Hindawi
Security and Communication Networks
Volume 2023, Article ID 6802359, 15 pages
https://doi.org/10.1155/2023/6802359

Research Article

SR2APT: A Detection and Strategic Alert Response Model against Multistage APT Attacks

Fan Shen,[1] **Levi Perigo**,[1] **and James H. Curry**[1,2]

[1]*Department of Computer Science, University of Colorado Boulder, Boulder, CO 80309, USA*
[2]*Department of Applied Mathematics, University of Colorado Boulder, Boulder, CO 80309, USA*

Correspondence should be addressed to Fan Shen; fan.shen@colorado.edu

Received 21 July 2022; Revised 9 October 2022; Accepted 11 October 2022; Published 19 April 2023

Academic Editor: Yuanyuan Huang

Copyright © 2023 Fan Shen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Advanced persistent threats are an emerging cyber threat to cyber-physical systems (CPS), especially those comprising mission-critical physical assets. However, defense against such attacks is challenging, due to their sophistication, stealthiness, and zero-day exploitation. Existing works in this area mainly focus on the detection of APT, but it might be too late or too costly to impede APT when it is detected with high confidence. Therefore, this work focuses on CPS intrusion detection and prevention against APT attacks and aims at preventing such attacks in earlier stages through a strategic response policy to imperfect APT alerts by leveraging the multistage characteristic of APT and a deep reinforcement learning formulation. A novel host-based APT detection and response model called SR2APT is proposed, which consists of a detection engine and a decision engine. The detection engine is based on graph convolutional network, which classifies a stream of system log provenance subgraphs as an APT stage or benign. Then, the detection results are transmitted to the decision engine sequentially, which is trained based on deep reinforcement learning and outputs the optimal response actions to APT alerts. Experimental results show that the GCN-based detection engine obtains 94% classification accuracy on a semisynthetic dataset of system logs and outperforms classification models based on SVM, CNN, and LSTM. The strategic alert response policy from the decision engine is compared with two baseline fixed response policies, and it achieves the best trade-off between preventing APT attacks and minimizing the impediments of mistaken active defense actions to benign activities that generate false alerts, thus obtaining the highest total rewards in the defense against APT attacks.

1. Introduction

Cyber-physical systems (CPS) integrate software components with physical processes, which provide monitoring and feedback loops to maintain and improve the performance of physical processes [1]. As the physical processes in CPS are usually mission-critical, such as the power grid or programmable logic controllers, these systems have been targeted by advanced persistent threats (APT) [2]. APT is a type of sophisticated cyberattack [3, 4], which aims at stealing valuable and possibly confidential data or causing the malfunction of the mission-critical infrastructure of the victim without being detected. Initially, the targets of APT are government or military sectors for espionage, but over recent years, the APT battlefield has extended to the IT infrastructure of high-profile companies [5], IoT networks [6], and public cloud platforms [7] for sabotage purpose or financial gains. Considering the devastating outcome of APT, any modern organizations should be aware of it and have protection mechanisms in place to defend against it, and the leveraging unique traits of sophisticated attacks like APT for accurate and agile detection is a focus of CPS intrusion detection system design [8].

Defense against APT presents an evolving challenge due to the unique characteristics of such attacks. Different from general computer viruses or worms that intend to cause widespread and indiscriminate damage, APT attacks focus on specific targets, e.g., a server storing sensitive data or a mission-critical infrastructure component, just like the programmable logic controllers (PLCs) compromised in

Stuxnet [3], therefore, attack vectors are usually well planned and customized for the target. In order to access the target without being detected, APT attackers move low and slow to maintain persistence in the victim's network, and these long-term campaigns span multiple stages, such as reconnaissance, weaponization, delivery (of malicious code), exploitation, installation (of malware), command and control communication, and action on objective [9]. Most importantly, APT is advanced as these well-resourced attackers are capable to exploit zero-day vulnerabilities. The attackers in Stuxnet, believed to be responsible for the malfunction of centrifuges in Iran's nuclear project, exploited more than four zero-day vulnerabilities [10].

Existing studies on APT detection can be categorized as either point detection, focusing on special behaviors or specific stages of APT attackers, or contextual detection combining the footprints of different APT stages. In terms of point detection, network packets and flows, especially those related to http(s) traffic, are analyzed in [11–13] to identify the command and control (C2) communication, which is commonly used by APT malware to receive instructions from or exfiltrate data to malicious servers. E-mail features are extracted to detect spear phishing e-mails used by APT attackers to deliver malicious PDF files or URLs to the victim's system [14, 15]. CPU and memory utilization, the content of the Windows Registry, and the System32 directory are also useful sources to capture the footprints of APT malware [16, 17]. Host-level system logs are analyzed in [18] to map a sequence of system logs to an APT stage. However, point detection methods may cause lots of false positives when anomalous behaviors can result from both APT and benign activities [19]. To reduce false positives, contextual detection methods correlate detection results of multiple APT stages. An attack pyramid model is proposed in [20] providing a thorough conceptual framework to link APT alerts of different stages across various planes. An attack chain model is used in [19] to describe the APT life cycle, and anomalous events from different APT stages are combined using statistical methods to assign scores to hosts, indicating the level of compromise by APT. A bottom-up approach is proposed in [21] to map low-level system logs to TTPs of APT defined by MITRE ATT & CK using predefined rules and then build a chain of TTPs based on node relations on the provenance graph and apply scoring rules to differentiate APT scenarios and benign scenarios.

This work focuses on the defense against a multistage APT attack on a host and considers the threat model in which the behavioral pattern of each APT stage can appear in both attack and benign scenarios. Therefore, it is insufficient to claim the existence of APT by the detection of a single APT stage. Existing works that tackle similar APT scenarios [19, 21] using contextual detection methods have demonstrated the feasibility to detect multiple APT stages from system logs. The goal of these works is to reduce false positives in APT detection by correlating alerts of multiple APT stages. However, there is no defender's intervention that would affect system activities during the collection of APT alerts, and it can be too late or too expensive to impede APT when it is detected with confidence. In this paper, an APT detection and alert response model called SR2APT is proposed to provide agile and strategic responses to imperfect APT alerts to secure critical hosts in CPS. On the one hand, SR2APT integrates the detection of APT stages and strategic responses to APT alerts such that a multistage APT is prohibited in earlier stages, which is more advantageous than ex post facto remedies when APT has progressed to later and more destructive stages. On the other hand, SR2APT applies a model-free deep reinforcement learning method to solve strategic alert response actions, which does not require assumptions about the strategies of the APT attacker compared with game theoretic studies. The two main components of SR2APT are described below, with more technical details in Section 4.

First, the detection engine in SR2APT is responsible for classifying a provenance subgraph, which is extracted from host-level system logs, as one stage of a multistage APT or a general benign class. A graph convolutional network (GCN) is used for the multiclassification task, which leverages useful features hidden in the structure of a graph. To the authors' knowledge, this is the first work of using GCN to analyze system logs for the detection of APT stages.

Second, the decision engine in SR2APT outputs optimal response actions to alerts from APT stages. It is trained based on deep reinforcement learning, which incorporates the defender's understanding of the threat model, interdependencies between APT stages, costs, and benefits of response actions. Although APT alerts can be triggered by both APT and benign activities in the threat model of this paper, the alert response strategy from the decision engine is able to prevent APT attacks in earlier stages and minimize the impediments to benign activities from mistaken active defense actions. Unlike existing contextual APT detection methods, SR2APT takes the defender's responses into consideration during detection, thus it enables cost-effective proactive defense against multistage APT attacks.

For evaluations, the detection engine of SR2APT is tested on a semisynthetic dataset, which includes host-level system logs about malicious behaviors of six APT stages and benign behaviors. The decision engine of SR2APT is evaluated on synthetic alert traces from both APT and benign scenarios, which are generated according to the interactions between the defender and the attacker. Experimental results show that the GCN-based detection engine achieves 94% classification accuracy and outperforms other kernels including SVM (support vector machine), one-dimensional CNN (convolutional neural network), and LSTM (long short-term memory). The strategic alert response policy from the decision engine achieves the highest total rewards in the defense against APT when compared with two baseline fixed response policies, because it optimally balances the trade-off between maximizing the number of prevented APT attacks and minimizing the impediments of mistaken active defense actions to benign activities that generate false alerts.

The remaining of this paper is outlined as follows. Section 2 reviews related works about defending against APT by the analysis of system logs. Section 3 presents the architecture of the proposed APT detection and alert response model, SR2APT. Section 4 provides technical details

of the main components of SR2APT. Section 5 shows the design and results of the proof of concept experiments. Section 6 concludes this work and suggests directions for future work.

