Contents lists available at ScienceDirect

Journal of Information Security and Applications

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

A Model-Free Approach to Intrusion Response Systems

Kieran Hughes [*], Kieran McLaughlin, Sakir Sezer

Centre for Secure Information Technologies, Queen's University Belfast, UK

ARTICLE INFO

Keywords:
Intrusion
Response
Systems
Reinforcement
Learning
Network
Security

ABSTRACT

With the rising number of data breaches, denial of service attacks and general malicious activity facing modern computer networks, there is an increasing need to quickly and effectively respond to attacks. Intrusion Detection Systems provide an automated method of identifying malicious activity within a network however the development of an Intrusion Response System which can automatically respond to these alerts is non-trivial. Current research in IRS proposes model-based methods for identifying possible routes a malicious actor may take when attacking a network and use subjective performance values for the cost and benefit of a response, both of which can be invalidated by the increasingly dynamic nature of network topologies and system configurations. The IRS proposed in this work utilises a Model-free Reinforcement Learning approach and evaluates the Reinforcement Learning agent's performance in stopping two distinct multi-stage attack scenarios on a virtualised testbed. Experimentation demonstrates that the agent can successfully halt both attack scenarios and find responses which have minimal impact on normal network operation based on experience gained through training. A further contribution is the novel use of a virtualised environment that demonstrates Intrusion Response Reinforcement Learning performance in a more realistic environment than simulated tasks common to previous literature.

1. Introduction

In light of modern, complex computing systems and the increasing online presence of businesses, the potential for malicious activity and network compromise has increased significantly. With the ease of access to information and the rapid increase in social media usage, security flaws and vulnerabilities are quickly and efficiently communicated to the wider Cyber Security community. However, this knowledge-sharing of exploits and proof of concepts also plays into the hands of skilled opportunist attackers and those with limited expert knowledge. Consequently, the number of cyber-attacks continues to rise. In an attempt to combat this increase in both internal and external threats, organisations have deployed Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) [1]. These systems are becoming increasingly proficient in detecting attempted or ongoing attacks on a computer network and providing alerts to a network administrator or security analyst. An expanding threat landscape and the emergence of easy-to-use attack tools results in larger workloads for Security Operations Centres (SOC), which are created to monitor networks and respond to alerts generated by IDS. While the development of IDS has been a more active area of research, Intrusion Response Systems (IRS) have been slowly advancing in recent years. Early IRS included static mappings between known alerts and practical responses [2]. Ultimately this met problems, as the size and complexity of computer networks began to grow, the number of static mappings which needed to be maintained grew significantly. This form of IRS is present in industry today, providing Security Incident and Event Management (SIEM) tools to administrators which enable the recording of a manual response to an alert as a macro, then automatically executing the macro upon another occurrence of said alert. More recent research into IRS acknowledges the difficultly in maintaining these static mappings and moved towards a dynamic response approach, based on modelling response selection as a Multi-Objective Optimisation Problem (MOOP). This approach involves the use of Attack Graphs (AG) or Attack Trees (AT) which model the network on which the IRS is deployed, identifying possible paths an attacker may take in order to reach their goal, such as a database [3]. Expert knowledge is used alongside this mapping to identify values for the positive and negative impact of applying a response at a particular defence point [4]. While this approach addresses the static response problem, modern networks are adopting increasingly dynamic. Therein lies the inherent difficulty of continuously re-modelling the network as nodes join or leave. Furthermore, arbitrary values are used to represent the potential impact or benefit gained when issuing a response. It is notably difficult to accurately predict the effect a particular

* Corresponding author.
E-mail address: khughes44@qub.ac.uk (K. Hughes).

https://doi.org/10.1016/j.jisa.2022.103150

Available online 7 March 2022

countermeasure will have on a live environment. The resultant potential for significant operational and financial risk reduces the likelihood of current IRS approaches being adopted by industry.

In this work, a Model-free Reinforcement Learning (RL) approach is used to combat the aforementioned problems. To facilitate the evaluation of non-arbitrary values for the effect of applying a response on a live network, an emulated training environment has been created. In this environment a RL agent can learn the near optimal response to specific attack scenarios, based on realistic network performance metrics.

For example, in approaches which use simulated testbeds and attack scenarios, tangible performance metrics such as traffic statistics, node behavior and timings are harder or impossible to capture, resulting in a reduced ability to accurately assess how the IRS will perform in a live network. To the best of our knowledge this is the first time an emulated environment has been used to implement RL for Intrusion Response. The main contributions of the work outlined in this paper are:

1 Requirements outline and practical implementation of a Model-free Reinforcement Learning Agent deployed in an emulated network, building on simulated approaches.
2 A novel quantifiable method of measuring the Cost and Benefit of applying a response in a live network environment in the form of a reward function.
3 Experimentation which demonstrates the ability of a Reinforcement Learning Agent to learn an effective response to multi-stage attack scenarios, building on related work which experiment with a single attack step or simulate a compromised network state.

The rest of this paper is organised as follows, Section 2 discusses related work Section 3. provides a background of Reinforcement Learning, Model-free approaches and the concept of Reward Section 4. outlines the methodology of the presented approach. It discusses how this work moves away from current theoretical approaches for RL in Intrusion Response. To apply RL to IRS it is necessary to develop a framework which can translate the functionality and working environment of an IRS to the components of RL such as a Partially Observable Markov Decision Process (POMDP), an action space and an environmental state space. This translation and the practical testbed and attack scenarios are discussed Section 5. presents the results of several experiments, specifically evaluating performance with differing RL configurations. Finally, Section 6 concludes and outlines potential future work.

2. Related work

While there are a number of works on sub-components of IRS such as countermeasure cost-benefit analysis and attack-defence network modelling, the number of proposed complete IRS is limited.

One of the pioneering IRS works was ADEPTS [5]. ADEPTS mapped out the possible routes an attacker could take when seeking to move through a network and named it I-Graph. I-Graphs were subsequently adopted by later IRS works and eventually became known as Attack Graphs. Subsequent work also inherited the problem associated with creating map of a network that is unable to dynamically update. While this approach was understandable at the time as traditional network environments were more rigid and did not typically change often, continuing to use this approach is futile due to the more dynamic nature of modern network topologies. Another defining aspect of ADEPTS was its use of a static cost for response evaluation. In a somewhat deterministic network such as an Industrial Control Network (ICS), which has minimal deviation in its network activity and operation, a static response cost could be formulated due to a clear understanding of how it will affect benign network activity. However, in the E-commerce network proposed and in modern networks today, this approach is not ideal.

A further advancement in IRS was Network Intrusion Detection and Countermeasure Selection in Virtual Networked Systems (NICE) [6]. NICE proposed an interesting approach through merging their own intrusion detection system and using alerts to identify the current operational status of each virtual machine (VM), referring to it as the state. However, the possible states of the VMs were limited to four options; stable, vulnerable, exploited and zombie. While this transition to evaluating the current situation of nodes before deploying responses is a step forward, it lacks the ability to accurately represent the state of the network and how the network is performing based on the current node states. NICE also uses an attack graph approach, however it is not completely static, if an alert is unknown and the system cannot evaluate a potential path, it then updates the graph post attack. While this is an improvement over ADEPTS, it is simply reacting to an unknown attack and does compensate for the addition of new virtual machines or potential vulnerabilities before they are exploited.

One of the most prominent approaches to IRS was Shameli-Sendi's Dynamic Optimal Countermeasure Selection for Intrusion Response System [7]. Shameli-Sendi acknowledged the static nature of current cost-benefit analysis parameters and introduced a number of variables which can be computed at attack time such administrative cost and incompatibility cost. The latter was updated dynamically based on the number of firewall rules currently in use. However, there is still an underlying reliance on offline modelling by security experts for an Attack Defence Tree (ADT) and a Service Dependency Graph (SDG).

With the development of Deep Learning and the emergence of powerful RL algorithms, very recent work on IRS has shifted to investigating the use of machine learning. In 2018, Li et al. applied a Q-Learning approach to selecting optimal Security Function Chaining (SFC) for Software Defined Networks (SDN) [8].

While this approach demonstrates the applicability of a RL solution to IR, the practical application is limited. A Markov Decision Process (MDP) is defined with a discrete state space of 101. It maintains some human interaction in that the security performance of each SFC, the actions in this case, are scored by security experts. The environment simulation is highly abstract which is therefore unable to effectively represent real world performance. A resource unit is used to represent a combination of compute resource. In our work we seek to build on this by representing real impact on resource through virtual machines.

Zolotukhin et al. recently proposed an interesting approach to IRS in a Software-defined Networking (SDN) domain [12]. The authors investigate two Model-free algorithms, namely Deep Q-Network (DQN) and Proximal Policy Optimisation (PPO) when adjusting SDN flows in response to an attack. Results are promising for both DQN and PPO, which further motivates this work. The work investigates singular response actions to single-step attacks, namely brute-forcing, Domain Name System (DNS) tunnelling and Distributed Denial of Service (DDoS). We seek to build on this work by looking at multi-stage attacks and investigating the performance of an IRS which in response can take multiple response actions.

In IRS research, the state space explosion problem is one which often limits the scaling of proposed systems to large realistic networks, comprehensively modelled approaches in particular. Acknowledging this problem, Iannucci et al. propose an interesting approach which automatically generates an adapted MDP which shifts the cause of the state space explosion problem from the size of the defended network to the scope of the attack [10]. This is an interesting approach, and we appreciate the significance of addressing the issues with modelling. However, with attacker unpredictability and a need to re-compute defensive planning with even a very minor change to the environment, there is a limit to the effectiveness to this approach when outside of ideal conditions. Hence we further justify a shift to a Model-free reinforcement learning approach.

Miehling et al. provide an interesting approach which acknowledges the stochastic nature of the attack in response planning [13]. Hence, they use a POMDP to model the attack state and experimentation is completed on a simulated and small Bayesian Attack Graph (BAG). As the entirety of the attack state is unknown the response policy incorporates a belief system based on previous experience and probabilities of future states. We appreciate this approach in dealing with the potential randomness of non-trivial attacks, however in their conclusion the authors acknowledge that this approach also suffers from a scalability problem due to state space explosion. Therefore, we believe we can build on with our proposed Model-free approach.

