms, such approaches mostly rely on host events (e.g., process-related events). Applying such approaches would require extensions of IoT devices, which we want to avoid. (3) *Used techniques:* the above approaches rely on graphs to analyze the causality between events. Unlike them, IoTEDef uses the attention mechanism with the neural network to associate the event windows.

NIDS based on the attention mechanism. We have discussed HAM [29] and SAAE-DNN [45] in Subsection 5.6.

7 Conclusion

In this paper, we have introduced IoTEDef, a kill chain-based approach for early detection of persistent attacks against IoT devices. To improve the accuracy of the infection detector, IoTEDef adopts a feedback strategy that backtracks past events to identify infection events when anomalies at the later steps of a kill chain are detected.

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.

Acknowledgement

The work reported in this paper has been supported by Cisco Research and by NSF under grant 2112471.

References

1. Andrea, H.: 10 benefits of internet of things (iot) in our lives and businesses. https://www.tech21century.com/internet-of-things-iot-benefits/ (2021), (Accessed on 09/13/2021)
2. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al.: Understanding the mirai botnet. In: 26th USENIX Security Symposium (2017)
3. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: International Conference on Learning Representations (2015)
4. Bertino, E., Islam, N.: Botnets and internet of things security. IEEE Computer **50**(2), 76–79 (2017)
5. Chaudhari, S., Mithal, V., Polatkan, G., Ramanath, R.: An attentive survey of attention models. ACM Transactions on Intelligent Systems and Technology (TIST) **12**(5), 1–32 (2021)
6. Cho, K., Merriënboer, B.V., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches (2014)
7. Cole, E.: Threat hunting: Open season on the adversary. https://de.malwarebytes.com/pdf/white-papers/SurveyThreat-Hunting-2016Malwarebytes.pdf (2016), (Accessed on 1/31/2022)
8. CoreSecurity: Pcapy (2014), (Accessed on 10/15/2021)
9. Dingee, D.: Iot, not people, now the weakest link in security. https://devops.com/iot-not-people-now-the-weakest-link-in-security/ (January 2019), (Accessed on 05/13/2021)
10. Eskandari, M., Janjua, Z.H., Vecchio, M., Antonelli, F.: Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. IEEE Internet of Things Journal **7**(8), 6882–6897 (2020)
11. Forney, G.D.: The viterbi algorithm. Proceedings of the IEEE **61**(3), 268–278 (1973)
12. Fu, Y., Yan, Z., Cao, J., Koné, O., Cao, X.: An automata based intrusion detection method for internet of things. Mobile Information Systems **2017** (2017)
13. Gartner: Addressing the cyber kill chain: Full gartner research report and looking-glass perspectives (2016), (Accessed on 06/03/2021)
14. Glassberg, J.: Jackware: A new type of ransomware could be 10 times as dangerous. https://finance.yahoo.com/news/ransomware-jackware-115229732.html (2021), (Accessed on 06/12/2021)
15. Gu, G., Porras, P.A., Yegneswaran, V., Fong, M.W., Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: USENIX Security Symposium. vol. 7, pp. 1–16 (2007)
16. Guo, C., Berkhahn, F.: Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737 (2016)
17. Haas, S., Fischer, M.: Gac: graph-based alert correlation for the detection of distributed multi-step attacks. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 979–988 (2018)