2. Related Work

System logs record essential activities executed by users, applications, or operating systems, serving as an important source for digital forensic analysis [22]. Each entry in system logs is known as an event, including the fields defined by the system log capturing tools, such as the event time, enter process name and ID, exit process name and ID, and event type. A trace of system logs is a sequence of events in the temporal order. Signature-based methods can be used to check if an event is malicious or not. For example, a simple string matching method is applied to system logs for the detection of data exfiltration, which is one of the main goals of APT attacks [23]. In this work, malicious actions such as rename, copy, and move to sensitive files are captured by system logs, and then a policy-based method is used to enforce a set of rules to detect malicious activities and send alert notifications. However, signature-based methods can only detect known attack patterns and require that the rules are well-defined, complete, and updated. Instead of focusing on a single event, occurrence-based methods are used to extract features of system behaviors that relate to multiple events. In [24], a trace of system logs is represented by short and fixed-length contiguous sequences, and then metrics based on Hamming distance are used to decide if a trace of system logs is malicious by comparing its representation with the representation of benign traces. An anomaly detection method based on k-nearest neighbors is proposed to analyze system logs in [25]. In that paper, the occurrences of each type of system calls are collected to form the feature vector of a trace of system logs. For an unknown trace, cosine similarity is applied to find its k-nearest neighbors in the feature space, and then those neighbors' classes are used to determine its class. To consider the ordering of system calls, Subba et al. [26] used the frequencies of unique n-grams to form the feature vector of a trace of system logs, where an n-gram is defined as a contiguous system calls of length. However, when dealing with large-scale real-world system logs, the number of unique n-grams can grow exponentially, and the dimension of the frequency vector increases, thus it becomes intractable to enumerate all patterns.

To efficiently detect APT by analyzing system logs, recent works suggest using provenance graphs or information flow graphs to structure system logs and then either obtaining more meaningful sequences of system logs from provenance graphs or applying graph-based methods to analyze provenance graphs. The advantage of provenance graphs is that system calls are connected based on their causal relations rather than being temporally ordered [27], meaning that two interdependent system calls spanning a long period can be more efficiently connected by provenance graphs than raw system logs. Figure 1 shows a provenance graph built from a sample trace of system logs.

The hidden Markov model (HMM) is used to study host-level system logs in [18] for identifying multiple APT phases. In this work, a sequence of low-level system logs pertaining to an APT phase is first extracted from a provenance system; next, it is translated to a sequence of abstracted states called storylines by HMM. Then, these high-level state sequences are fed into three multiclassification models: LSTM, 1-dimensional CNN, and SVM, to predict the APT phase of a sequence of system logs. Their results show that using HMM-based storylines outperforms using the original sequence of system logs in the multiclassification models. However, HMM requires prior knowledge of definitions of hidden states, and such models are computationally expensive.

HOLMES [21] is a bottom-up framework to detect APT from system logs with the help of provenance graphs. The three layers in this model from low to high are system logs, TTPs (tactics, techniques, and procedures) defined by MITRE ATT&CK, and APT stages. First, a provenance graph is generated from system logs, and then TTPs are identified from local structures of the graph based on predefined rules that are matched by node properties, edge types, and distances between nodes. Finally, a weighted product scoring rule is applied to aggregate a chain of TTPs and assign it a threat score. Though HOLMES is able to assign distinguishable threat scores to APT and benign scenarios, since it relies on predefined rules to map system logs to TTPs, this model has high requirements for prior knowledge.

A Bayesian LSTM neural network is used to detect APT system log traces [27], in which these fixed-length traces are obtained from a host-level system provenance graph. Context information and neighborhood information are encoded for each event node in the provenance graph. The advantage of BNN is that it provides uncertainty to the predictions. This work focuses on predicted APT traces with low uncertainty and constructs the attack story for such a trace by finding its most similar trace in the training set. However, their way of constructing the attack story assumes that the APT traces in the training set are well understood, which can be difficult to guarantee as the split of the training set and testing set usually requires data shuffling and random sampling.

UNICORN [28] is an APT detection model that periodically summarizes the graph histogram by counting unique subtrees in the whole system provenance graph of a host and compares graph histograms efficiently using graph sketching. It detects anomalies by building evolutionary models for regular behaviors and using unsupervised learning to determine the state of current graph histogram in each model, if the state does not match the state transition pattern of any evolutionary models, it indicates the existence of APT. However, UNICORN can lead to high false positives if the set of evolutionary models for regular behaviors is not complete, or there exist anomalies that are not APT related.

A detection model that is capable to identify novel APT attacks is proposed in [29] by analyzing provenance graphs derived from system events. By using online metric learning (OML), their model learns a latent feature embedding by minimizing the distance between provenance subgraphs of the same class and maximizing the distance between

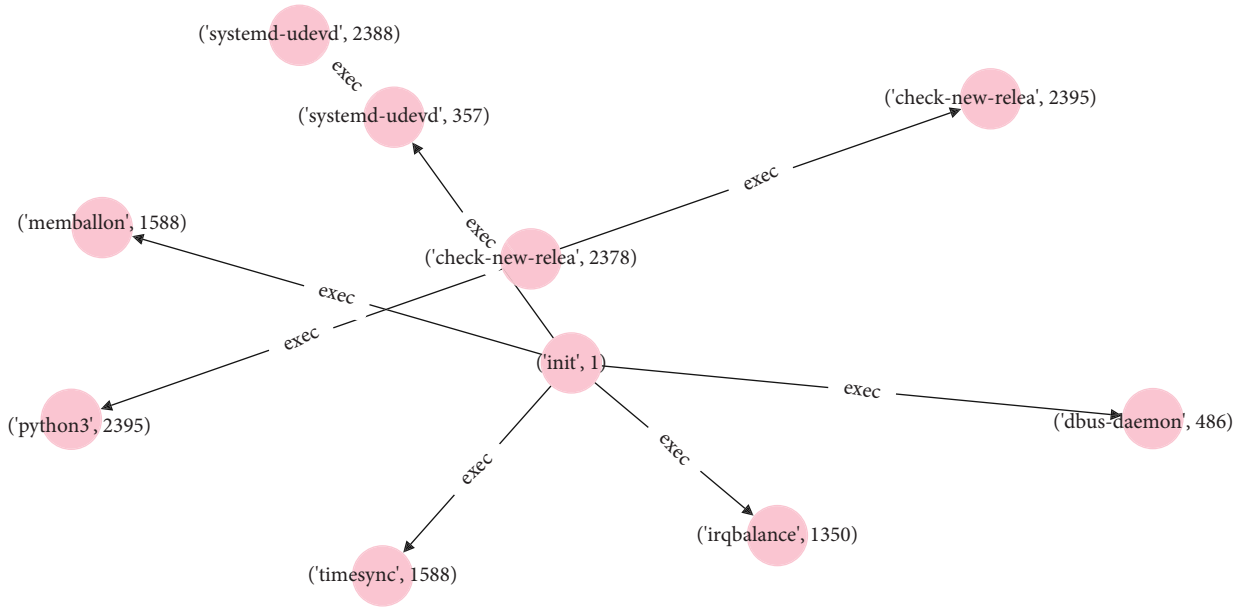

Figure 1: Provenance graph of a sample trace of system logs. Nodes represent system entities. Arrows and labels on edges indicate the direction and type of relations between entities.

subgraphs of different classes. However, using node2vec and the average over nodes to get the features of a graph in this paper only considers the neighboring information but omits the heterogeneity of nodes and edges. Additionally, the output of their model is either "benign" or "attacks," hence it cannot differentiate different stages of APT, though instances of different APT stages are labeled in their data.

There is no doubt that provenance graphs are an effective way to structure system logs and provide more informative data to improve the detection accuracy of APT attacks. However, more work needs to be performed to address some challenges in this area. First, appropriate graph-based learning approaches need to be applied to provenance graphs generated from system logs to efficiently detect APT by encoding important information such as the graph structure and properties of nodes and edges. Second, advanced APT can imitate benign activities or make use of benign applications to fulfill its goals, for example, by exploiting zero-day vulnerabilities of benign software. To detect such malicious behaviors of APT, lots of false positives will be incurred as APT alerts can also be triggered by benign activities. Therefore, strategic ways are needed to deal with false positives.

In this work, SR2APT, a novel host-based detection and response model against multistage APT attacks is proposed by analyzing system logs. To address the first challenge, an anomaly detection engine based on graph convolutional network (GCN) is built to identify different stages of APT and benign behaviors from subgraphs of a system log provenance graph. It improves detection accuracy by leveraging useful features hidden in the graph structure. To deal with the issue of false positives, a decision engine based on deep reinforcement learning is built, which provides optimal alert response actions such that actual APT attacks are prevented in earlier stages and minimal mistaken active defense actions are enforced on benign activities that generate false alerts.