Gonzalez-Granadillo et al. propose a novel approach to optimal countermeasure selection in [14]. A volume-based alternative to attack-surface modelling. Indeed, this provides an effective way to view how multiple countermeasures combine against the scope of an attack.

One of the most related and interesting works was produced by Iannucci et al. [11]. Iannucci et al. accurately frames the problem with current model-based IRS and how they are quickly becoming obsolete due to the change in network structure and regular host configuration changes. Iannucci implements a Model-free approach, using Deep Q-Learning. A software simulation is developed which replicates a model of the system which the IRS is intended to protect. Upon training on the simulated model, the IRS with the trained RL agent can be deployed on the live system. The work investigates the performance of Deep Q-Learning when the environment and action spaces are changed, such as the addition of a new response. A main comparison here to the work proposed in this paper is the realism of the environment state space. In order to investigate performance through a significantly large number of episodes, Iannucci et al. represented the state attributes as Boolean values and therefore changes in state were near instant, leading to very minimal time per episode. Although Iannucci et al. consider a scenario that successfully represents attack state, it could be argued that the representation of the network state using Boolean states introduces abstractions that may result in loss of information compared to a real attack scenario. However, we believe that at present, fully simulated approaches are best suited to investigate these events through generalisation based on their ability to complete significantly more episodes in a shorter period.

Table 1 provides a summary of key IRS work and their attributes. The Experimentation column describes the method used to evaluate the proposed IRS, with Virtualised referring to the use of virtual machines and Simulated referring to experiments which are entirely encapsulated within code. Within the Testbed Scale column, small refers to less than 10 nodes, medium is greater than 10 and less than 20, and large refers to greater than 20. The IRS proposed in this work builds on previous work, particularly [10], in three main areas based on the findings illustrated in Table 1: evaluation of response cost based on real environmental performance changes as opposed to predefined estimations, the used of a virtualised experimentation environment over fully simulated models when RL is used, and a performance comparison of positive and negative Reward functions

3. Reinforcement learning

Reinforcement Learning is a branch of machine learning in which an agent interacts with an environment and observes the effect that taking an action has on the state of the environment [15]. The change in the observed state is used by the agent to learn whether the actions taken are beneficial or detrimental to achieving its goal. This observation is parameterised in the form of a Reward value, as illustrated by Fig. 1.

At a simplified level, the target of a RL agent is to maximise the accumulated reward R gained over a number of state changes, known as a timestep. However, this is not an appropriate approach, as reward r, gained at later timesteps is less likely to occur, therefore it must be discounted. This is typically known as the discount factor, $\gamma \in [0,1]$, and is adjusted accordingly to favor short-term or long-term rewards, (1).

$$R = \sum_{k=0}^{t} \gamma^k r_t + k + 1 \tag{1}$$

Early approaches modelled a reinforcement learning task as a Markov Decision Process, which provided a mathematical framework for representing the stochastic nature of a RL agents action choice [16].

An MDP is represented as a 5-tuple: (S, A, P_a, R_a, γ) where S represents the set of all possible states, referred to as the state space. A is the complete set of possible actions, the action space can also be referred to as A_s, which is the set of actions possible in a given state s. P_a is the probability that taking an action in a given state S will lead to the state

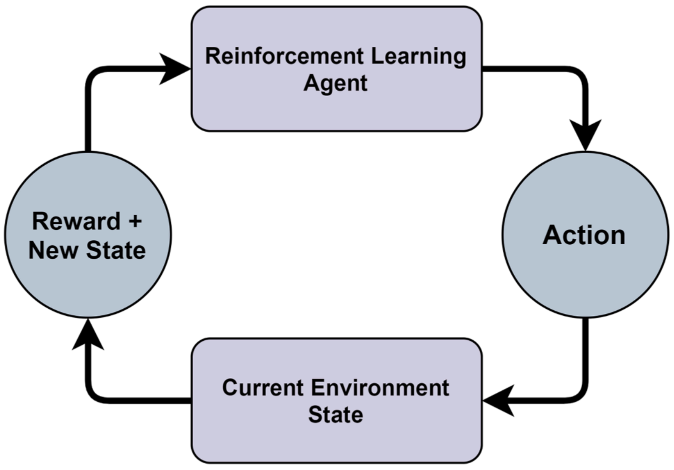

Fig. 1. Reinforcement learning model.

Table 1
Comparison of related work.

IRS	Environment	Response Selection Method	Experimentation	Testbed Scale	Response Cost Evaluation	Response Benefit Evaluation
ADEPTS [5]	Attack Graph	Benefit - Cost	Physical	Small	Static predefined variable	Dynamic
NICE [6]	Attack Graph	Benefit / Cost	Virtualised	Small	Static predefined variable	Dynamic
ORCEF [4]	Attack Graph	MOOP	Virtualised	Medium	Dynamic	Manual
RRE [9]	Attack-Response Tree	Game-Theory	Simulated	Small	Static Predefined variable	Dynamic
Dynamic Optimal Countermeasure Selection [7]	Attack-Defence Tree	MOOP	Virtualised	Small	Dynamic	Dynamic
Auto Selection of Security Service Function Chaining [8]	MDP	Q-Learning	Simulated	Medium	Dynamic	Dynamic
A Model-Integrated Approach to Designing Self-Protecting Systems [10]	Reduced MDP	Dynamic Programming	Virtualised	Small	Static Predefined variable	Dynamic
A hybrid Model-free approach for near optimal IR [11]	Model-free	Policy Optimisation	Simulated	Small	Static Predefined variable	Dynamic
A Model-free Approach to Intrusion Response Systems	Model-free	Policy Optimisation	Virtualised	Small	Dynamic	Dynamic

S', (2). R_a is the reward gained during the transition between states, given action *a*. A timestep is denoted by *t*.

$$P_a(S,\ S') = P(S_{t+1} = S' | S_t = s,\ a_t = a) \tag{2}$$

If an environment can be modelled as a complete MDP, it can be used to find the optimal policy π, which defines the mapping of actions to states which maximise the total reward gained from the start state to end state. This is known as model-based RL. Dynamic Programming provides methods to compute the optimal policy, specifically a Value Iteration formula in which a random initial value *V(s)* is defined and iteratively improved until it converges at the optimal value function *V*(s)*, (3). From here the optimal policy *π*(s)* can be determined in (4) as a single policy iteration.

$$V^*(s) = max_a\{\sum_{s'} P_a(s)(R_a(s'|s,\ a) + \gamma V(s'))\} \tag{3}$$

$$\pi^*(s) = \text{arg}max_a\{\sum_{s'} P_a(s)(R_a(s'|s,\ a) + \gamma V_i(s'))\} \tag{4}$$

However, determining the optimal policy when using MDPs can become significantly difficult as the complexity of the environment increases. An increase in the total possible states and the available actions can dramatically increase the computational complexity of finding the optimal policy. Furthermore, the ability to successfully model a problem as an MDP is a significant challenge. Outside of simplified game scenarios such as checkers, where all possible states of the environment can be seen at any stage, it is rare to have a real-world problem that can be completely modelled as an MDP.

3.1. Model-free reinforcement learning

Model-based reinforcement learning queries a predictive model of the environment to determine the consequences of taking an action. However as this modelled environment grows, the memory and computational resource needed to maintain this model grows exponentially. Furthermore, depending on the ranges of variables within an environment, it may not be feasible to effectively define a state.

Understanding the limitations of model-based RL and acknowledging the potential benefits of RL solutions to real-world problems led to the development of effective Model-free approaches. With Model-free RL, there are no defined state transition probabilities and rewards. Instead the agent learns the impact an action has on the environment through direct interaction, often referred to as a trial and error. However, as a Model-free environment can potentially have continuous state and action spaces, it is not as simple as iteratively trying all possible state-action pairs. For example, in order to allow the RL Agent to converge towards an optimal policy without overlooking potentially high rewarding state-action values, it is important to find an effective balance between exploration of the state space and exploitation of previous experience. Several hyperparameters can be used to manipulate the balance between exploration and exploitation, for example, in the popular Model-free approach Q-learning, a learning rate, $\alpha \in [0, 1]$ can be defined to control how much the agent learns from its new experiences. The closer α to 0, the more weight is placed on previous knowledge, so new exploration experience is mostly ignored, essentially resulting in a learning agent which cannot learn. If α was set to 1, the agent would only use latest information and no prior knowledge would be exploited.

One of the most recent and popular contributions to Model-free RL is Proximal policy optimisation [17]. Proximal policy optimisation builds on Policy gradient (PG) methods by reducing the margin of error when updating the policy. For example, if an agent was attempting to find the optimal route to the top of a mountain, by making large adjustment to its policy, which is route in this case, it may include the possibility of stepping off the side of the mountain and falling the entire way back to the bottom. This leads to large variance in reward. With Trust-Region Policy Optimisation (TRPO), the change in policy is limited using a KL-Divergence method [18]. KL-Divergence is used to measure the difference between two data distributions and can be applied to Policy Optimisation to ensure smoother learning. However, this second level of optimisation adds significant computation complexity. PPO introduces a method of clipping the surrogate objective which provides the same benefits of TRPO but removes the secondary optimisation by building it into the primary PG approach. PPO also offers several other benefits which make it a very good general approach for solving real-world RL problems. It is relatively easy to implement and allows flexibility when adjusting hyperparameters which tune the underlying Neural Network which represents the policy.