18. Habibi, J., Midi, D., Mudgerikar, A., Bertino, E.: Heimdall: Mitigating the internet of insecure things. IEEE Internet of Things Journal **4**(4), 968–978 (2017)
19. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Unicorn: Runtime provenance-based detector for advanced persistent threats. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2020)
20. Hutchins, E.M., Cloppert, M.J., Amin, R.M., et al.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research **1**(1), 80 (2011)
21. Jallad, K.A., Aljnidi, M., Desouki, M.S.: Anomaly detection optimization using big data and deep learning to reduce false-positive. Journal of Big Data **7**(1) (2020)
22. Javed, M., Paxson, V.: Detecting stealthy, distributed ssh brute-forcing. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 85–96 (2013)
23. Kang, H., Ahn, D., Lee, G., Yoo, J., Park, K., Kim, H.: Iot network intrusion dataset. https://ieee-dataport.org/open-access/iot-network-intrusion-dataset (2019), (Accessed on 06/03/2021)
24. Keras: Keras. https://keras.io/ (2016), (Accessed on 10/15/2021)
25. Klassen, F., AppNeta: Tcpreplay. https://tcpreplay.appneta.com/ (2018), (Accessed on 06/03/2021)
26. Krebs, B.: Reaper: Calm before the iot security storm? https://krebsonsecurity.com/2017/10/reaper-calm-before-the-iot-security-storm/ (October 2017), (Accessed on 05/07/2021)
27. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. pp. 1–6 (2010)
28. Lashkari, A.H.: Cicflowmeter features. https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt (2018), (Accessed on 05/19/2022)
29. Liu, C., Liu, Y., Yan, Y., Wang, J.: An intrusion detection model with hierarchical attention mechanism. IEEE Access **8**, 67542–67554 (2020)
30. Luong, M.T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. In: The 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015) (2015)
31. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data mining and knowledge discovery **1**(3), 259–289 (1997)
32. Martin, L.: Seven ways to apply the cyber kill chain with a threat intelligence platform. lockheed martin corporation (2015) (2015)
33. McMillen, D., Alvarez, M.: Mirai iot botnet: Mining for bitcoins? https://securityintelligence.com/mirai-iot-botnet-mining-for-bitcoins/ (April 2017), (Accessed on 05/07/2021)
34. Midi, D., Rullo, A., Mudgerikar, A., Bertino, E.: Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 656–666. IEEE (2017)
35. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: Holmes: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE Symposium on Security and Privacy (S&P). pp. 1137–1152. IEEE (2019)
36. Msehgal: Protect your iot devices from log4j 2 vulnerability. https://live.paloaltonetworks.com/t5/blogs/protect-your-iot-devices-from-log4j-2-vulnerability/ba-p/453381 (2021), (Accessed on 1/14/2022)

37. Nguyen, T.D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., Sadeghi, A.R.: Dïot: A federated self-learning anomaly detection system for iot. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 756–767. IEEE (2019)
38. Osborne, C.: This is why the mozi botnet will linger on. https://www.zdnet.com/article/this-is-why-the-mozi-botnet-will-linger-on/ (2021), (Accessed on 1/27/2022)
39. Palmer, D.: This sneaky hacking group hid inside networks for 18 months without being detected. https://www.zdnet.com/article/this-sneaky-hacking-group-hid-inside-networks-for-18-months- without-being-detected/ (2022), (Accessed on 5/18/2022)
40. Research, C.P.: Iotroop botnet: The full investigation. https://research.checkpoint.com/2017/iotroop-botnet-full-investigation/ (March 2017), (Accessed on 05/07/2021)
41. Soleimani, M., Ghorbani, A.A.: Multi-layer episode filtering for the multi-step attack detection. Computer Communications **35**(11), 1368–1379 (2012)
42. Sqrrl Data, I.: A framework for cyber threat hunting. https://www.threathunting.net/files/framework-for-threat-hunting-whitepaper.pdf (2018), (Accessed on 1/31/2022)
43. Storm, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B.: Mitre att&ck: Design and philosophy (2018), (Accessed on 06/03/2021)
44. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. pp. 3104 – 3112 (2014)
45. Tang, C., Luktarhan, N., Zhao, Y.: Saae-dnn: Deep learning method on intrusion detection. Symmetry **12**(10), 1695 (2020)

A Dataset Generation

In our experiment, we use the dataset from [23]. 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 experiment:

- **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 predefined credentials to infiltrate into a target device. We label these packets as *Infection*.
- **Flooding:** these packets are SYN/ACK/HTTP/UDP flooding packets to cause a DoS condition to a victim. These packets are tagged as *Action*.

Due to the limited number of datasets, we manipulate the existing dataset to create new diverse scenarios. For example, we want to generate a dataset with a specified number of infection packets at a certain time and a number of 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 a number of 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 the 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.

This way, we can flexibly generate a new dataset. The dataset generation script is available at https://github.com/iotedef.