3. Overview of Model Architecture

SR2APT can fit in the IDS architecture defined by the Intrusion Detection Working Group (IDWG), which consists of four modules: an event module, a database module, an analysis module, and a response module [30]. Figure 2 summarizes the functions of these modules and how functions are performed in SR2APT; the dashed arrows in it indicate important workflows including the inputs and the outputs of two engines that drive the APT detection and strategic response to alerts.

The event module incorporates a provenance graph generator, which dynamically processes system call events captured by monitoring tools such that the values of certain fields are properly extracted to update the causal relations on the provenance graph accordingly.

The database module stores all necessary information to support the functionalities of the analysis module, including

(i) A whole provenance graph that is updated dynamically; but the proposed APT detection engine does not take the entire graph as its input; instead, it analyzes focal subgraphs that are local structures of the entire graph. The subgraphs can be obtained by searching the neighborhood of a node of interest or a new node. In the experiments of this paper, a public dataset is used whose subgraphs are formed by querying specific node names backward a predefined number of steps. Each subgraph is sent to the detection engine in the analysis module. The detection result indicating the class of a subgraph is

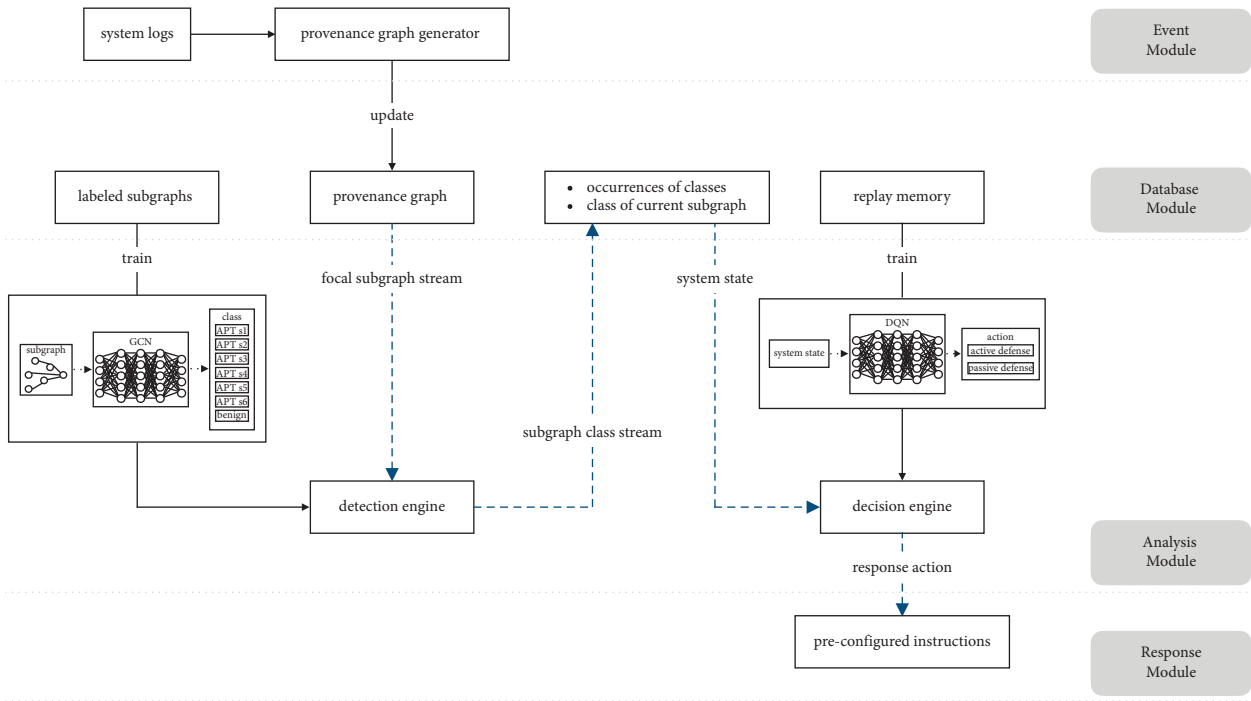

Figure 2: The functional modules and workflows in SR2APT. Blue dashed arrows indicate the input and output of the detection engine and decision engine.

stored in the database module, and the subgraph will be discarded after being evaluated.

(ii) Representative subgraphs with their class labels and abstracted representations including a node feature vector and an adjacency matrix; these subgraphs are used for training the graph convolutional network in the detection engine, and they can be periodically updated to adapt to new APT behavioral patterns.

(iii) Occurrences of all classes received from the detection engine within the current observing window, which will be used to form the state of the system together with the newest detection result. The reason for including historical counts of all classes is that they may exhibit different patterns when they are generated in attack scenarios and nonattack scenarios because APT stages usually appear more independently in nonattack scenarios than in attack scenarios [19].

(iv) Replay memory, which is a key term from the deep Q-network algorithm. It is used for training strategic response actions in the decision engine. Each entry in the replay memory is a 4-tuple: current system state, response action, next system state, and immediate reward.

The analysis module is where the intelligence lies, and it consists of a detection engine and a decision engine to effectively defend against multistage APT. The core of the detection engine is a graph convolutional network that classifies a graph as one of the multiple classes. In SR2APT, these classes include different stages of APT and a general benign class. The decision engine outputs the optimal response action to the APT alert based on the state of the system, which is formulated by deep reinforcement learning and is trained using the deep Q-network algorithm.

The response module automatically executes preconfigured instructions corresponding to the response action outputted from the decision engine in the analysis module. Since the main focuses of this work are novel modelings of the detection engine and the decision engine, a simplified set of response actions to APT alerts is considered in the experiments, in which only two types of responses are available: passive defense and active defense. Examples of preconfigured instructions in the response module for passive defense can be simply "no operation," and the instructions for active defense can be "stop process," "reset password," "reboot host," etc.

4. APT Detection and Response Model

In this section, technical details of the detection engine and decision engine in SR2APT are provided. For each part, the mathematical model used to formulate the problem is introduced first, followed by how it is integrated into SR2APT.

4.1. GCN-Based APT Stage Detection Engine

4.1.1. Graph Convolutional Network. To effectively classify a provenance graph generated from system logs, it is desired to encode more useful information such as the graph structure, properties of nodes, and edges. A graph convolutional network (GCN) model [31] is used in SR2APT to encode all this information to perform a multiclass classification task.

Additionally, it does not require prior knowledge to build the model, and it can handle graphs of various sizes directly.

The GCN model used in SR2APT is based on the eigen decomposition of the graph Laplacian matrix, and it addresses limitations of efficiency and the vertex localization problem in the first spectral-based graph convolutional network model [32], by truncating the Chebyshev polynomial to first order and relaxing the largest eigenvalue. More specifically, given an undirected graph $G = (V, E)$, where V is the set of nodes with $|V| = n$ and E is the set of edges with $|E| = m$, the adjacency matrix of the graph is denoted by A with $A_{ij} = w_{ij}$ where w_{ij} is the weight of the edge between node i and node j. If no edge exists between node i and node j, then $w_{ij} = 0$. For an undirected graph, A is symmetric. The degree diagonal matrix D is defined as $D_{ii} = \Sigma_{j=1}^{n} A_{ij}$, and the Laplacian matrix of the graph denoted by L is computed by $L = D - A$. Additionally, the normalized Laplacian matrix is computed by $\widetilde{L} = I - D^{-1/2}AD^{-1/2}$ where I is the identity matrix of size $n \times n$. In the GCN model, the input and output of each graph convolutional layer are a matrix X_l of size $n \times d_l$, which is the node feature representation where d_l is the dimension of the feature map defined for the layer l. The propagation operator applied to the graph convolutional layer in GCN is defined as follows:

$$X_l = \sigma\left(\overline{D}^{-1/2}\overline{AD}^{-1/2}X_{l-1}\Theta_l\right), \tag{1}$$

where X_l is the output node feature and X_{l-1} is the input node feature for the graph convolutional layer l. $\overline{A}$ is computed by $\overline{A} = A + I$ which adds self-loops to the adjacency matrix of the original graph. Accordingly, $\overline{D}$ is the degree diagonal matrix of $\overline{A}$, and Θ_l is the parameter matrix of size $d_{l-1} \times d_l$ which is learned in the model training phase. σ represents an activation function, which is usually a differentiable nonlinear function used in neural networks to learn more complex functions, e.g., ReLU, sigmoid, softmax, and tanh. X_0 is the node feature representation ingested by the first graph convolutional layer and the model input.