3.2. Reward

A RL agent requires feedback to determine the impact of the action it takes. This feedback is referred to as the reward and can be intrinsic, typically in model-based learning, or more commonly extrinsic, where the reward is directly determined from the updated state values after an action. In Model-free RL, the agent takes an action based on its current policy, observes the new state of the environment and then determines the reward by passing the updated state parameters to a function which evaluates them based on predefined targets. For example, if one considers a RL problem based on training a car to complete a lap of race circuit. The circuit can be broken down into several very small sections and at each section there are a number of observations which can be made about the environment, such as the bearing of the car, the speed of the car and the distance to the finish line. These observations make up the current state of the environment and are used to compute the reward. The reward can be heavily goal oriented such as only returning a reward if the distance to the finish line is zero, indicating the completion of a lap. To more effectively control the performance of the car, rewarding or penalising the car based on its current speed can result in a quicker lap time. However, it is important to consider the difficulty the car will have staying on the track or avoiding a collision at a higher speed, which in turn may significantly increase the time it takes for the agent to learn to complete a full lap. Evidently, it is necessary to carefully define the function for computing the extrinsic reward, as a poorly crafted function can result in inability to converge towards an optimal policy or significantly increase the required training time.

4. Methodology

A significant motivation for this work was to uncover an approach to IRS which moves away from static and manual attack modelling. Realistically, the defender is never certain as to the motivation, skill and behavior of a malicious actor. Even with extensive security analysis and pen-testing, it is impossible to determine that a sufficiently proficient attacker will not discover vulnerabilities or misconfigurations that were overlooked by the defender. Similarly, IRS which take a modelling approach often define subjective values for important measures such as potential damage caused by an attack and the cost to normal network behavior caused by a response [19]. To realise an IRS which can evaluate its performance based on real values derived from a live network, this work defines Intrusion Response as a RL problem. A PPO Agent takes the role of the decision maker, choosing response actions and observing the performance and impact of the response though monitoring the state of a simulated network environment.

4.1. Testbed environment

A network environment was created using the Graphical Network Simulator 3 (GNS3). This network consists of two subnets, a node acting as the external local area network (LAN) and a supervisory network, shown in Fig. 2.

An OpenBSD virtual machine was configured to act as a router and a

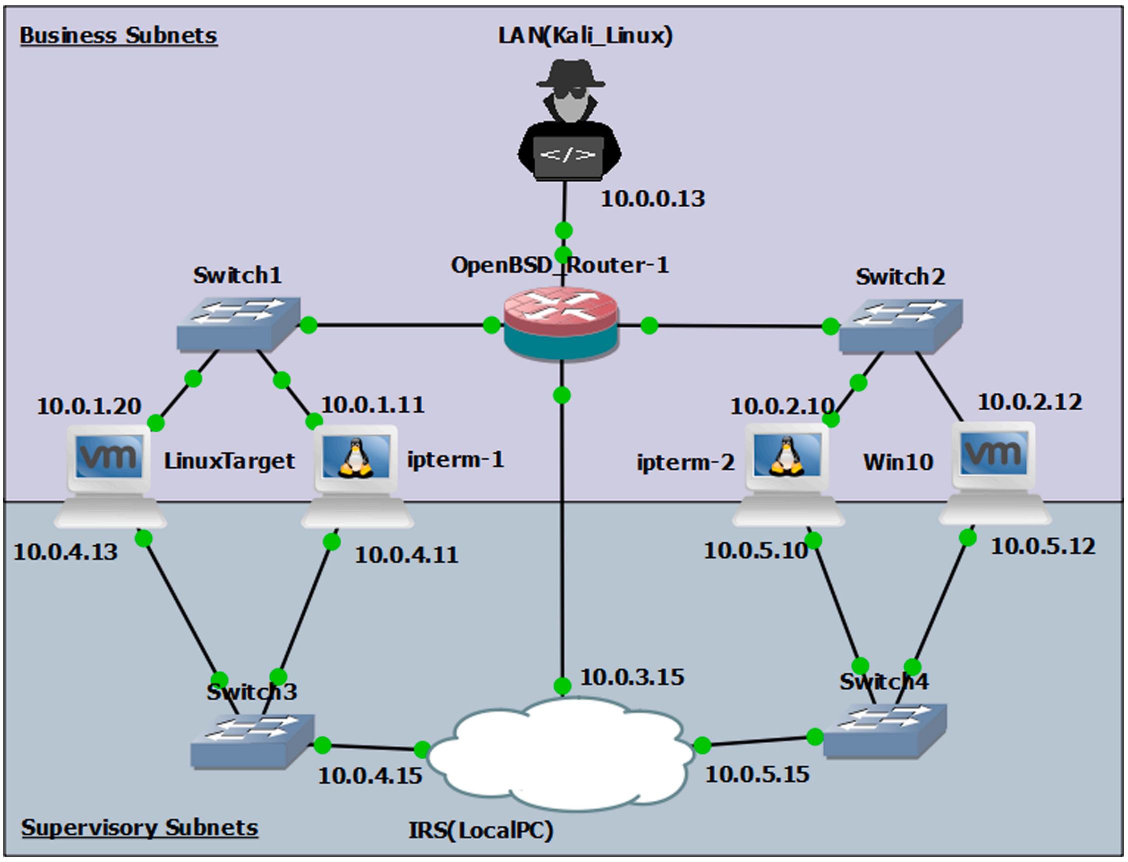

Fig. 2. GNS3 network testbed.

firewall. In its original state, the OpenBSD Packet Filter (pf) firewall is initially configured to prevent traffic from the LAN reaching the 10.0.2.0/24 subnet. For the purpose of recreating the attacker, a Kali Linux machine denotes the external LAN. Subnet 10.0.1.0/24 consists of a Metasploitable virtual machine [20] and an IpTerm node which is a Debian based lightweight networking toolbox. Subnet 10.0.2.0/24 consists of another IpTerm node and a Windows 10 virtual machine. The virtual machines are hosted using VMWare 15 Pro. VirtualBox version 6.0 was used originally, however it was discovered that VirtualBox encountered issues when attempting to restore all nodes to snapshots, which is a required function for RL environment resets. These environmental resets are a fundamental part of the RL process, therefore to accomplish this in with a real virtualised testbed instead of a simulation requires significant host control and timing.

The host hardware consists of a Ryzen 7 3700 × 8 core 16 thread CPU with a 3.6 ghz base clock. 32 gb of DDR4 RAM at 3600 MHz. Node control for the virtual machines such as shutdown, restarting and restoring to snapshots was scripted using a combination of VMWare's command line utility and CURL commands to the GNS3 network. To restore the network to the original state in order to begin the next training episode, all nodes are stopped with a CURL commands, the VMWare commands to restore the virtual machines to their snapshots is then called before finally restarting all the nodes. This process is what costs the most amount of time and significantly reduces the training speed of the RL agent. The execution of these scripts was controlled by the RL solution developed in Python v3.7.0.

Paramiko is a python package which was used to handle automated interfacing with the VMs during RL training and evaluation, as it allows SSH output to be piped back into the Python application. Tensorforce is a Python Deep RL library built on top of TensorFlow. A main benefit of Tensorforce is that it has several out of the box RL algorithms, including PPO, and each algorithm is highly configurable using the agent's constructor method. Tensorforce is also developed to be abstract, in that the algorithm used by the RL agent can be changed without significant redevelopment. This is beneficial for performance comparisons.

Initial motivation behind the development of this testbed was to develop a traditional yet straightforward business LAN, an approach which has been taken regularly in IRS research. The use of full virtual machines has several advantages over simulated approaches. In particular our IRS is able to interact with the machines over SSH and collect as much information as the machine's operating systems can provide. This is particularly beneficial for accurate and real-time state parameter collection. Indeed there are some limitations to our testbed in terms of realism and scalability, in particular the memory and requirements to run several full virtual machines simultaneously, which limits the ability to replicate a large scale network. Similarly, with the development of static testbeds rather than a network simulation, network traffic simulation is a complex process and is achieve artificially in our testbed through ICMP packets.

4.2. *Reinforcement learning requirements*

The goal of the RL agent is to find the action or combination of actions which return the most reward from the environment. Repetitive training establishes an effective success or failure measure of a response, with the most successful response becoming the best policy for when the attack occurs in the live environment. This adoption of this approach allows an IRS to continually train using a range of different attack scenarios, changing network topologies and adopt or remove possible actions.

In order to efficiently integrate a Reinforcement learning solution to an intrusion response problem, there are three high level challenges which have not previously been identified in related work, outlined in Fig. 3. Firstly, a translation of a network environment to a RL

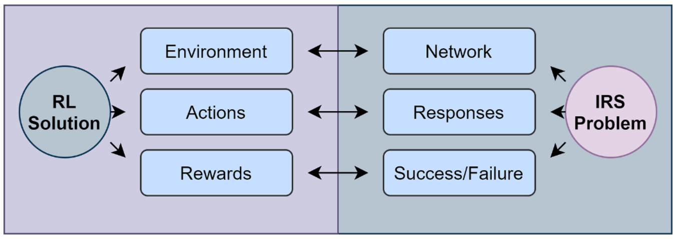

Fig. 3. Mapping RL to IRS.

environment which can be represented as a continuous state space. In model-based IRS approaches, there is a benefit of a finite state space, this allows all system states to be fully understood and reward gained when moving between states can be predefined. However, the ability of models to represent the complexities of a network under attack in a discrete state-space is extremely challenging and resource intensive. Hence, we believe a continuous state space is better suited to represent network environments. Secondly, a range of IRS responses which can be made available as a discrete action space and can allow automated execution in the environment. This component of an IRS can be quite challenging in terms of unique application to custom networks, for example the scripting to automate a response action in one environment may be completely different in another environment depending on the types of devices, operating systems and software. Where possible, the response actions available to the agent should be low level and atomic. This allows the agent to discover unique and effective combinations to a broader range of attacks, which was indeed outlined in [21]. Finally, a method of performance evaluation to monitor the success or failure of a response and translate that into a reward for the RL agent. This requirement is achieved through the custom reward functions which encourage the agent to prioritise stopping the attack, such as providing a high reward when the environment state includes no alerts.

The first requirement, translating a real network to a RL environment, involves the realization of five functions, defined in Table 2. These are the main required functions, however, several other auxiliary functions are necessary in order to efficiently interact with the environment, such as the environment output data handlers. An interesting function to note here is the episode termination function, in this work we used a combination of the termination function and the initial configuration of the Agent. Tensorforce allows us to define the maximum number of timesteps allowed per episode, which we set to 10. On top of this we built in a custom termination function which terminated the episode if more than 2 min and 30 s had elapsed. This primary purpose of the custom terminal function was to prevent hangs and speed up training.