4.1.2. APT Stage Detection Engine. The GCN-based detection engine in SR2APT performs a multiclass classification task on provenance subgraphs. In the event module of SR2APT, system logs are structured in the form of a provenance graph using tools such as CamFlow [33], and GrAALF [34] is leveraged to build the subgraph dataset used for the experiments in this work. A valid system log record for a provenance graph should have two nodes with some identifiers and their relation. Taking GrAALF as an example, the fields "timestamp," "from_name," "from_id," "to_name," "to_id," and "event_type" are extracted from valid system log records. Then, based on the "id" information, the causal relations between nodes are established to form a provenance graph; "name" and "event_type" are labels of nodes and edges in the graph.

Subgraphs from a provenance graph are the neighboring structure around nodes of interest. Nodes of interest can be newly added nodes on a streaming provenance graph [21], specific processes, or files [34]. Then, the neighboring structure of a node can be obtained by using some search methods, for example, traversing backwards or forwards for a certain number of steps from the node. In the operation of SR2APT, the timestamp of nodes is used to make sure that subgraphs sent to the detection engine maintain the correct temporal order of system activities, which is important for solving optimal sequential response actions in the decision engine.

The labels of nodes and edges in a graph are encoded into numeric values, and a length threshold is applied to deal with long labels. The input of the GCN model, also known as the node feature X_0, is a vector of node labels. The values in the adjacency matrix of a graph, which is used for the operation of graph convolutional layers, are edge labels.

The GCN-based detection engine in the experiments of this paper consists of an input layer and three graph convolutional layers, and each of them is followed by a pooling layer, a dense layer, and an output layer. A cross-entropy loss function is used; therefore, the true label of a graph is represented as a one-hot vector. Seven classes are defined for the multiclass classification task, including six classes indicating APT stages 1, 2, 3, 4, 5, 6, and a general benign class. The model output is a 7-dimensional vector, which is a probability distribution over all seven classes, and the class with the largest probability is the predicted label of the input graph. The detection engine continuously transmits its outputs to the database module. If the current classification result is positive, indicating an alert of one APT stage, the current system state is then sent to the decision engine to decide the optimal response action to this alert.

4.2. DQN-Based APT Response Decision Engine

4.2.1. Deep Reinforcement Learning. The motivation of APT prevention in SR2APT is to strategically respond to APT alerts in a threat model where APT alerts can be triggered by both benign activities and APT attacks. Such a proactive defense approach can reduce the damage of APT by prohibiting it from evolving to deeper stages, which is more cost-effective than reactive approaches whose responses can be too late as APT might already or almost succeed at the moment of the defender's first response. However, enforcing active defense actions on the APT alerts triggered by benign activities incurs costs, which should be avoided as much as possible.

The decision engine in SR2APT considers APT characteristics (multistage, stealthiness, and advance), costs, and benefits of response actions to APT alerts from benign or APT attack activities. It then formulates the problem of strategic alert response based on deep reinforcement learning. The goal of a reinforcement learning agent is to learn a transformation from environment states to a probability distribution over feasible actions (also known as a policy) such that its cumulative reward is maximized [35]. By thoughtfully modeling the evolution of APT stages and the costs and benefits of response actions in different situations, a deep reinforcement learning model is able to drive the decision engine to generate an optimal response policy.

The mathematical formulation of reinforcement learning (RL) is based on a 5-tuple (S, A, R, P, γ); S is a set of environmental states that can be observed by the agent. A is a set of actions that the agent can take in each state. R is a set of rewards received by the agent after taking an action. P is a state transition model represented by a matrix where P_{ij} is the probability of transitioning from state i to state j. γ is the discount factor that controls the importance of future rewards in the characterization of the agent's cumulative reward. The goal of RL is to learn the optimal policy in each state. Since the system state defined in the decision engine of SR2APT is high-dimensional, a deep reinforcement learning algorithm is used, in which a neural network learns a function to map system states to a distribution over response actions.

In SR2APT, the defender is the only RL agent, but it can be extended to a multiagent problem by considering the attacker as another strategic agent. Figure 3 illustrates the abstracted interactions between an RL agent and the environment in a deep reinforcement learning setting. At time t, the agent takes action a_t in state s_t, then the environment transitions to the next state s_{t+1} and the agent receives its reward r_t of taking that action. As this is a sequential decision-making process and the action taken in the current state not only affects the immediate reward but also affects future rewards through state transition, the objective of the RL agent is to maximize the cumulative reward (also called value function) starting from the current state that is formulated as follows:

$$V(s_t) = r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \ldots + \gamma^{T-t} r(s_T). \quad (2)$$

If the action of the agent is taken as another independent variable, a state-action value function (also called the Q function) is formulated as follows, where $P(s'_{t+1}|s_t, a_t)$ is the probability of transitioning to state s'_{t+1}, when the agent takes action a_t in the state s_t:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s'_{t+1}} P(s'_{t+1}|s_t, a_t) \cdot V(s'_{t+1}). \quad (3)$$

Therefore, the objective function of the agent can be reformulated as follows:

$$\max_{\pi(s)} \sum_a Q(s, a) \cdot \pi(a), \quad (4)$$

where $\pi(s)$ is called a policy, which is a probability distribution over all feasible actions of the agent in the state s.

The deep Q-network (DQN) algorithm [36] is used to train the decision engine. DQN uses a deep convolution neural network called Q-network to approximate the state-action value function $Q(s, a, \theta)$. The advantages of this technique are twofold. First, it leverages a replay mechanism by randomly sampling a mini batch from a replay memory to train the Q-network for every update. Each data unit stored in the replay memory is represented by a 4-tuple (s_t, a_t, r_t, s_{t+1}), meaning that taking action a_t in state s_t leads to an immediate reward r_t and state s_{t+1}, and it can be viewed as an "experience." Second, a slowly updated neural network called a target network $Q(s, a, \theta')$ is used to compute the target value Q-network in every iteration. The parameters of the target network are copied from the Q-network periodically, and they are fixed in each iteration. These two special designs ensure the stability of the DQN algorithm, smooth out the learning process, and effectively avoid divergence.

The DQN algorithm is implemented as follows. Before training, a replay memory of fixed-length, which stores the most recent experiences, a Q-network $Q(s, a, \theta)$, and a target network $Q(s, a, \theta')$ with the same random parameters $Q(s, a, \theta')$ are initialized. In the beginning of every iteration i, the agent is in a state s_t, then it chooses an action a_t based on the rule of ε – greedy which balances exploitation (stick to the current best action) and exploration (randomly try other actions). After taking the action, the agent receives reward r_t and learns that the state transitions to s_{t+1}. The 4-tuple (s_t, a_t, r_t, s_{t+1}) is then stored in the replay memory. Once the size of the replay memory is greater than the predefined batch size, a batch of 4-tuples is sampled randomly from the replay memory to update the parameters of the Q-network according to the loss function as follows:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim B}\left[\left(r + \max_{a'} Q\left(s', a', \theta'\right) - Q(s, a, \theta_i)\right)^2\right], \quad (5)$$

where B is the sampled batch, the first $Q(\cdot)$ is the target network, and the second $Q(\cdot)$ is the Q-network. Note that, if s' is a terminal state, the term $\max_{a'} Q(s', a', \theta')$ in the loss function will not exist; therefore, the Q-network needs to be trained by a new episode of interaction.

4.2.2. Decision Engine. The problem setting in the decision engine of SR2APT is as follows. APT attackers intend to hide their behaviors by imitating benign activities, for example, the data exfiltration stage in APT might look like a normal file transfer behavior by an authenticated user. Therefore, the threat model considered in this paper is that APT-related alerts from the detection engine can be triggered by APT attacks or benign activities. For the sake of brevity, in the rest of the paper, actual APT activities were referred to as AAPT, and the complementary benign activities imitated by APT were referred to as BAPT. By constructing dedicated cost/reward functions based on the understanding of APT attacks, a strategic alert response policy can be generated by the decision engine such that AAPT is effectively impeded in earlier stages while less BAPT is mistakenly impeded, without explicitly knowing if an APT alert is triggered by AAPT or BAPT. To integrate APT characteristics into the decision engine, the effectiveness of active defense in stopping APT movement is assumed to be limited, as APT attackers are capable to exploit zero-day vulnerabilities. The experiments that evaluate the performance of the decision engine in this paper focus on a six-stage APT attack; however, it can be fitted into other APT attack scenarios by modifying the parameters in the deep reinforcement learning model. States, actions, rewards, and state transitions defined in the decision engine are described in detail as follows.

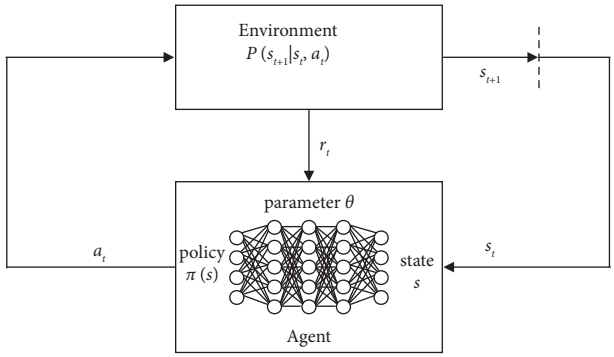

Figure 3: Interactions between agent and environment in deep reinforcement learning (a: action; r: reward; s: state; P: state transition model).

(1) States. The system states are constructed by fusing the classification results from the detection engine from the beginning of the current monitoring episode. It is represented by an 8-dimensional vector $s = (c_0, c_1, c_2, c_3, c_4, c_5, c_6, cs)$ where c_0 is the number of occurrences of the benign class, $c_i\,(i \in [1,6])$ is the number of occurrences of each APT s_i class, and cs indicates the current signal (classification result). Each episode starts from a system state where an alert of APT $s_i\,(i \in [1,5])$ is raised, for example, an initial system state can be $s = (0,0,1,0,0,0,0,2)$ which means the current signal is APT s_2 class and the occurrence of APT s_2 class is 1 while the occurrence of other classes is 0.

In the threat model considered by this paper, either the APT attacker or the defender takes control of the host in a monitoring episode. Therefore, terminal states are different depending on who is taking over the host. When the attacker is in control, a sequence of detection results that the decision engine receives is regarded as an AAPT alert trace. Terminal states are those states in which the current signal is 6 (indicating the attacker wins to the end), or the current signal is 0 and it results from the defender's active defense (indicating the defender takes over again), or the maximum length of a monitoring episode is reached. When the defender is in control, a sequence of detection results is regarded as a BAPT alert trace, and the only terminal state is the state in which the maximum length of a monitoring episode is reached.

(2) Actions. For nonterminal states, a defender can choose from two response actions: passive defense and active defense. In practice, examples of active defense are to investigate suspicious behaviors, patch vulnerabilities, reboot machines, and reset passwords; passive defense can be simply ignoring the alert, either because of not trusting it or for the purpose of continuous observation.

(3) Rewards. After taking a response action, the decision engine will receive a reward as feedback from the environment. In SR2APT, this reward is dependent on the current detection result and whether it is part of an alert trace of AAPT or BAPT and whether the action is active defense or passive defense. Since earlier APT stages usually have limited damage than later stages, an exponential function is used to quantify the damage of an APT stage and the cost of active defense to impede its movement. For example, if activities of an AAPT are being monitored and current detection result is APT stage s_i, the damage of the behavior that triggers this alert is $d^{s_i}\,(d > 0)$ and the cost of active defense against it is $c^{s_i}\,(c > 0)$, then the reward of choosing active defense action is $d^{s_i} - c^{s_i}\,(d > c)$ while the reward of choosing passive defense action is $-d^{s_i}$, However, if active defense action is enforced on the APT alerts triggered by benign activities, the defender receives a penalty. In SR2APT, this penalty is also an exponential function of APT stage, that is, if current detection result is APT stage s_i which is triggered by a BAPT, then the reward of taking active defense action is $-p^{s_i}\,(p > 0)$ while the reward is 0 by taking passive defense action.

(4) State Transition. Deep reinforcement learning allows to model real-world uncertainties via probabilistic state transitions. In SR2APT, state transitions are decided by the ordering of detection results, which is affected by the interdependencies between APT stages conditioning on the defender's responses to APT alerts. In this paper, the effectiveness of active defense action against APT is assumed to be limited, in order to model an APT attacker's ability to exploit zero-day vulnerabilities. In addition, the ordering of different detection results in an AAPT alert trace and a BAPT alert trace should exhibit different patterns because APT stages are more interdependent in AAPT but not in BAPT. It would be ideal to use real AAPT alert traces and BAPT alert traces to train the decision engine, but synthetic alert traces also work if real data is not available as long as the synthetic traces can represent the behavioral patterns of AAPT and BAPT. In the experiments of this paper, synthetic alert traces are used and they are generated based on the following behavioral patterns of AAPT and BAPT.

(i) Current detection result is the benign class. No matter if it is part of AAPT or BAPT, the defender does not respond to it. Therefore, if it is part of AAPT, then it indicates that the attacker is staying quiet, and the next expected detection result is the benign class or a higher APT stage according to the attacker's movement plan. If it is part of BAPT, the next expected detection result is one of the seven classes defined in the detection engine at random.

(ii) Current detection result is an APT stage s_i and it is part of AAPT. When the defender responds by taking a passive defense action, the attacker will stay quiet for a random period of time and then evolve to stage s_{i+1}, When the defender takes an active defense action, the movement of APT would be effectively impeded with probability $e\,(0 < e < 1)$, in this case, the attacker will either be kicked out of the defender's system or stay quiet for a while then restart from an earlier stage $s_j\,(j \in [1,i))$. However, the active defense could also be ineffective with probability $(1 - e)$, in this case, the attacker will stay quiet for a while then evolve to stage s_{i+1}.

(iii) Current detection result is an APT stage s_i and it is part of BAPT. No matter what response action the defender takes, the next expected detection result is either the benign class or APT stage s_j ($j \in [1, 6]$) at random.

5. Evaluation

In this section, the proof of concept experiments are designed and implemented to evaluate the performance of SR2APT in defending against APT. First, a semisynthetic dataset of system log provenance graphs is used to evaluate the performance of the GCN-based detection engine in the subgraph classification. Then, synthetic alert traces from both APT and benign scenarios are used to compare the performance of the strategic alert response policy from the DQN-based decision engine and two baseline fixed response policies.

5.1. Provenance Subgraph Classification. The dataset of system log provenance graphs from [18] is used to train and test the detection engine, which covers the system logs of system activities about six APT stages: system reconnaissance, network discovery, persistence, privilege escalation, asset discovery, and data exfiltration; as well as some benign system activities.

To construct the dataset, first Sysdig is used to monitor the activities on a host and generate system logs, which allows up to 57 fields to structure raw log data. Then, GrAALF [34] is used to get subgraph samples for each class in the detection engine. It is a Java application that can process raw system logs, store processed data in a graph database, and support queries to retrieve subgraphs. Regarding the dataset used in the experiment of this paper, nodes of interest are predefined for each APT stage behavior e.g., "whoami" is a keyword for the reconnaissance stage [18]. Then a subgraph sample for an APT stage is obtained by starting from the nodes of interest to traverse backwards or forwards for certain steps. For instance, Figure 4 shows the subgraph data returned by GrAALF, which corresponds to the APT privilege escalation stage, by running the query "*back select ∗ f rom ∗ where name is su*." Eventually, 300 subgraphs for each of the seven classes are obtained by using the provided query templates and resampling. The seven classes in the experiments are benign (class 0), APT s1 (class 1), APT s2 (class 2), APT s3 (class 3), APT s4 (class 4), APT s5 (class 5), and APT s6 (class 6). Out of total 2100 subgraphs, 80% are used for training the detection engine, 10% are used for validation, and 10% are used for testing.

Figure 5 shows the confusion matrix generated by comparing the true labels and predicted labels of 210 tested subgraphs. The X-axis and Y-axis are labeled by the seven classes defined in the detection engine, and the value at row i and column j is the number of subgraphs whose true label is class i and predicted label is class j. As shown in Figure 5, the values at off-diagonal positions are much smaller than the values at diagonal positions for every row and column, meaning that the detection engine of SR2APT can effectively identify malicious behaviors of different APT stages and benign behaviors, with overall 94% classification accuracy, which outperforms other compared classification models SVM (83%), CNN (84%) and LSTM (86%).