Table 2
Function requirements for RL environment.

Function	Description	Method
reset()	Reset the network to a starting state. For optimal learning, this must be a quick process.	Shutdown node VMs and restore to snapshots.
execute()	Upon the RL agents action selection, execute the response in the live environment.	Use monitoring network to access nodes and execute predefined response action script.
getState()	Observe the current environmental attributes and return as state object.	Use monitoring network to access nodes and gather information such as traffic statistics.
isTerminal ()	Evaluate the current state of the environment against criteria for ending the episode.	Execute the getState() function and pass state attributes as parameters.
getReward ()	Provide a means of incentivizing the RL agent through defining rewarding or penalising logic.	Execute the getState() function and pass state attributes as parameters.

When executing an action and subsequently observing the next state using the *getState* function, an agent needs to be able to efficiently obtain updated environment information. This process occurs after every action execution, which can be several times within a single training episode. The repetitive and dynamic nature of this approach requires an investigation into network analysis tools which can provide accurate situational awareness in near real-time. For example, if the RL agent decides to add a rule to the firewall, in order to evaluate the impact of this on the network, a method which retrieves network traffic statistics such as dropped packets, errors and unresponsive nodes is required. This is also relevant to when an attacker makes a move, in Model-free approaches, there is no determined nature to the attacker's moves as you would find in game theoretic approaches, where the defender selects an action and the attacker follows with a reaction [9]. Instead, to best represent the somewhat stochastic nature of a realistic attacker, the attack scenario runs simultaneously to the RL agent. When the attacker makes a malicious action such as exploiting a vulnerability, launching a Denial of Service (DOS) or even disabling a node, the agent will observe the compromised environment state and receive a low reward. This will initiate the sequence of actions in which the RL agent will attempt to stop the attack.

The second requirement is the translation of typical IRS actions to an action space available to the reinforcement learning environment. The countermeasures made available to the RL agent in this work were defined based on surveying a range of proposed IRS and filtering by applicability to the proposed environment. The collection of countermeasures is defined as Python functions in Table 3.

Actions selected by the RL agent are executed immediately in the

Table 3
Response actions.

Action ID	Function	Description
0	block_all_traffic()	Add block-all rule to the router firewall and restart service
1	reset_host(windows)	Two curl requests to GNS3, shut down and start VM
2	reset_host(ipterm1)	Two curl requests to GNS3, shut down and start VM
3	reset_host(ipterm2)	Two curl requests to GNS3, shut down and start VM
4	shutdown_host(windows)	Single CURL shut down request to GNS3
5	shutdown_host(ipterm1)	Single CURL shut down request to GNS3
6	shutdown_host(ipterm2)	Single CURL shut down request to GNS3
7	disable_interface(windows, ethernet0)	Ipconfig down command
8	disable_interface(router, em0)	Ifconfig down command
9	disable_interface(router, em1)	Ifconfig down command
10	disable_interface(router, em2)	Ifconfig down command
11	enable_interface(windows, ethernet0)	Ipconfig up command
12	enable_interface (initalTarget, eth0)	Ifconfig up command
13	enable_interface(router, em0)	Ifconfig up command
14	enable_interface(router, em1)	Ifconfig up command
15	enable_interface(router, em2)	Ifconfig up command
16	block_ip(windows)	Append singular block rule to the router firewall and restart service
17	block_ip(ubuntu)	Append singular block rule to the router firewall and restart service
18	block_ip(ipterm1)	Append singular block rule to the router firewall and restart service
19	block_ip(ipterm2)	Append singular block rule to the router firewall and restart service
20	Take no action	The RL agent idles before reobserving the state

environment, therefore the functionality to automatically apply the action must be created. With a discrete action space, an effective approach to this is to build the responses as functions and provide a switcher method to the agent. The RL agent selects the integer value between 0 and the length of the action based on its current policy. This integer is then passed to the execute function defined in Table 2 which finds and calls the associated python function. While the action space in this work requires a degree of tailoring to the nodes in the network, an ideal IRS would use an action space which can be universal to a node within the network.

The performance of a response in IRS is an area which is often overlooked, some approaches score the response based on whether or not the alert which triggered the response is still present within the network after a set period of time [22]. In this RL approach, the performance of a response is directly evaluated prior to being deployed in the live network. To achieve the ability to evaluate the reward based on the current state of the network, several high-level and low-level parameters were identified and collected during the agent's observation of the environment, these parameters are outlined in Table 4.

Selection of state parameters is an important aspect of developing a Model-free reinforcement learning solution as they determine the extent of agent's understanding of the environment. The parameters in Table 4 are selected based on the availability of information present within the defended environment and its perceived relevance.

Overlooking critical aspects of the network could indeed result in the inability of the agent to understand the impacts of its actions or the attacker's actions. The collection and use of real environmental variables in this work instead of simulated approaches added complexity but also provided the agent with a true representation of how a live network will respond to actions.

IRS are generally somewhat reliant on the ability of an intrusion detection system to successfully detect an attack. However, the use of the state parameters in Table 4 provides further observation of the environment which can result in a response even if the IDS fails to identify the attack. In particular, they can accurately reflect the damage being caused by an attack. It is however important to note here that we also need to consider that benign actions such as a human legitimately powering off a node should not trigger a response. Therefore if this ability to act without a detected alert is provided, the construction of the reward function which defines what we consider a malicious network state needs to be carefully considered.

Table 4
State parameters.

Parameters	Description	Variable Type
Nodes up	The number of states which are responding to ICMP Pings.	Integer
Nodes down	The number of states which are not responding to ICMP Pings.	Integer
Alerts / indicators of compromise	Alerts occurred since previous action execution. Alert log is cleared after each step.	Returns a list of alert class objects containing specific alert details, i. e. src, dst, type
Packets errors	Number of packet errors or dropped packets from benign traffic since previous state observation.	Integer
Elapsed time	Time since episode started. Used as incentivization and terminal state check.	Double

4.3. Attack scenarios

To provide the RL agent the scope to decide several factors such as the optimal time of response and optimal location to apply responses, a multi-stage attack has been scripted, illustrated as a flow chart in Fig. 4. The attacker's goal is to extract information from the Windows 10 machine. However, the firewall prevents the external LAN communicating with the subnet containing the Windows machine.

LAN traffic can access the 10.0.1.0/24 subnet, as it is representing the external facing business servers such as the webserver. The attacker exploits a publicly known vulnerability in 10.0.1.20, specifically a backdoor in a file transfer protocol, CVE-2011-2523. From here the attacker escalates privileges and uses the compromised machine as a pivot by manipulating the routing table to redirect traffic from the attacker to the Windows machine. After successfully completing the pivot, the attacker can reach the Windows machine but cannot access it, therefore an internal brute-force SSH is attempted in fast mode using a subset of the popular RockYou dataset. Upon successful completion of the brute-force, the attacker locates the sensitive data located within a text file on the windows machine and extracts it, in the form of a string.

Due to the iterative nature of RL, it is necessary to script the attack to be fully automated and repetitive. All stages of the attack execute sequentially unless halted by the IRS, this decoupling of the attacker and defender compared to game theoretic approaches provides an element of realism in that an attack will not wait for the defender to make a move. Consequently, the attacker is free to take an available action at any time until a terminal state is reached, at which point the environment will reset. Once the environment reset function is called by the RL algorithm, all virtual machines restore to a previous state and reboot, including the attacker virtual machine at 10.0.0.13. The attack script was developed using Metasploit custom scripting and added to a batch file which is executed upon startup of the Kali virtual machine. The ability of the attack scenario to reach the end state, which is the successful extraction of sensitive data, is determined by the success or failure of the RL agent to halt the attack.

To provide a comparison for the performance of the IRS and assess the applicability of the proposed approach to a different problem, a second multi-stage attack scenario was created, illustrated in Fig. 5. The attacker now exploits a different vulnerability on the externally facing machine. Like Scenario 1, privilege escalation is achieved, and a pivot is created. This time the attacker targets the router and firewall, using the compromised machine to launch a proxy SSH brute-force. If the attacker succeeds, the next step is to disable the Packet Filter firewall. A third step has been added to this attack, as the firewall has now been disabled, the attacker can directly access the poorly configured Windows machine and performs an anonymous FTP login and extracts sensitive data.

4.4. Reward

As discussed in Section 3, the RL agent must be able to compute an extrinsic reward based on observing the current state of the network. While this component of RL is imperative to enabling the RL to perform well and achieve its goal, its definition and explanation is overlooked in related work. The function for computing this reward is defined in Algorithm 2. The function requires two input parameters, first the state, an object variable with 5 properties which describe the state, from Table 3. The second input parameter is the total number of timesteps taken by the RL agent during the current episode. A timestep is defined as a complete iteration, in which an action is selected, and the new state and reward are received.

Weights are used within Algorithm 1 to determine the significance of the individual goals of the agent, these are selected by the developer based on the importance of the parameters. For example, keeping the number of alerts less than 1 is very important and therefore the reward gained is multiplied by 1.5.

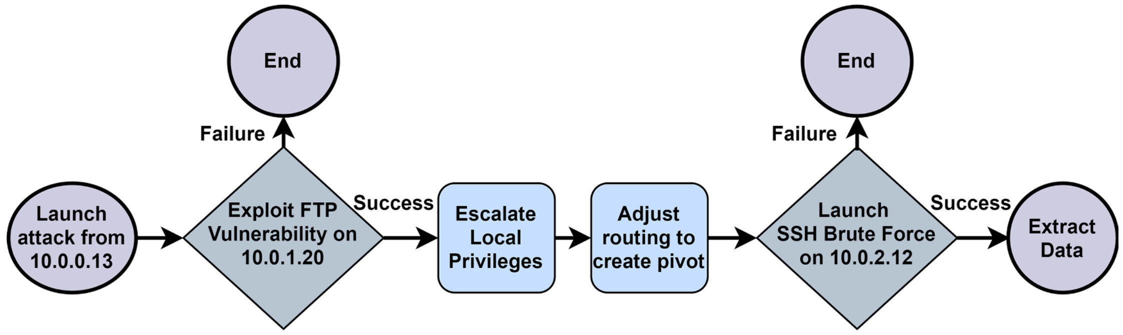
End
Failure
Launch attack from 10.0.0.13
Exploit FTP Vulnerability on 10.0.1.20
Success
Escalate Local Privileges
Adjust routing to create pivot
End
Failure
Launch SSH Brute Force on 10.0.2.12
Success
Extract Data

Fig. 4. Multi-stage attack scenario 1 flowchart.