The basic metrics for evaluating the performance of a classifier are true positive (TP), true negative (TN), false positive (FP), and false negative (FN). TP is the number of positive instances that are correctly classified as positive. TN is the number of negative instances that are correctly classified as negative. FP is the number of negative instances that are incorrectly classified as positive. FN is the number of positive instances that are incorrectly classified as negative. Other important metrics including the true positive rate (TPR), false positive rate (FPR), precision, and recall are calculated as follows:

$$\begin{aligned} \text{TPR} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ \text{FPR} &= \frac{\text{FP}}{\text{FP} + \text{TN}}, \\ \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}. \end{aligned} \tag{6}$$

The AUC-ROC curve of each class classified by the detection engine of SR2APT is shown in Figure 6, which evaluates the performance of the proposed detection engine in classifying each class, under different probability threshold settings. When the probability threshold changes, it could be the case that both the TPR and the FPR change or only one of them changes, thus producing vertical, horizontal, and slant segments on an AUC-ROC curve. The black straight dashed line is a benchmark indicating a model has no classification power. Figure 6 shows the AUC (area under the curve) of each class classified by the detection engine is almost 1.00, meaning that the detection engine is not sensitive to the probability threshold, and the learned feature representation for each class is effective.

$$\text{F1} - \text{score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{\text{TP}}{\text{TP} + 1/2\,(\text{FN} + \text{FP})}. \tag{7}$$

Additionally, the F1-score is used to compare the proposed GCN-based detection engine with some classic classification models, because it evaluates a model by considering both FP and FN. From Equation (7), the F1-score is the harmonic mean of precision and recall, where precision reflects the impact of FP and recall reflects the impact of FN. The F1-score of a model is high if both precision and recall are high, and is low if one or both of them are low. As the detection results are important signals to the decision engine, both FP and FN are not desired, thus the F1-score is a good metric to compare detection models using different kernels. Three commonly used kernels of classification models for anomaly detection are tested on the same dataset, which are support vector machine (SVM), one-dimensional

```
{ "sequence_number": 42578,
  "user": "boba_fett",
  "from_id": 2402 ,
  "from_name": "perl",
  "evt_type": "exec",
  "to_name": "sh" ,
  "to_id": 2665 ,
  "count": 1},

{ "sequence_number": 42592,
  "user": "boba_fett",
  "from_id": 2665 ,
  "from_name": "sh",
  "evt_type": "exec",
  "to_name": "fl7Klh5a" ,
  "to_id": 2666 ,
  "count": 1},

{ "sequence_number": 43311,
  "user": "boba_fett",
  "from_id": 2666 ,
  "from_name": "fl7Klh5a",
  "evt_type": "exec",
  "to_name": "su" ,
  "to_id": 2679 ,
  "count": 1}
```


Figure 4: A sequence of system logs for an APT privilege escalation stage.

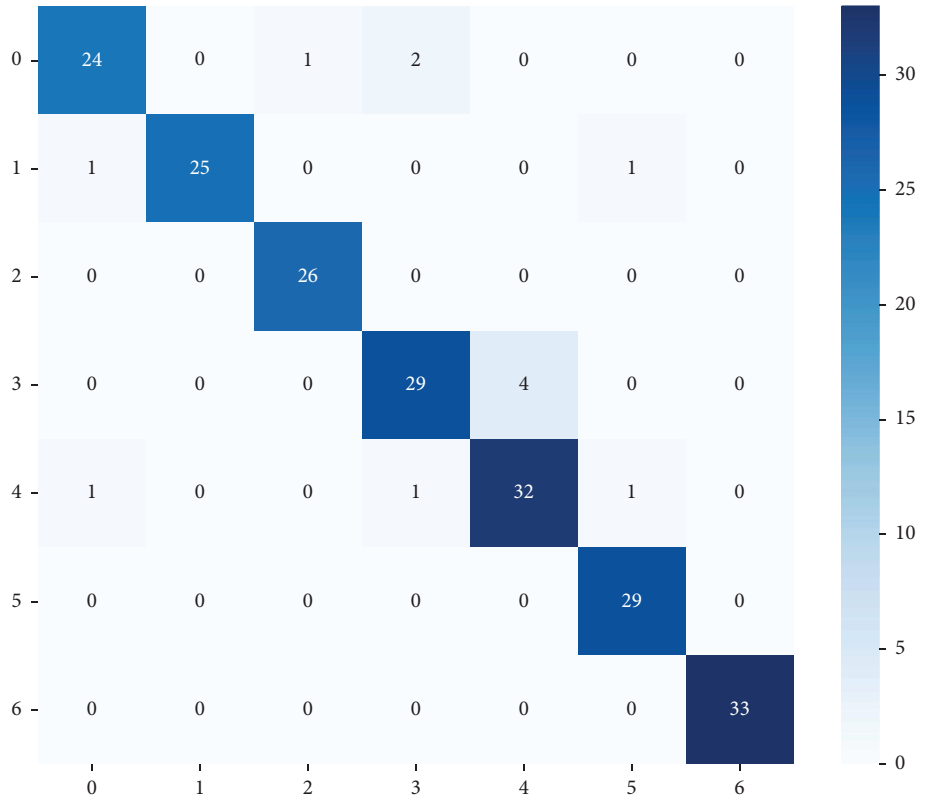

Figure 5: The confusion matrix for the tested provenance subgraphs.

convolutional neural network (CNN), and long short-term memory (LSTM). To implement the kernels other than GCN, "from_name," "evt_type," and "to_name" in the sequence of system logs corresponding to a subgraph are extracted in temporal order to form the input of classification models. The parameters of every model are tuned such that it achieves the highest accuracy. The sklearn Python library is used to implement the linear SVM model. The Keras Python library is used to implement the one-dimensional CNN model and the LSTM model. The one-dimensional CNN consists of one 1D convolutional layer (filters = 20, kernel size = 3), one 1D max pooling layer (pool size = 3), and one dense layer with softmax activation function being used. For the LSTM neural network, it consists of one masking layer, three LSTM layers with each followed by a dropout layer (dropout rate = 0.2), and one dense layer using the softmax activation function. To train the one-dimensional CNN model and the LSTM model, the Adam optimizer and categorical cross-entropy loss function are used. As shown in Table 1, the GCN-based detection

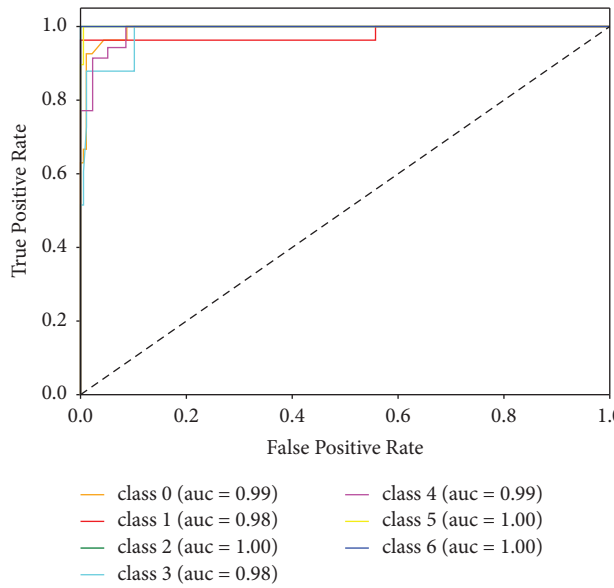

Figure 6: The AUC-ROC curve for each class classified by the detection engine of SR2APT.

Table 1: The F1-score of classification models based on SVM, CNN, LSTM, and GCN regarding each class.

	Benign	APT s1	APT s2	APT s3	APT s4	APT s5	APT s6
SVM	1.00	0.52	0.96	0.80	0.73	0.89	0.89
CNN	**0.98**	0.65	0.95	0.72	0.77	0.89	0.90
LSTM	0.95	0.62	**1.00**	0.88	0.87	0.78	0.87
GCN	0.91	**0.96**	0.98	**0.89**	**0.90**	**0.97**	**1.00**

engine achieves the highest F1-score for classes of APT s1, APT s3, APT s4, APT s5, and APT s6. For the other two classes of benign and APT s2, the GCN-based detection engine still achieves an F1-score that is greater than 0.9. Therefore, GCN improves the performance of the detection engine in provenance subgraph classification compared to SVM, CNN, and LSTM, because it leverages useful features hidden in the dependencies of nodes in a graph.

5.2. Strategic Response Policy to APT Alerts. To train and evaluate the performance of the decision engine that outputs strategic response actions to sequential APT alerts within a monitoring window, synthetic alert traces for APT scenarios (AAPT) and benign scenarios (BAPT) are used. These synthetic alert traces are generated according to the behavioral patterns of the APT attack considered in this paper, which are affected by the defender's response actions. More details about how synthetic alert traces are generated are provided in the state transition part of Section 4.

When implementing the proposed decision engine, preliminary works are needed to provide gain/loss feedback after taking a response action, which is host-dependent and is out of the scope of this paper. To generate the feedback information for the proof of concept experiments, the parameter settings used in the experiments are: $d = 4$, $c = 1.5$, $p = 1.5$ which are the base in the exponential function representing the damage of each APT stage, the cost of active defenses against each APT stage, and penalty of mistaken active defenses to APT alerts triggered by benign activities, respectively; $e = 0.9$ which indicates the effectiveness rate of active defenses in slowing down the movement of APT. Note that the parameters in the experiments are chosen only because they allow the total reward of the DQN response policy to be positive, but they can be other values. The advantage of the DQN policy is not dependent on parameter settings, rather it is because the DQN policy is learned from trial and error. Therefore, with sufficient experiences the DQN policy is always able to approximate the optimal policy. When implementing the decision engine, the defender should use their own feedback mechanism based on the status of the protected target, rather than the parameter settings used in the proof of concept experiments.

To demonstrate the advantages of the response policy generated by the decision engine (DQN policy) in the SR2APT model, the DQN policy is compared with two fixed rate response policies. A fixed rate policy p is defined as follows: every time an APT alert is reported, a defender takes an active defense action with a fixed probability p. One baseline fixed rate policy is $p = 1.0$ which represents the most aggressive response policy, and the other fixed rate policy is $p = 0.5$.

The decision engine was trained for 100 episodes. Each episode simulates an alert trace of AAPT or BAPT, which is dynamically affected by the defender's response actions. Before the start of an episode, whether this episode simulates AAPT or BAPT is decided by the ratio of AAPT: BAPT = 3 : 7, which represents the defender's perception of how often

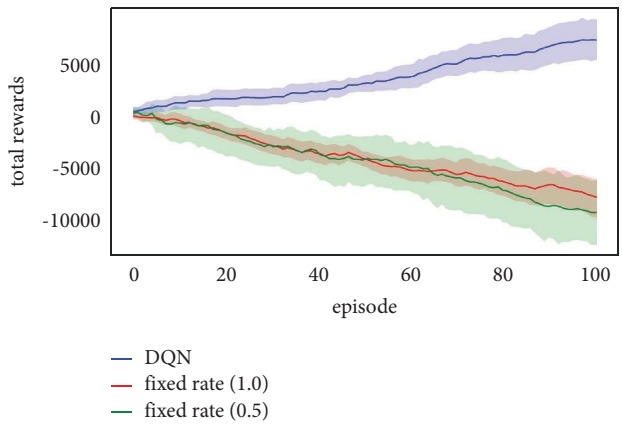

Figure 7: Total rewards by following the DQN response policy from the decision engine of SR2APT and two fixed response policies.

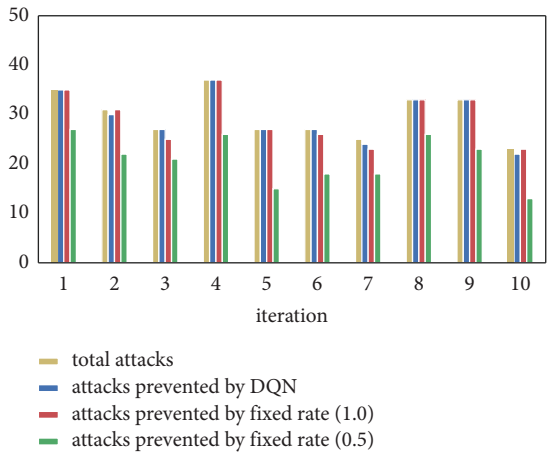

Figure 8: Number of APT attacks prevented by following the DQN response policy from the decision engine of SR2APT and two fixed response policies.

the system is taken over by the attacker and the defender. Once the decision engine is trained, the DQN response policy is evaluated by running the decision engine for 10 iterations, where each iteration consists of 100 episodes.

First, the total rewards are compared by following the DQN alert response policy from the decision engine and two fixed rate policies. As shown in Figure 7, the two fixed rate policies perform equally worse than the DQN policy. Although a more aggressive fixed rate policy can prevent damage from APT activities by frequently taking active defenses, it also increases the probability of mistaken active defenses to false APT alerts triggered by benign activities, and vice versa. However, the DQN policy learns to implicitly treat the APT alerts triggered by AAPT and BAPT differently, to minimize the penalties from mistakenly impeding BAPT and maximize the gains from slowing down the movement of AAPT by taking active defenses.

Next, the ability of each response policy to successfully prevent AAPT is evaluated. Here, a successful prevention against AAPT means that the APT attacker is defeated out of the defender's system. In Figure 8, the yellow bar represents the total number of APT attacks in each iteration, and the other three bars represent the number of attacks that are successfully prevented by following the three response policies, respectively. As shown in Figure 8, the less aggressive fixed rate policy ($p = 0.5$) is the worst, but both the DQN policy and the most aggressive fixed rate policy ($p = 1.0$) are able to prevent almost all attacks in every iteration. Since the most aggressive fixed rate policy instructs the defender to take an active defense action against all APT alerts, it is difficult for an APT attack to succeed under this circumstance. Although the DQN response policy from the decision engine treats APT alerts differently, it still performs extremely well to prevent attacks, and it achieves greater total rewards than the most aggressive fixed rate policy.

To further address the advantage of the strategic DQN response policy, the number of mistaken active defenses enforced on BAPT by following the three policies are collected. In Figure 9, each colored bar represents the total number of active defense actions enforced on the APT alerts triggered by BAPT by following the corresponding policy. It shows that the most aggressive fixed rate policy ($p = 1.0$) is the worst, as it instructs the defender to take active defense actions for all APT alerts from BAPT, which is not a wise policy for the considered threat model, where BAPT alert traces appear more frequently than AAPT. The less

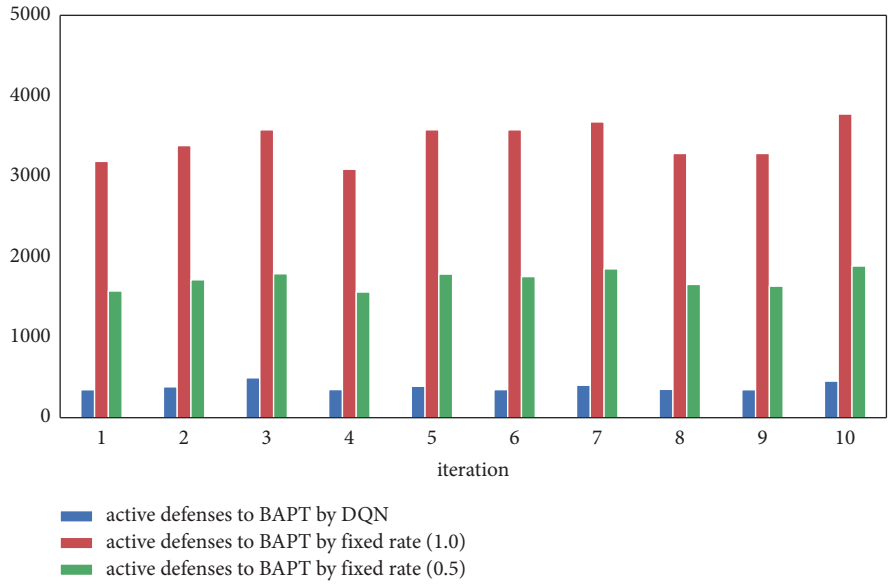

Figure 9: Number of active defense actions enforced on benign activities by following the DQN response policy from the decision engine of SR2APT and two fixed response policies.

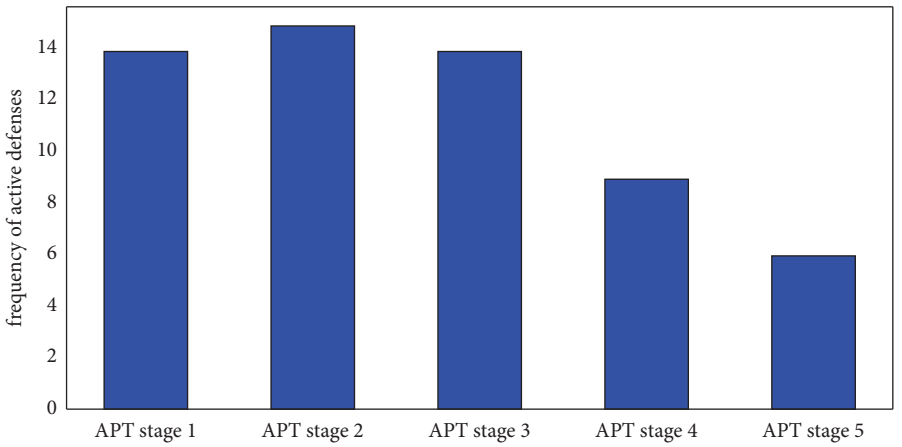

Figure 10: Average frequencies of active defense actions to APT stages of 1 to 5 in APT scenarios.

aggressive fixed rate policy ($p = 0.5$) reduces the number of mistaken active defenses to benign activities, but it is still much worse than the DQN policy. As the system states in the decision engine take historical detection results into consideration, it is likely that the system states for an alert from AAPT and an alert from BAPT are quite different because of the interdependencies between APT stages; hence, the DQN policy will choose not to take an active defense action to an alert triggered by BAPT as much as possible.