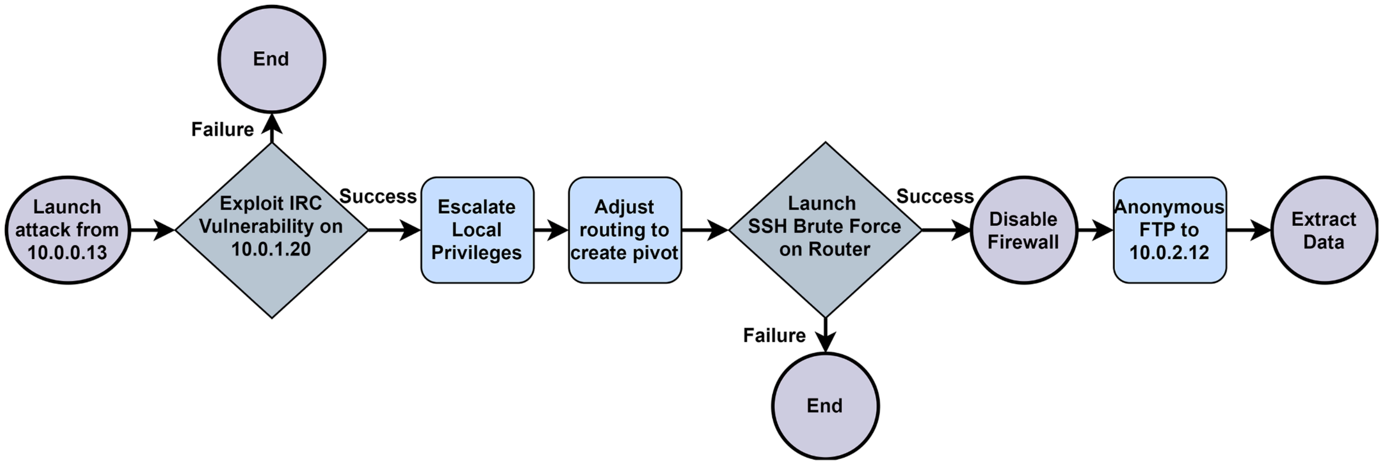
End
Failure
Launch attack from 10.0.0.13
Exploit IRC Vulnerability on 10.0.1.20
Success
Escalate Local Privileges
Adjust routing to create pivot
Launch SSH Brute Force on Router
Success
Disable Firewall
Anonymous FTP to 10.0.2.12
Extract Data
Failure
End

Fig. 5. Multi-stage attack scenario 2 flowchart.

Algorithm 1. Reward evaluation

```
Require: state, steps
  reward = state.nodesUp * 50
  reward = reward – (state.packetErrors)
  If state.alerts < 1 Then
  reward = reward * 1.5
  Else
  reward = reward * 0.4
  If state.nodesUp = 5 And state.alerts = 0 Then
  reward = reward * 4
  reward = (1 - (steps / 10)) * reward
  Return reward
```

The authors of [11] define a reward function as a weighted time and cost penalty. However, the values for the time and cost of an action appear to be predefined and not determined based on the impact of deployment within the environment. A standard approach to defining this function is to provide a heavy reward whenever the agent reaches a state which represents the successful achievement of its overall goal. In this case that has been defined as having all nodes up and responding, while also having no new alerts. This would represent the successful halting of the attack and the return to a normal network operation. Other less impactful reward modifiers can be used to control the performance of the agent, for example, encouraging reaching the goal quicker, implemented here using a reward coefficient between 0 and 1 based on the total number of steps taken, with more steps resulting in a coefficient closer to 0. During the intialisation of the agent it is possible to define the maximum number of timesteps allowed, which based on the length and complexity of an episode, has been set to 5. This means that the agent can select and deploy 5 responses during each training episode. While the agent may successfully halt the attack in 1 or 2 steps, the option to take additional actions is provided to investigate whether the agent can find an optimal response, namely stopping the attack and then selecting the 'take no action' response for the remaining timesteps. Another interesting approach is to use a negative reward, for example instead of rewarding the agent for selecting an action which leads to an improved state, the agent begins which a large reward which decreases based on negative impact of its action.

4.5. *Proposed framework*

The overall system framework is illustrated in Fig. 6. This diagram illustrates the loops present within a singular training episode. A number of training episodes, specified by the batch size, are completed before the agent's policy is updated, meaning the experience gained is used to change the probabilities of taking actions in a particular state.

Three variables are collected from the testbed during each timestep. In our experimentation, node information includes a status which reflects whether the node is up and responding to packets, however this information can be extended. Traffic statistics includes the number of packets errors at each timestep and is collected using the command line tool nstat. Snort alerts are the result of an automated parsing of a snort alert log, this log is cleared at the end of each episode.

While this structure identifies the general interaction between a RL agent and an IRS environment, the interfaces between the attacker, environment and RL agent all require individual customisation.

Specifically, the deployment of a custom attack scenario and the Python functions for retrieving state variables from the nodes within the network testbed. This framework identifies the key functionality required to implement RL in an IRS scenario while domain and goal specific information can be easily swapped out such as the attack type and the situational information collection from the network testbed.

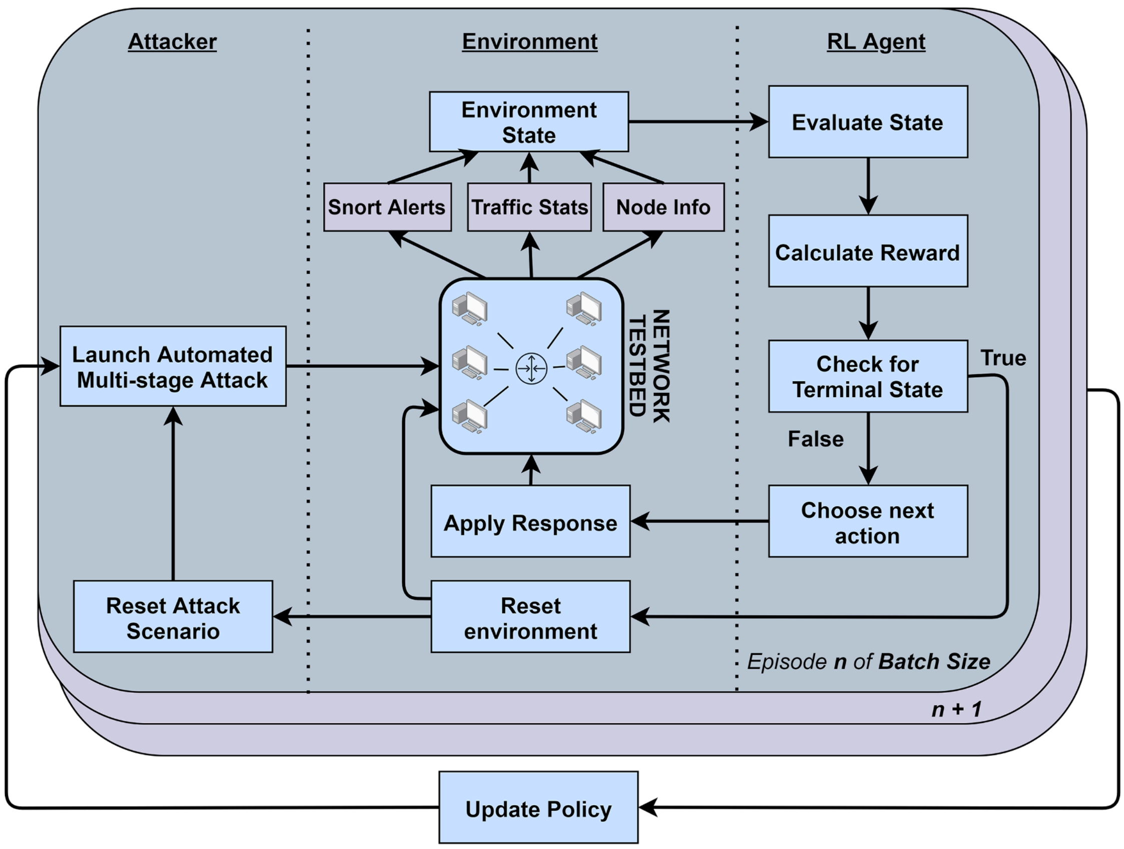

Fig. 6. RL training framework.

5. Experimentation

With the goal of simulating a realistic network environment, the use of fully virtualized machines resulted in significant boot times. Therefore, resetting the RL Environment is a time-consuming process. Subsequently, training time is longer than typical abstract RL experiments, with an average episode time of ~140 s compared to ~73 ms in [11]. Results show that despite fewer training episodes due to longer episode time, the agent is able to determine an effective response to the attack scenario based on the action provided. The configuration of PPO hyperparameters provided a means to adapt the aggressiveness of learning to compensate for the small number of episodes. Fig. 7 shows the reward gain during 50 training episodes and 10 evaluation episodes when using the TensorForce default agent configuration, Algorithm 1 and with the batch size adjusted to 7.

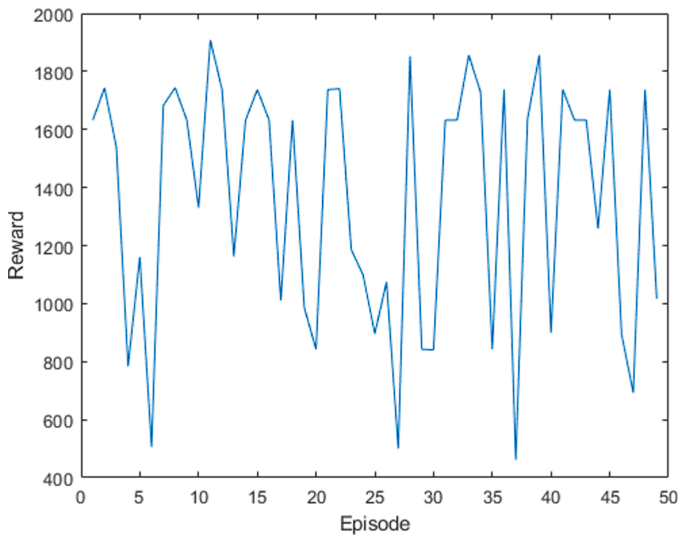

Fig. 7. Batch size 7, default PPO settings.

Fig. 7 also demonstrates the differing approach to reward gain when compared to traditional RL approaches which begin with a very low reward from the initial episode and has a typically positive gradient. Due to Algorithm 2, the agent is heavily rewarded when the network is in a benign state, therefore by taking random actions from the outset, the RL agent is still able to achieve a high reward if the action does not negatively impact the environment. Significant dips in reward can be seen as the successful completion of the multistage attack. As the default configuration of the TensorForce PPO agent was with a learning rate of 1e-3, Fig. 7 understandably demonstrates an inability to converge over a small number of training episodes.

5.1. *Positively rewarded vs negatively rewarded*

The performance of the RL agent and its ability to move towards an optimal policy is significantly influenced by the development of the reward function. There are no clear guidelines as to how to define the reward function, as it is typically unique to the environment in which the RL agent operates. It also depends on what the agent is attempting to achieve. It is possible to define the function with an expectation of how the agent should perform, however optimising for maximal performance can often require a trial-and-error approach. One main distinction that can be made is whether to use a positively rewarded or negatively rewarded function. Algorithm 1 was created as a positive reward function, increasing the reward gained whenever the agent took actions which placed the environment in desirable states. The use of either a positive or negative reward function can be seen as an optimisation technique which is highly dependent on the unique goal of the agent and the environmental state parameters. For example, in a computer game the agent can be rewarded for how far it progresses through a level, whereas in our work we are seeking to halt the progress of an attacker. Hence it is simpler to provide the agent with a high initial reward and penalise it based on how far the attacker progresses.

In contrast, Algorithm 2 defines a negative reward which penalised the RL agent based on undesirable state parameters. For example, the agent is heavily penalised for new alert logs. This encourages the RL to avoid that state in the future. The results of using Algorithm 2 demonstrates that the agent performs better with the penalisation and converges towards an optimal policy, shown in Fig. 8. Defining update frequency as 1 clearly has a positive effect on the convergence of the policy. This is a positive example of the benefits of tailoring RL algorithms to suit the problem.

Algorithm 2. Negative reward evaluation

```
Require: state, steps
  reward = 3000
  If state.alerts >= 1 Then
  reward = reward – (state.alerts * 500)
  If state.nodesDown >= 1 Then
  reward = reward – (state.nodesDown * 500)
  If state.packetErrors >= 1 Then
  reward = reward – (state.packetErrors * 50)
  reward = reward – (steps * 150)
  If reward < 0 Then
  reward = 0
  Return reward
```

With very long run times resulting in few training episodes, a more frequent update of the policy is required. The training illustrated by Fig. 9 has a larger batch size and update frequency, while this is still relatively small compared to simulated experiments which can run much more episodes, it still demonstrates poor convergance. However, due to the nature of the problem the agent is trying to solve, this training still finds a policy in a small number of training episodes which is able to succesfully stop the attack on all occasions during evaluation.

5.2. Adjusting hyperparameters

As TensorForce allows the abstract configuration of RL algorithms, it provided an opportunity to adjust hyperparameter values in order to compensate for the low number of episodes due to long environment reset times. The small number of training episodes means that the RL agent is only able to perform a low number of policy updates. We are able to somewhat facilitate this through the adjustment of agent hyper

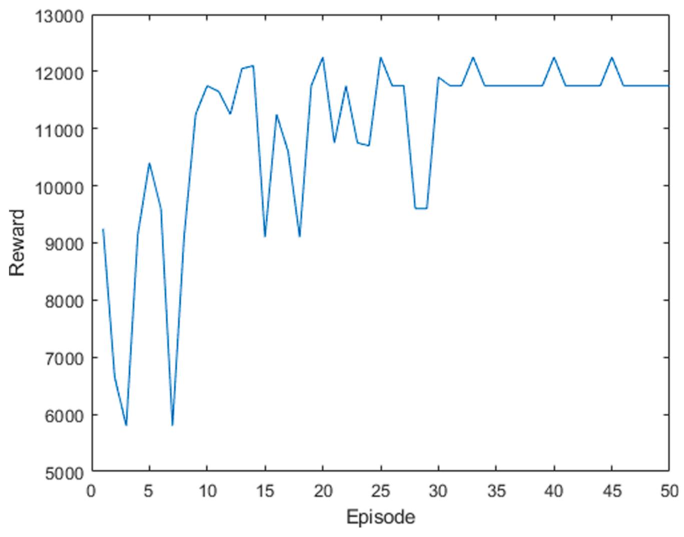

Fig. 8. Update frequency 1, negative reward function.

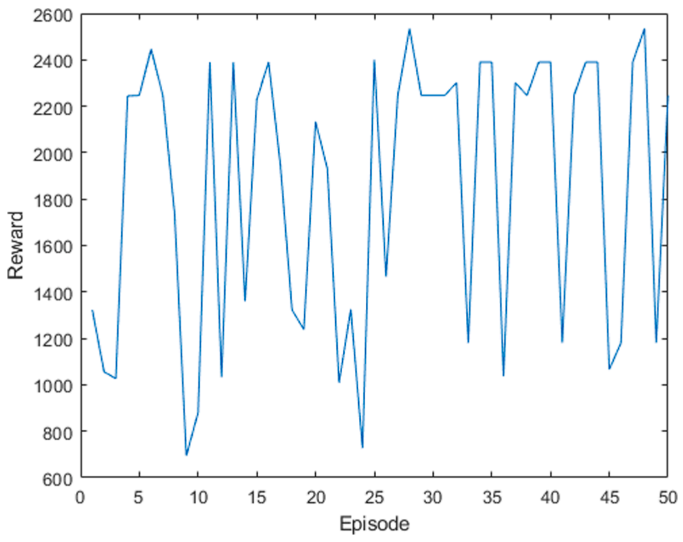

Fig. 9. Batch size 4, PPO, update frequency 4.

parameters such as using a faster learning rate and a shorter update frequency.

An initial change was to lower the batch-size from a typical size of 32. This change defines a lower number of episodes are used to update the underlying policy. Reducing the batch size can improve training time, however the results can be a lot more volatile. The next hyperparameter which was adjusted was the update frequency which defines how often the agent's policy is updated. Due to the very low number of episodes available, this was set to 4 for experimentation, seen in Fig. 9. This resulted in a higher average reward meaning a better performance during evaluation, reaffirming that despite a minimal number of episodes, it is possible to adjust RL algorithms to suit the environmental circumstances.

A description of the hyperparameters and the ranges used during experimentation to find the optimal configuration are provided in Table 5.

The range of values in Table 5 are reflective of the nature of the Intrusion Response problem addressed, i.e. with the long duration of a training episode and therefore fewer total episodes, low values were used for the batch size and update frequency. The policy, which defines the agent's understanding of how much reward is gained when taking an action in a specific state, needed to be updated frequently in order to reach a level of performance which could successfully stop all stages of the attack. Less frequent policy updates with a small number of training episodes results in more exploration than exploitation and therefore an effective policy may not be found. The optimal values were found to be 50 training episodes with a batch size of 6, update frequency of 3 and a learning rate of 1e-2.

Table 5
Hyperparameters.

Hyperparameters	Description	Experimented Range
Episode number	Defines the number of episodes of training to complete. More episodes result in longer training.	30 - 150
Batch size	The number of episodes used in a policy update to adjust the underlying neural net.	2 - 8
Update frequency	Defines the number of episodes after which a policy update occurs.	1 - 4
Learning rate	Defines how the extent to which new experience overrides previous experience.	1e-3 – 1e-1

5.3. Response performance

Fig. 10 displays the Frequency distribution of the actions selected by the RL agent when using a negative reward function. The agent avoids the harsh penalties by avoiding the selection of actions which result in a heavily penalised state, such as shutting down nodes (see actions in Table 3). It is also clear that the agent finds a method of stopping the multistage attack through the execution of action 10 or 17. However it can be noted that action 14 was selected regularly. This action executes a command which enables an interface on the router. This action was selected repeatedly following the halting of the attack using action 10, 16 or 17. Despite not being the optimal action, the agent found that it results in a minimal loss in reward as it does not have a large effect on the environment. While the selection of action 20 (no response) would be the optimal action in this case, it provides a small increase in reward. This level of optimality would likely be seen after a much larger number of training episodes, however due to the long nature of episodes in a practical environment, a tradeoff been practicality of response and training time has to be considered.

Fig. 11 demonstrates the action frequency based on 4 complete training periods, further demonstrating the identification of response actions which halt the attack with minimal cost. As indicated in Figs. 7–9, in the evaluation stage containing 10 episodes, the RL agent successfully stops the multistage attack every time.

Given its overall goal, the main performance indicator of an IRS is its ability to stop an ongoing attack. In this work the attack scripts automatically place flags on the compromised machines if the attack is successful. In Attack Scenario 1, the initial flag is placed after the attack exploits the FTP vulnerability, the second flag is placed after the attack completes the pivot, brute-forces the Windows machine and extracts the data. Fig. 12 presents a record of the flags throughout the attacks, ranging from 0 successful flags to a completely successful attack with 2 flags.