Finally, the strategic DQN alert response policy from the decision engine prevents APT attacks in earlier stages. The average frequencies of active defense actions in one iteration are collected for each APT stage, except for the APT stage s_6, which is an indicator of a terminal state where no response action is available. As shown in Figure 10, more active defense actions are enforced on the alerts corresponding to earlier stages of APT attacks, which is as expected because greater damages and costs of defense are assigned to later APT stages in the considered threat model.

Overall, the strategic APT alert response policy generated by the decision engine in the SR2APT model achieves the best performance compared with two fixed rate response policies, regarding the total rewards in the defense against APT, the ability to successfully prevent APT attacks, and mistaken active defenses for false APT alerts triggered by benign activities.

6. Conclusion

In this paper, a novel host-based APT detection and response model called SR2APT is proposed, which can be integrated into cyber-physical systems to provide cost-effective and agile protection to critical hosts against sophisticated APT attacks. The proposed model addresses two important but overlooked problems in the defense against multistage APT: preventing APT attacks in earlier stages and minimizing the impediments of mistaken response actions to benign

activities that generate false alerts. SR2APT tackles these two problems through a strategic APT alert response policy.

The detection engine of SR2APT applies a graph convolutional network to analyze subgraphs of a provenance graph generated from system logs, to detect malicious behaviors of each APT stage as well as benign behaviors. The experimental results on a semisynthetic dataset show that the GCN-based detection engine obtains 94% classification accuracy and outperforms classification models based on other kernels including SVM, CNN, and LSTM. To derive optimal response actions to sequential APT alerts from the detection engine, the decision engine of SR2APT is built upon deep reinforcement learning, which considers the characteristics of APT attacks such as multistage, stealthiness, and zero-day exploitation capability. With dedicated cost and reward functions in the decision engine such that they are compatible with the threat model considered in this paper, the alert response policy generated by the decision engine outperforms two baseline fixed response policies and achieves the highest total rewards in the defense against APT. Because it optimally balances the trade-off between maximizing the number of prevented APT attacks and minimizing the impediments to benign activities from mistaken active defense actions. In addition, the experimental results show that the strategic alert response policy from the decision engine is able to prevent APT attacks in earlier stages by taking advantage of deep reinforcement learning.

SR2APT can be applied to other APT scenarios, because both its detection engine and decision engine are flexible. This work focuses on multistage APT attacks on a host, thus a potential extension is to apply SR2APT to defend against APT attacks moving across multiple hosts. In that case, host-level system logs need to be replaced by network-level monitoring data, e.g., network traffic in the detection engine, to detect the lateral movement stage of APT. In addition, strategic alert responses in different situations where the defender has incomplete monitoring information or limited defense resources are also promising directions of the future work.

Data Availability

The host-level system logs data and synthetic APT alert traces for both APT and benign scenarios used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

Publication of this article was funded by the University of Colorado Boulder Libraries Open Access Fund.

References

[1] S. Han, M. Xie, H. Chen, and Y. Ling, "Intrusion detection in cyber-physical systems: techniques and challenges," *IEEE Systems Journal*, vol. 8, no. 4, pp. 1052–1062, 2014.

[2] L. Huang and Q. Zhu, "A dynamic games approach to proactive defense strategies against Advanced Persistent Threats in cyber-physical systems," *Computers & Security*, vol. 89, Article ID 101660, 2020.

[3] R. Langner, "Stuxnet: dissecting a cyberwarfare weapon," *IEEE Security and Privacy Magazine*, vol. 9, no. 3, pp. 49–51, 2011.

[4] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019.

[5] A. Juels and T. Yen, "Sherlock Holmes and the case of the advanced persistent threat," in *Proceedings of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 12)*, San Jose, CA, USA, Apr. 2012.

[6] A. Rot and B. Olszewski, "Advanced persistent threats attacks in cyberspace. Threats, vulnerabilities, methods of protection," in *Proc. FedCSIS (Position Papers)*, pp. 113–117, Czech Republic, Prague, 2017.

[7] D. Gonzales, J. M. Kaplan, E. Saltzman, Z. Winkelman, and D. Woods, "Cloud-Trust: a security assessment model for infrastructure as a service (IaaS) clouds," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 523–536, 2017.

[8] R. Mitchell and I. R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–29, Apr. 2014.

[9] P. N. Bahrami, "Cyber kill chain-based taxonomy of advanced persistent threat actors: analogy of tactics, techniques, and procedures," *Journal of Information Processing Systems*, vol. 15, no. 4, pp. 865–889, Aug. 2019.

[10] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[11] N. Villeneuve and J. Bennett, "Detecting APT activity with network traffic analysis," *Trend Micro Inc*, vol. 547, p. 78, 2012 Available at: http://www.trendmicro.com/cloud-content/us/pdfs/securityintelligence/.

[12] X. Wang, "Detection of command and control in advanced persistent threat based on independent access," in *Proc. IEEE Int. Conf. Commun. (ICC)*, pp. 1–6, Kuala Lumpur, Malaysia, May 2016.

[13] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido, "Analysis of high volumes of network traffic for advanced persistent threat detection," *Computer Networks*, vol. 109, pp. 127–141, Nov. 2016.

[14] N. Nissim, A. Cohen, C. Glezer, and Y. Elovici, "Detection of malicious PDF files and directions for enhancements: a state-of-the art survey," *Computers & Security*, vol. 48, pp. 246–266, Feb. 2015.

[15] J. V. Chandra, N. Challa, and S. K. Pasupuleti, "A practical approach to E-mail spam filters to protect data from advanced persistent threat," in *Proc. 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–5, Nagercoil, India, Mar2016.

[16] S. Chandran, P. Hrudya, and P. Poornachandran, "An efficient classification model for detecting advanced persistent threat," in *Proc. 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2001–2009, Kochi, India, Aug. 2015.

[17] N. Mohamed and B. Belaton, "SBI model for the detection of advanced persistent threat based on strange behavior of using credential dumping technique," *IEEE Access*, vol. 9, pp. 42919–42932, 2021.

[18] M. AbuOdeh, "A novel AI-based methodology for identifying cyber attacks in honey pots," in *Proc. AAAI Conference on Artificial Intelligence*, pp. 15224–15231, Feb. 2021.
[19] J. Sexton, C. Storlie, and J. Neil, "Attack chain detection," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 8, no. 5-6, pp. 353–363, Aug. 2015.
[20] P. Giura and W. Wang, "A context-based detection framework for advanced persistent threats," in *Proceedings of the 2012 International Conference on Cyber Security*, pp. 69–74, Alexandria, VA, USA, Dec. 2012.
[21] M. M. Sadegh, "Holmes: real-time apt detection through correlation of suspicious information flows," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, pp. 1137–1152, San Francisco, CA, USA, May 2019.
[22] H. Studiawan, F. Sohel, and C. Payne, "A survey on forensic investigation of operating system logs," *Digital Investigation*, vol. 29, pp. 1–20, Jun. 2019.
[23] A. Awad, "Data leakage detection using system call provenance," in *Proceedings of the 2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, pp. 486–491, Ostrava, Czech Republic, September 2016.
[24] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.
[25] Y. Liao and V. R. Vemuri, "Using text categorization techniques for intrusion detection," in *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, USA, Aug. 2002.
[26] B. Subba, S. Biswas, and S. Karmakar, "Host based intrusion detection system using frequency analysis of n-gram terms," in *Proceedings of the 2017 IEEE Region Ten Conference (TENCON)*, pp. 2006–2011, Penang, Malaysia, Nov. 2017.
[27] M. Anjum, S. Iqbal, and B. Hamelin, "ANUBIS: a provenance graph-based framework for advanced persistent threat detection," Dec. 2021, Available at: https://arxiv.org/abs/2112.11032.
[28] X. Han, "Unicorn: runtime provenance-based detector for advanced persistent threats," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, Feb. 2020.
[29] G. Ayoade, "Evolving advanced persistent threat detection using provenance graph and metric learning," in *Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, Avignon, France, Feb 2020.
[30] V. Jyothsna, V. Rama Prasad, and K. Munivara Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Application*, vol. 28, no. 7, pp. 26–35, Aug. 2011.
[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 2017 International Conference on Learning Representations*, pp. 1–14, Toulon, France, Apr. 2017.
[32] J. Bruna, "Spectral networks and locally connected networks on graphs," in *Proceedings of the 2nd International Conference on Learning Representations*, Banff, Canada, Apr. 2014.
[33] T. Pasquier, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 405–418, 2017.
[34] O. Setayeshfar, C. Adkins, M. Jones, K. H. Lee, and P. Doshi, "Graalf: supporting graphical analysis of audit logs for forensics," *Software Impacts*, vol. 8, Article ID 100068, 2021.
[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, Cambridge, MA, USA, 2018.
[36] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.