It is evident that as the training progresses the IRS becomes more adept at stopping the attack, as demonstrated by the increasing number of episodes with 0 flags created. An evaluation of the trained agent is then completed over 10 episodes, in which 0 flags were planted, resulting in an attack success of 0%. Fig. 13 demonstrates a similar performance in Scenario 2. This attack has three stages and therefore 3 flags, and the results demonstrate that at the start of training many attacks completed fully, resulting in the attacker successfully extracting information from the target system. 2 flags indicates that the IRS stopped the attack midway, upon the disabling of the firewall, and 1 flag

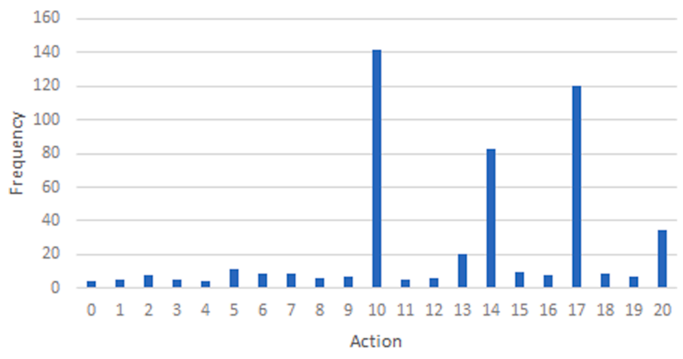

Fig. 10. Action frequency distribution with a negative reward function.

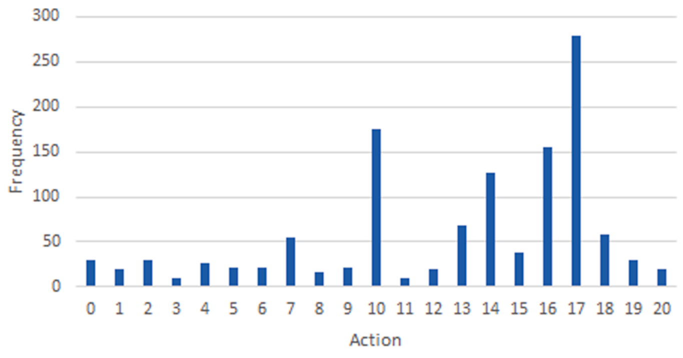

Fig. 11. 4-run combined action frequency distribution.

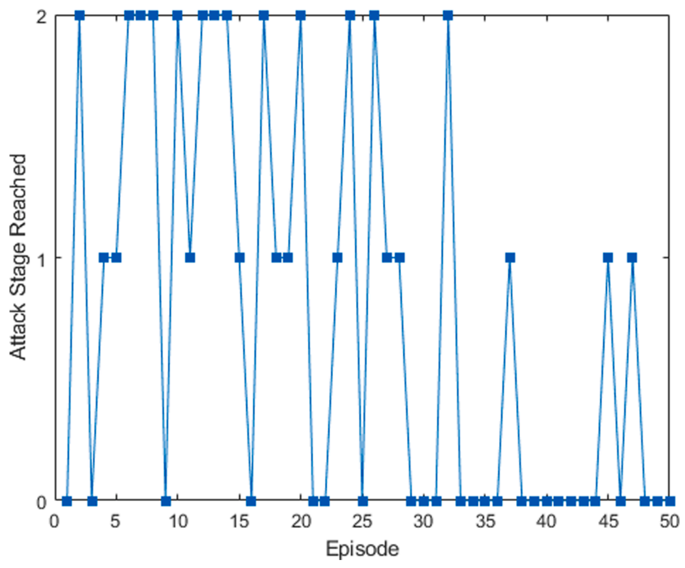

Fig. 12. Attack scenario 1, flags successfully planted by the attacker.

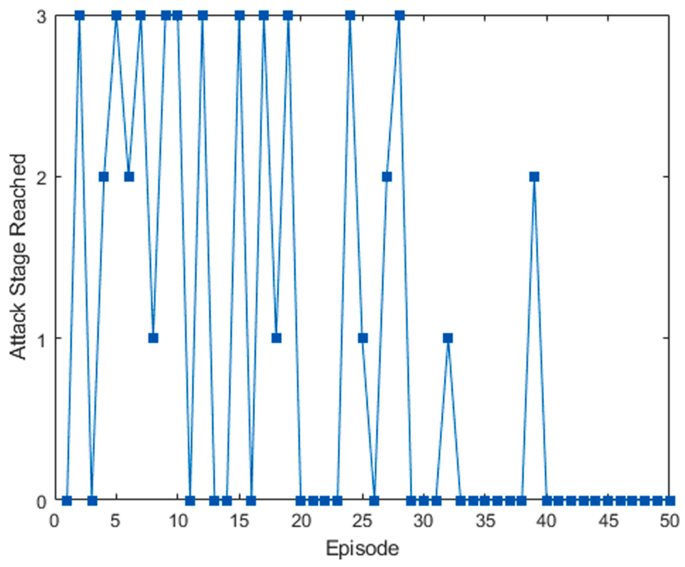

Fig. 13. Attack scenario 2, flags successfully planted by the attacker.

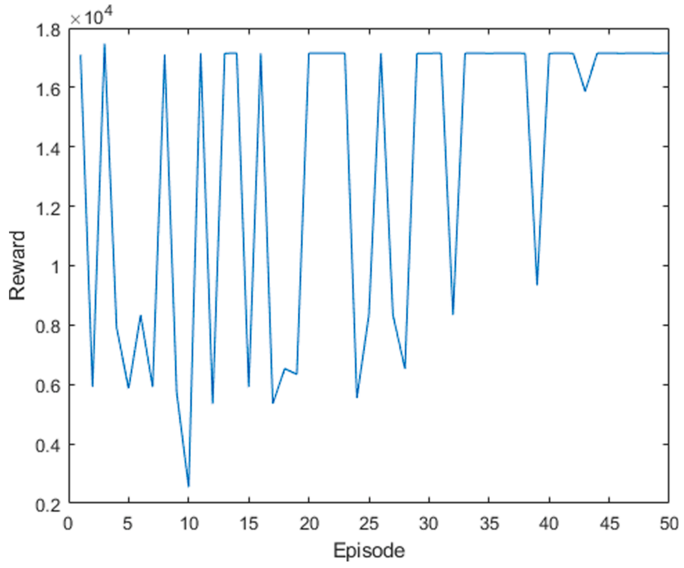

Fig. 14. Reward gained during attack scenario 2, negative reward function.

indicates the attack was stopped after the initial exploit. 0 flags indicates that the IRS, acting upon an early alert log, was able to stop the attack before the initial compromise completes.

The reward gained by the RL Agent when responding to attack Scenario 2 is illustrated in Fig. 14. The Fig. demonstrates the correlation between reward gained and the number of flags created in Fig. 13. We can observe that Fig. 14 converges towards the near optimal policy, and this reflects the occurrences of 0 flags towards the end of the training period.

A secondary goal which we sought to validate was the ability of the IRS to understand when it had stopped the attack, hence avoiding the deployment of follow up responses, which may disrupt normal network operation. The reward functions defined in Algorithms 1 and 2 were developed to incentivise this optimality. Hence for Attack Scenario 1 the optimal response is Action 10 followed by continually selecting action 20 which is the idle response. Results demonstrated that the IRS successfully found Action 10 and achieved its primary goal of stopping the attack, it only applied a fully optimal response on a small number of occasions, shown by the low frequency of Action 20. With Model-free approaches, the often non-deterministic nature of state parameters results in an inability to explicitly state the optimal reward value prior to the agent gaining real experience from the environment. For example in this work the state parameter indicating the number of packet errors is non-deterministic as it is collected at runtime using the nstat command line tool. While we suggest that outlining the optimal reward value is not an ideal performance validator for Model-free approaches, it is indeed possible to estimate a near-optimal reward. When taking the optimal actions for attack scenario 1 with algorithm 2, if we use the training average of 25 packet errors $\pm$ 3, between each call of the nstat command, we achieve an approximated optimal reward of 12,750. We suggest that the small spikes or dips in Fig. 8 represent this fluctuation in the number of errors as 1 error indeed results in a penalisation value of 50. While this is a good indicator, applied Model-free approaches may be better validated through their goal-specific performance, hence the use of attack stage monitoring which clearly indicates the successful and early halting of the attack.

6. Conclusion and future work

As a shift in paradigm from the traditional IRS approaches proposed in [4,7,23], a Model-free quantifiable performance evaluation IRS has been proposed in this work. Requirements for applying a RL solution to an Intrusion Response problem have been identified and met and a framework has been proposed to enable simplified replication on unique environments.

The need to define subjective values for significant evaluation variables such as Response cost and response benefit [19], has been removed though the use of direct experience gained from response execution in a simulated environment. In particular, the experience-based selection method used in this approach allows the evaluation of response impact based on differing circumstances. For example, in related work which define a static value for the cost and benefit of a response, that value is unchanged regardless of differing circumstance, such as the severity of the attack or the location at which the response is deployed. Whereas now the impact of a response is evaluated dynamically, the change in cost and benefit due to environment changes and previously deployed responses is taken into account. Reliance on model-based approaches which used static or semi-dynamic AGs and ADTs [7,24,25], resulting in outdated and potentially negatively impacting response selection, has been addressed with Model-free RL.

Model-free RL can identify an effective response, the correct location to deploy a response and also an effective stage of an ongoing attack to deploy a response. Building on recent RL IRS work, [11,26,27], which implement hypothetical attack scenarios, a real multi-stage attack scenario was scripted using Kali Linux and used for realistic performance evaluation. Experimentation demonstrated that a RL agent was successfully able to find an effective response and succeeded in stopping a multi-stage attack. When tuned to facilitate a small number of training episodes, PPO was able to find an effective sequence of response actions. A comparison between positively and negatively rewarded agents demonstrated that the reward gain was less volatile with Algorithm 2 however Algorithm 1 was able to determine the better response, specifically the ability to identify the no response action which is the optimal action to take when the environment is not in an attack state. Two further areas in which the proposed IRS builds on related work is the use of a large number of responses and the inclusion of a multi-stage attack scenario. The former demonstrates how a large actionable response set can be used by an automated IRS over limited or attack specific response sets, while the later adds realism to performance evaluation and investigates the ability of an IRS to defend against a complex targeted attack rather than a singular attack stage such as a Denial of Service or drive by compromise. Comparing [11] as the closest related work, this furthers the conclusion that Model-free RL is an effective approach to IRS by investigating the applicability and performance using realistic simulation with a fully virtualised testbed, it also provides a novel dynamic method of evaluating the cost and benefit of responses through general situational awareness data collected from the virtual machines.

The concept of handling false negative alerts or alerts which are rare-but-benign is a challenging topic within IRS. With Deep RL we provide an agent with a Reward Function which evaluates the current state of the environment. The current state is represented by parameters containing alert information, which indeed could be rare-but-benign. However, an interesting aspect of this approach is that the RL agent also uses environmental information such as nodes alive and packets dropped. This is then used in the decision process when selecting an optimal action. Suggesting that despite false alerts or benign activity, the proposed approach is aptly able to respond, as the RL agent is not entirely reliant on the alert information. Instead, the severity of the environment state and subsequently the response is based on a range of environmental factors. This is a concept with could be explored in detail in future work.

Scalability continues to be a limitation when it comes to evaluating the performance of IRS. Approaches which create abstract simulations can represent large scale networks but lack realism and practical results. Extending the testbed of this approach to a more powerful and job-specific computing node may allow an increase in the number of nodes and also a reduced episode time, resulting in the ability to complete a larger number of training episodes. Building on this, there is also an inherent difficulty to evaluating IRS proposals from a quantitative view due to the lack of a standardised testbed. In research on Intrusion Detection Systems standardised datasets are used such as the Coburg Intrusion Detection Datasets (CIDDS) [28]. The lack of standardised testbeds results in an inability to directly compare IRS performance without full recreation of another paper's testbed and attack scenarios. Hence when we look at IRS proposals, comparisons against related work are often not defined in terms of performance. Interestingly, Montemaggio et al. have recently proposed a framework which enables quantitative evaluation of Self-Protecting systems [29].

We acknowledge a limitation to the number of training episodes that ran, which is primarily to the long boot times of the full virtual machines used in our testbed. This will be a prominent challenge for emulated environments as research continues.Generalisation to changes in the attack or the defended network is a significant potential attribute of Model-free IRS. However, a large number of training episodes is needed to realise this potential. We hope that with future work we can build on this approach with the use of faster technology such as containers or parallelisation, both of which would significantly improve the environment reset time A particular area of comparison would be to [11] which investigates the ability to adapt the optimal response after the addition of a new action. It may be interesting to attempt to train a more broadly experienced IRS which can adapt to a range of attack scenarios. A distributed IRS approach could be adopted, in which a range of RL agents have been trained to handle an attack of a particular genre, for example, an agent could be trained for each tactic from the MITRE ATT&CK framework, or even each technique within [30]. The presented framework would provide an effective starting point due to the ability to change the type of attack or add and remove response actions with ease.

CRediT authorship contribution statement

Kieran Hughes: Methodology, Software, Investigation, Validation, Formal analysis, Writing – original draft. **Kieran McLaughlin:** Conceptualization, Writing – review & editing, Visualization, Supervision. **Sakir Sezer:** Resources, Project administration, Funding acquisition.

Declaration of Competing Interest

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

References

[1] Roesch M. Snort - lightweight intrusion detection for networks. In: Proceedings of the LISA '99 13th systems administration conference; 1999.
[2] Somayaji A, Forrest S. Automated response using system-call delays. In: Proceedings of the 9th USENIX security symposium; 2000. p. 14.
[3] Roy A, Kim DS, Trivedi KS. Cyber security analysis using attack countermeasure trees. In: Proceedings of the ACM's international conference series; 2010. https://doi.org/10.1145/1852666.1852698.
[4] Shameli-Sendi A, Dagenais M. ORCEF: online response cost evaluation framework for intrusion response system. J Netw Comput Appl 2015;55:89–107. https://doi.org/10.1016/j.jnca.2015.05.004.
[5] Foo B, Wu YS, Mao YC, Bagchi S, Spafford EA. Adaptive intrusion response using attack graphs in an e-commerce environment. In: Proceedings of the international conference on dependable systems and networks; 2005. p. 508–17.
[6] Chung CJ, Khatkar P, Xing T, Lee J, Huang D. NICE: Network intrusion detection and countermeasure selection in virtual network systems. IEEE Trans Dependable Secur Comput 2013;10:198–211. https://doi.org/10.1109/TDSC.2013.8.
[7] Shameli-Sendi A, Louafi H, He W, Cheriet M. Dynamic optimal countermeasure selection for intrusion response system. IEEE Trans Dependable Secur Comput 2018;15:755–70. https://doi.org/10.1109/TDSC.2016.2615622.
[8] Li G, Zhou H, Feng B, Li G, Yu S. Automatic selection of security service function chaining using reinforcement learning. In: Proceedings of the IEEE globecom workshops (GC Wkshps); 2018. https://doi.org/10.1109/GLOCOMW.2018.8644122. 2018 - Proc 2019.
[9] Zonouz SA, Khurana H, Sanders WH, Yardley TM. RRE: a game-theoretic intrusion response and recovery engine. IEEE Trans Parallel Distrib Syst 2014;25:395–406. https://doi.org/10.1109/TPDS.2013.211.
[10] Iannucci S, Abdelwahed S, Montemaggio A, Hannis M, Leonard L, King JS, et al. A model-integrated approach to designing self-protecting systems. IEEE Trans Softw Eng 2020;46:1380–92. https://doi.org/10.1109/TSE.2018.2880218.
[11] Iannucci S, Cardellini V, Barba OD, Banicescu I. A hybrid Model-free approach for the near-optimal intrusion response control of non-stationary systems. Futur Gener Comput Syst 2020;109:111–24. https://doi.org/10.1016/j.future.2020.03.018.
[12] Zolotukhin M, Kumar S, Hamalainen T. Reinforcement learning for attack mitigation in SDN-enabled networks. In: Proceedings of the 6th IEEE conference on network softwarization (NetSoft); 2020. https://doi.org/10.1109/netsoft48620.2020.9165383.
[13] Miehling E, Rasouli M, Teneketzis D. Optimal defense policies for partially observable spreading processes on Bayesian attack graphs. In: Proceedings of the 2nd ACM workshop on moving target defense; 2015. p. 67–76. https://doi.org/10.1145/280475.28084828. Co-Located with CCS 2015.
[14] Gonzalez-Granadillo G, Garcia-Alfaro J, Alvarez E, El-Barbori M, Debar H. Selecting optimal countermeasures for attacks against critical systems using the attack volume model and the RORI index. Comput Electr Eng 2015;47:13–34. https://doi.org/10.1016/j.compeleceng.2015.07.023.
[15] Sutton R, Barto A. Reinforcement learning: an introduction. Robotica 1999;17:229–35. https://doi.org/10.1017/s0263574799211174.
[16] Bellman R. A Markovian decision process. Indiana Univ Math J 1957;6:679–84. https://doi.org/10.1512/iumj.1957.6.56038.
[17] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal policy optimization algorithms 2017. Accessible at: arxiv.org/abs/1707.06347.
[18] Schulman J, Levine S, Moritz P, Jordan M, Abbeel P. Trust region policy optimization. In: Proceedings of the 32nd international conference on machine learning. 3; 2015 2015. p. 1889–97.
[19] Li X, Zhou C, Tian YC, Qin Y. A dynamic decision-making approach for intrusion response in industrial control systems. IEEE Trans Ind Inform 2019;15:2544–54. https://doi.org/10.1109/TII.2018.2866445.
[20] Rapid7. Metasploitable 2010 [online]. Accessible at: https://information.rapid7.com/download-metasploitable-2017.html Accessed February 10, 2021.
[21] Iannucci S, Abdelwahed S. A probabilistic approach to autonomic security management. In: Proceedings of the IEEE international conference on autonomic computing (ICAC); 2016. p. 157–66. https://doi.org/10.1109/ICAC.2016.12.
[22] Mateos V, Villagrá VA, Romero F, Berrocal J. Definition of response metrics for an ontology-based automated intrusion response systems. Comput Electr Eng 2012. https://doi.org/10.1016/j.compeleceng.2012.06.001.
[23] Sharma RK, Issac B, Kalita HK. Intrusion detection and response system inspired by the defense mechanism of plants. IEEE Access 2019;7:52427–39. https://doi.org/10.1109/access.2019.2912114.
[24] Li W, Tian S. An ontology-based intrusion alerts correlation system. Expert Syst Appl 2010;37:7138–46. https://doi.org/10.1016/j.eswa.2010.03.068.
[25] Ji X, Yu H, Fan G, Fu W. Attack-defense trees based cyber security analysis for CPSs. In: Proceedings of the 17th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD); 2016. p. 693–8. https://doi.org/10.1109/SNPD.2016.7515980.
[26] Lin H, Slagell A, Kalbarczyk ZT, Sauer PW, Iyer RK. Runtime semantic security analysis to detect and mitigate control-related attacks in power grids. IEEE Trans Smart Grid 2018;9:163–78. https://doi.org/10.1109/TSG.2016.2547742.
[27] Iannucci S, Chen Q. Abdelwahed S. High-performance intrusion response planning on many-core architectures. In: Proceedings of the 25th international conference on computer communication and networks (ICCCN); 2016. p. 1–6. https://doi.org/10.1109/ICCCN.2016.7568529.
[28] Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A. Flow-based benchmark data sets for intrusion detection. In: Proceedings of the European conference on cyber warfare and security (ECCWS); 2017. p. 361–9.
[29] Montemaggio A, Iannucci S, Bhowmik T, Hamilton J. Designing a methodological framework for the empirical evaluation of self-protecting systems. In: Proceedings of the IEEE international conference on autonomic computing and self-organizing systems companion (ACSOS-C); 2020. p. 218–23. https://doi.org/10.1109/ACSOS-C51401.2020.00059.
[30] Strom B.E., Miller D.P., Nickels K.C., Pennington A.G., Thomas C.B. 2018. MITRE ATT&CK: design and philosophy. Accessible at: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf.