Smart Security Audit: Reinforcement Learning with a Deep Neural Network Approximator

Konstantin Pozdniakov, Eduardo Alonso, Vladimir Stankovic
[konstantin.pozdniakov.1, E.Alonso, Vladimir.Stankovic.1]@city.ac.uk
City, University of London

Kimberly Tam, Kevin Jones
[kimberly.tam, kevin.jones]@plymouth.ac.uk
University of Plymouth

***Abstract*—A significant challenge in modern computer security is the growing skill gap as intruder capabilities increase, making it necessary to begin automating elements of penetration testing so analysts can contend with the growing number of cyber threats. In this paper, we attempt to assist human analysts by automating a single host penetration attack. To do so, a smart agent performs different attack sequences to find vulnerabilities in a target system. As it does so, it accumulates knowledge, learns new attack sequences and improves its own internal penetration testing logic. As a result, this agent (`AgentPen` for simplicity) is able to successfully penetrate hosts it has never interacted with before. A computer security administrator using this tool would receive a comprehensive, automated sequence of actions leading to a security breach, highlighting potential vulnerabilities, and reducing the amount of menial tasks a typical penetration tester would need to execute. To achieve autonomy, we apply an unsupervised machine learning algorithm, Q-learning, with an approximator that incorporates a deep neural network architecture. The security audit itself is modelled as a Markov Decision Process in order to test a number of decision-making strategies and compare their convergence to optimality. A series of experimental results is presented to show how this approach can be effectively used to automate penetration testing using a scalable, i.e. not exhaustive, and adaptive approach.**

***Index Terms*—Pentesting, audit, Q-learning, reinforcement learning, deep neural network**

I. Introduction

Computer security auditing is a decision-making process where a computer system is comprehensively explored and tested, in a secure environment, to reveal potential vulnerabilities. Strategically exploring a target system, as an attacker would, to discover and fix vulnerabilities is currently done by cyber-security specialists. While often resulting in high accuracy, human-based decision making is a limited resource in today's world, due to skill shortages and rising threats [1]. Therefore, developing an intelligent automated audit process to complement human resources would be an useful aid going forwards. However, most existing penetration testing tools (i.e. pentesting) rely on predefined datasets of exploits and trigger vulnerabilities one-by-one. The drawback of this is that, in practice, strategic human decisions are needed at each exploit "link" in an attack chain, consuming considerable resources. Conversely, existing automated attack tools struggle to adapt to environmental changes or update an attack strategy mid-test. Related works are discussed further in Section I-A.

This paper presents a novel, adaptive approach to pentesting, using an automated, learning-based, decision-making process. The tool, `AgentPen`, removes the human element from the actual penetration attack stage and discovers its own strategy, adapting and improving its attack logic in real-time. This mitigates human limitations, such as speed and scalability. That said, this approach does not completely remove the need for cyber-security expertise, but instead raises human analysts to a higher level of control. Human-based decisions will still drive (1) the audit with high-level learning algorithm modifications, (2) hacking ethics, legal concerns, and results, and (3) develop machine learning strategies and solutions to increase the performance. `AgentPen` can be tuned by analysts, given specific tasks and goals, relieving cyber-security experts from the more menial, routine, pentesting tasks and enabling them to focus on higher-level control tasks.

To achieve a sophisticated level of automation, `AgentPen` applies a model-free reinforcement machine learning algorithm. Unlike most other methods, this pentesting does not use a full predefined model of the environment and, instead, learns the environment using Q-learning [2]. To better generalize the solution, the original Q-learning algorithm was modified by integrating an approximator to improve its performance while maintaining accuracy. This approach has shown that it can mitigate all the disadvantages of pre-defined environmental models and also address our research problems (Section I-B).

A. Related Work and Background

Most computer security audit software tools are limited by human resources, target environment changes, or both, making many state-of-the-art audit processes time consuming and/or error-prone. This is particularly troubling with growing skill-shortages in cyber-security [3], threat increases [4], and increasing number of computing systems [1]. Auditing relies on the use of software exploits, pieces of computer code that trigger a system vulnerability, and additional actions (i.e., payloads) may follow after a successful penetration. Nessus [5] executes Network Vulnerability Tests one by one, using a list of targets, without examining real-time environmental changes. Nessus does not automate any decision-making and is not able to operate independently. Core Impact [6] is allegedly the most advanced vulnerability scanner on the market and builds automated attack plans to be executed by an auditor. The limits of this strategy is that the attack plan is built on a target environment model prior to penetration testing, making the tool highly dependent on the model's

quality and unable to adapt mid-attack. For example, while building its model and generating an attack plan, Core Impact assumes that the network configuration is static and in order to deal with network/host configuration changes, it would need to recreate the environment model and recalculate the whole attack plan, increasing cost. This tool has a few other drawbacks, internal model parameters compromise scalability and it does not have enough data for new OS versions or network configurations, reducing likelihood an optimal attack plan is used. Another popular penetration testing tool that lacks intelligent automation is Rapid 7 Nexpose [7]. This scanner supports the entire vulnerability management life-cycle by using Metasploit [8], a vulnerability exploitation penetration tool. While this can distribute multiple scan engines across the network for flexibility, Nexpose cannot learn attack strategies itself. In summary, the majority of well-known commercial vulnerability scanners have shown many benefits, but are highly dependent on the expertise and decisions of security specialists. These tools have also exhibited adaptability issues, unable to update an attack strategy if the environment changes.

Within academic literature, security audit scope studies have proposed improvements to the accuracy, speed and state-space reduction of penetration testing processes [9]–[12]. However, these rely heavily on the initial modelling stage and demand significant security expert involvement. Machine learning techniques have mainly been used in Intrusion Detection Systems (IDSs) to improve detection, classify threats, and extract features [13]–[16] rather than automating audits *per se*. Other approaches like model checking [17], [18] identify vulnerabilities by comparing several models over a large space. However, this large space approach is inherently exhaustive, difficult to scale [19], [20], and not adaptive to target environment changes. Another non-security paper [21] used model-free machine learning for agile, self-adaptive congestion management in networks.

This paper uses machine learning techniques such as reinforcement learning (RL), Q-learning in particular, and neural networks. In RL, an agent learns by interacting with the environment, executing actions at a given state, observing the immediate outcome, i.e. reward signal, and making subsequent predictions about the future value of their choices (i.e. Q-value of a state-action pair) [22]. Q-learning [2] has been proven to converge to an optimal solution in tabular cases [23] and adjustments based on experience are executed independently of the policy followed. In tabular cases, Q-values are stored in a table for every Markov Decision Process (MDP) state-action pair. Problems can arise when a large state space makes the corresponding table intractable. Past studies have overcome this with an approximator, [24]–[26]. To account for state-action space limitations in tabular cases, our approach uses a neural network-based approximator to give our tool both adaptive and scalable capabilities, which is novel compared to the state-of-the-art approaches.

Neural networks are universal approximators implemented as layers of connected neurons that transmit signals to each other and are modelled by their architecture [27]. To approximate the state-action space in sequential tasks (e.g., a series of exploits in an attack sequence) `AgentPen` uses a Recurrent Neural Network (RNN), specifically, the Elman Neural Network. RNNs typically have three layers (input, hidden and output) which are connected in a feed-forward manner with additional connections between hidden and input layers to feed hidden neuron values back to the input. This structure means previously experienced values to future inputs can be incorporated, so `AgentPen` can learn a sequence of actions, "memorizing" previously experienced states [28]–[30]. For low overheads, i.e. high scalability, we choose to use the Elman type RNN [31], [32].This has a special layout with additional neurons feeding to every neuron in the hidden layer of its previous value, implementing an analog of memory.

B. Problem Statement

To effectively pentest systems in the current threat landscape [4], more automation needs to be introduced. However, accuracy and performance are the two challenges that must be addressed. To mirror realistic threats for quality, we consider both *local*, on the target, and *remote*, within the same network, exploits targeting Windows and Linux systems. Part of both quality and quantity problems is selecting which machine learning methods would best automate `AgentPen`. As the target system is assumed unknown in our approach, we consider the learning algorithm to be model-free. This deviates from previous approaches with pre-defined models while still addressing the problem of attack-sequence quality.

While building an approximator to address the challenge of quantity, i.e. scalable performance overhead, it was crucial to choose the best features to describe the state-action space for the problem at hand. More specifically, performance problems with the proposed approach to automate pentesting audits are (1) how to decrease number of features representing the state space and (2) how to approximate state-action space with a small feature set. To achieve this, we use an Elman-type recurrent neural network (RNN) - see Section II-E. Too many features would place a significant load on the learning algorithm and may add unacceptable time overheads during the attack stages. To address this problem, we compress the feature set into a smaller space using an autoencoder, just prior to learning. This approach compresses the neural network input to a smaller, hidden layer with minimal error. In our experiments this reduced feature set helped address the performance challenge without negatively affecting the quality of pentesting achieved.

II. Methodology

In most current pentesting approaches, a human auditor is needed to launch exploits and perform different attacks, using a set of tools. In contrast, this paper proposes transferring the menial pentesting tasks to the intelligent automated `AgentPen`. This section is dedicated to describing our methodology to the noted challenges of achieving this.

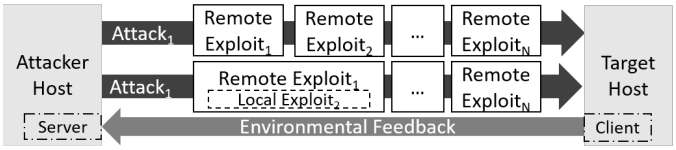

Fig. 1. Automatic learning pentesting audit process

A. Architecture of the Audit System

`AgentPen` consists of a client (target) and server (host) (see Figure 1) on the same corporate network. The server performs multi-step attacks without human guidance by generating and executing a sequence of actions one-by-one in a chain. Actions include exploits and discoveries, such as port scan. After the server attacks a target, the client is notified of the attack. It then checks whether the exploit was successful and sends the results back to the server. It also restarts local services if needed.

The target host in this paper is Windows 7 with a number of potentially vulnerable ports and services (see Tables I-II). It is important to note that `AgentPen` can use any set of exploits and work with OSs besides Windows. For the purpose of producing useful experimental results in Section III, Win7 was used because it is outdated and vulnerable, testing `AgentPen`'s ability to learn an optimal attack strategy, not just a valid one. Although targets and environments may change (e.g., target updates to Win10), the techniques presented are able to adapt by using a set of viable exploits during its learning penetration process. Exploits used by `AgentPen` are customized for the environment, derived from Metasploit and other external sources. For Win7, remote code execution and local privileges buffer overflow exploits were used. Exploit payloads were also used to trigger uniquely-named processes within targets, and once `AgentPen` successfully penetrates one target, it automatically moves on to pentest its next target in a network consisting of more than 80 hosts.

All decisions on exploits sequences, and the generation of custom strings for attacks, are done using server-side Q-learning, while the client generates reward signals. The environmental feedback provided by the client is fed back into the server Q-learning algorithm, indicating if the exploit successfully breached the target. This updates the reward signal (see Section II-C) according to the server's actions. This client-server architecture is designed to prevent malicious uses of `AgentPen` as they should have admin rights on the target to make the server-client system work. Therefore only an authorized owner of the target should be able to run autonomous penetration audits.

B. `AgentPen` Audit Process

Our audit process is modelled as an MDP, with a transition function to define states resulting from executing specific actions in a given state and a corresponding reward function to quantify executed action rewards. These audit states and actions can be found in Tables I and II. An example of transitions between the 18 states is shown in Figure 2, where `Init` is the initial system state. `AgentPen` moves between states by executing exploits and discovery actions that reveal the target environment (Table I, 1-9) including running services. Discovery actions help reduce the action space and, therefore, performance overhead. If it were discovered that the target's OS is Windows, Linux-related exploits would not be used.

TABLE I
SECURE AUDIT STATE NUMBER, START STATE, AND TRIGGERS.

#	Start State	State trigger and requirements
1	H8.34_Win	The host has a Windows OS
2	H8.34_Lx	The host has a Linux OS
3	H8.34_P445	The host has an open port 44
4	H8.34_P135	The host has an open port 135
5	H8.34_P22	The host has an open port 22
6	H8.34_P2525	The host has an open port 2525
7	H8.34_serv1	The host has a vulnerable $RDP_{Service}$
8	H8.34_serv2	The host has a vulnerable $Msvcrt_{Service}$
9	H8.34_SSH	The host has a SSH service
10	H8.34_Expl$_1$	Remote $Exploit_1$ for $serv_1$ is executed
11	H8.34_Expl$_2$	Remote $Exploit_2$ is executed
12	H8.34_Expl$_3$	Local $Exploit_3$ is executed
13	H8.34_SSH	SSH brute force attack is performed
14	H8.34_Expl$_1$_2	Remote $Exploit_1$ executed as 2^{nd} exploit
15	H8.34_Expl$_2$_2	Remote $Exploit_2$ executed as 2^{nd} exploit
16	H8.34_Expl$_3$_2	Local $Exploit_3$ executed as 2^{nd} exploit
17	Fail_terminal	Final state, dead end, i.e. exploit failed
18	Suc_terminal	Final state, target host hacked

TABLE II
CHECKS PRIOR MDP ACTIONS TO CHANGE STATE.

State Check	Description
OS_Win_Check	Checks for Windows OS using `nmap`
OS_Linux_Check	Checks for Linux OS using `nmap`
Port445_Check	Checks if Port445 is open using `nmap`
Port135_Check	Checks if Port135 is open using `nmap`
Port22_Check	Checks if Port22 is open using `nmap`
Port2525_Check	Checks if Port2525 is open using `nmap`
Service$_1$_Check	Check Service$_1$ process is running
Service$_2$_Check	Check Service$_2$ process is running
SSH_Check	Check Port22 and parse connection requests
Exploit$_1$	Grants local attacker admin privileges
Exploit$_2$	Grants remote user-level privileges
Exploit$_3$_local	Exploit local services, privilege escalation
SSH	SSH Brute force

In our experiments, targets were left intentionally vulnerable for remotely executed $Exploit_1$, which grants admin privileges to the attacker, and $Exploit_2$, which grants user-level privileges. Any privilege changes or successful remote/local code executions is monitored by the feedback agent. Vulnerabilities in local services, $RDP_{Service}$ and $Msvcrt_{Service}$, are exploited locally using $Exploit_3$ to escalate attacker privileges to admin. RDP is responsible for remote connections and Msvrcrt supports executions of basic C functions from Microsoft C Runtime Library. If exploits required unique strings, `AgentPen` automatically generates these with RL.

C. Reward Function

A reward function guides learning by allocating a numerical signal to each transition and, as previously mentioned, it is produced by the client. Intuitively, the "better" an action is at leading to a successful attack, a higher associated reward is returned. This is critical for the automated adaptive attack

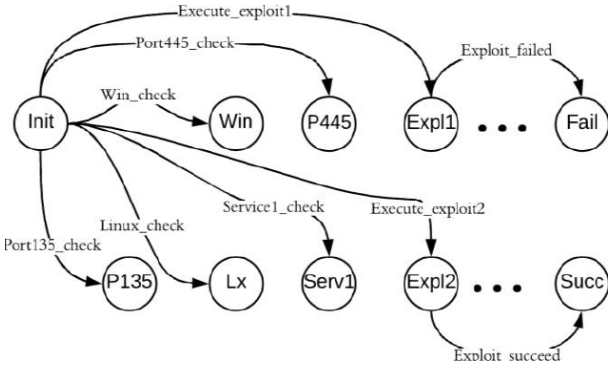

Fig. 2. MDP audit system transitions from initial state.

Algorithm 1: Q-Learning Algorithm

```
1  Initialize Q(state, action);
2  Observe current state;
3  for every time step do
4      Select action from state;
5      if state trigger successful then
6          take action;
7          check next state' and reward;
8          update Q-values (see Eq 2);
9      else
10         end action;
11     end
12     state ← states'
13 end
```

strategy, independent from external human help or environment models. While the reinforced learning algorithm and approximator elements will be defined in the next subsection, here we define the reward function. This is the key for finding an optimal attack sequence as the agent maximizes its cumulative reward via trial and error. The reward function used (see Eq 1) is defined as a linear combination of the main reward and additional rewards, defined in the bullets below.

$$Reward = R_{main} + R_1 + R_2 + R_3 \quad (1)$$

Here R_{main} is defined as a value for reaching a successful terminal state. The additional reward components are defined as follows:

- R_1: exploit effectiveness according to the metasploit rating [33]. The more effective, the higher the reward;
- R_2: possible OS/service harm if exploit used, according to the Common Vulnerabilities and Exposures database [34]. If system service crashes, the reward is smaller;
- R_3: additional testing actions (e.g. port scan), defined as $20/n$ where n is the number of iterations per episode and 20 was obtained experimentally. R_3 motivates an agent to apply environment checks before actual attacks.

High reward values signify better attack choices in order to reach the successful terminal state (18 in Table I). To optimize the learning process negative rewards are also applied if the agent enters an action-loop, if it had learned a non-optimal experience, helping it break out and reach either terminal state. Human ingenuity is still required to get these coefficients right and this negative reward is elided from Eq1. In our approximated Q-learning experiments, the reward values are mapped to the range [0,1] as those are the min and max values for neural network neurons. These reward values are designed to provide a fast, technically valid, penetration attack strategy.

D. Q-Learning

This model-free approach is autonomous and does not require previous knowledge on the target environment; it uses reinforcement learning instead, Q-learning in particular, to identify new system vulnerabilities through the execution of attack sequences. At each time step in the sequence, `AgentPen` chooses an action randomly according to a softmax distribution (i.e., following a softmax policy), which incorporates the knowledge about previously experienced states. As a result, the actions leading to higher reward states will be chosen more often. At the end of an attack sequence the audit agent receives a numerical signal from either the client or target agent indicating if it was successful in penetrating the host. With this information the agent updates its estimations (i.e., Q-values for each state-action pair), about the long-term value of executing the attack, for which the audit agent receives the immediate reward, and what `AgentPen` believes it would receive in subsequent attacks. This Q-learning algorithm can be succinctly described with the pseudo-code in Algorithm 1 where update "Q-values" is achieved with Eq 2.

$$Q(s,a) \leftarrow Q(s,a) + \alpha([r + \gamma max Q(s',a')] - Q(s,a)) \quad (2)$$

Here s and s' define current and next states respectively. a and a' stand for current and future actions. r represents a $Reward$ defined in Eq. 1. Since `AgentPen` does not know the underlying MDP model, its initial Q-value estimations are initialized to zero, as it is ignorant and inexperienced. This only occurs at the start of an audit, Step 1 of learning Algorithm 1. In Step 2 the agent observers and checks which audit state it is in and which actions it is allowed to execute. It then executes one of the actions following the softmax policy. After executing an action successfully two things happen: the agent moves to another state and receives a reward, as shown by Step 7's if-statement. By interacting with the target environment, the agent updates its Q-value estimations, at Step 8, using Eq. 2. This defines the learning rule, which is used to find the new Q-value based on the old value plus the estimation (prediction) error multiplied by a learning rate α, a value between 0 and 1. This dictates how quickly the error is updated, modulating the learning process. The error itself is the difference between the old estimation and the result of adding the immediate reward the agent receives after executing the action and what it estimates will be the best option in the future, discounted by a γ parameter, which also ranges from 0 to 1, and represents how confident the agent is about its future predictions. The agent follows this process until a terminal state is reached and this entire process, starting from `Init`, is considered as one learning episode or epoch, and this is repeated until the agent converges to an

optimal penetration strategy, i.e. an attack sequence which best maximizes cumulative rewards.

E. Deep Architecture Approximator

Although AgentPen may eventually successfully penetrate the target, to improve performance, an autoencoder reduces the state space. Traditional implementations of Q-learning use a table, or matrix, where the utility function stores its values. As the state space increases, the standard tabular approach does not scale well and often negatively affects overall performance. An approximator can mitigate this problem by learning the Q-function itself, instead of Q-values one at a time. To improve audit performances this paper uses an Elman-type RNN.

Like any other neural network, an Elman-type RNN consists of interconnected nodes that process information across its architecture, from input to output nodes. Characteristically, RNN also has hidden nodes which can have multiple input and output connections. Elman networks, a three-layer version of RNN, includes a critical "context unit" that feeds the values from previous processing steps back to the hidden layer neurons, merging them with current input values, providing a type of "memory". This gives AgentPen the ability to remember a sequence of pentesting actions. The number of input, middle, and output nodes per layer in an Elman-RNN is not defined and can be varied. Backpropagation through time (BPTT) algorithm is used to train the neural network [35]. This algorithm is an application of traditional backpropagation (BP) [36] to the RNN sequence data. BP propagates the inputs of neural network to calculate the output. Next, the actual output is compared to the predicted one and the derivatives of the error with respect to the network weights are calculated. On the later step, all weight are adjusted to minimize the output error. This process continues until the error is minimised enough.

The main principle of the autoencoder's operation is the adjustment of internal weights within the neural network to recreate the input as an output with minimal error. This can be used to reduce the input layer to the smaller hidden layer with minimal loss. As a result, the autoencoder learns to extract the correlation of input data automatically and maps it to a smaller space, increasing performance without sacrificing its adaptive pentesting capabilities. In the experiments evaluating AgentPen's performance, in Section III, features describing the target host are fed as inputs to the autoencoder. As a result, the autoencoder maps the features to a smaller hidden layer, which are then fed as inputs to the approximator-based Elman-type RNN. These interactions form the deep learning architecture for AgentPen's Q-learning approximation.

In summary, the very nature of reinforcement learning for our research problem is to learn the optimal sequence of penetration attacks by prioritizing the autonomous AgentPen's actions to gain maximum cumulative rewards. The approach proposed in this paper does so by exploring and exploiting the target system(s) intelligently to discover vulnerabilities through interactions with its environment, as opposed to having all actions dictated by a pre-generated model. In addition, unlike supervised learning, AgentPen is not trained with successful attacks but instead learns from scratch. This is particularly valuable when the audit tool is capable of discovering attack sequences without analyst involvement. Therefore, this reinforcement learning approach offers a critical mechanism for AgentPen to learn the optimal sequence of system penetration actions, given metrics of its behaviour, even when starting from complete ignorance. Alternatively, AgentPen can be pre-trained before deployment into the environment if an analyst chooses to. In this case it will be able to identify the vulnerabilities much faster based on the similarity of vulnerable configurations it already experienced.

III. Experiments and Results

To test AgentPen several experiments were designed to illustrate how Q-learning could be successfully applied to the pentesting process and discover its own attack strategy despite unforeseen environmental changes. Without a pre-defined environment model AgentPen can only execute actions and learn from trial/errors it experienced in real-time.

Experiments are divided into two groups, tabular Q-learning (Figure 3 (a)) and Elman-RNN approximated Q-learning (Figure 3 (b) and (c)). The first group was intended to test the basic hypothesis of the model-free approach, as outlined in Section II and whether it could autonomously learn a technically correct penetration attack sequence. In these experiments the Q-learning algorithm (see Algorithm 1) is used without human assistance or prior knowledge of predefined transitions and rewards for every state-action pair. The reward was defined by general reward function based on successful terminal state reach and exploratory actions only. The relaxation of predefined transitions and rewards requirements leads to learning them directly from interaction with the environment. Since these first experiments were designed to only test the basic concepts, the penetration testing process was defined as a specific MDP with state and action spaces.

Once it is established that the tabular setup (first group of experiments) in Figure 3 (a) could optimally learn a successful attack strategy, the other two setups (second group of experiments) were designed to test the performance enhancing approximator and AgentPen's real-time adaptive capabilities. First group shows the basic model-free approach. The second group demonstrates the extension of model-free approach to the real world scenarios when using a state space reducing autoencoder and aproximator to process large state spaces. These second group experiments do not compare the learning speed with the tabular approach, but test the concept with a larger real-world state space using processes previously discussed. The second group setups seen in Figure 3 (b) and (c) use our autoencoder as explained in Section II. With this, AgentPen not only learns an attack strategy in real time, but extends that accumulated knowledge to cases it has never experienced before, like a new target.

The experiments in Group 1 have no state space reduction, so we used the optimal hyperparameters (e.g., learning rate, discount factor) of the Q-learning algorithm identified with our tabular Q-learning experiments. In experiments related to

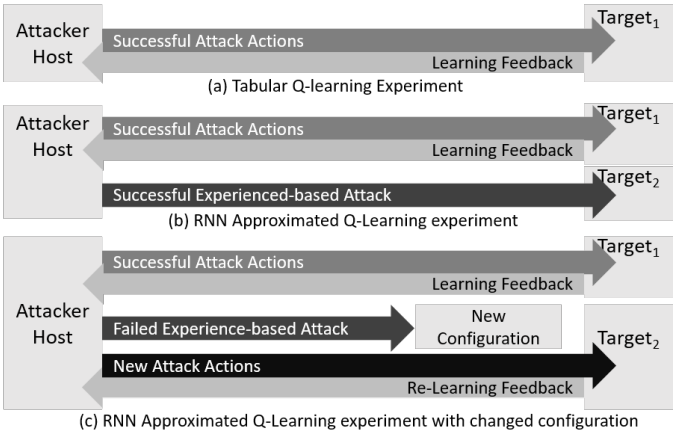

Fig. 3. Overview of automatic audit experiments.

Group 2, approximator architecture related hyperparameters were tested in order to check pentesting learning performance: the autoencoder was tested with different values of weight decay, number of iterations, neurons per hidden layer, and sparsity penalties. RNN hyperparameters, such as weight decay, number of iterations and neurons quantity per hidden layer, were tested as well.

In first setup for Group 2, the successful model-free attacks were demonstrated in non changing environment. In second setup for Group 2, the environment was changed during the audit process. With these changes, the attack strategy that was learned earlier would no longer lead to successful penetration. This change is introduced to test if our approach is truly able to adapt attacks in real-time. In our experiments, it is shown that `AgentPen` can do this, independently finding a new successful penetration attack with its own logic in several real-world environments.

A. Group 1: Tabular Q-Learning

The first set of experiments to penetrate and escalate privilege in the target consists of learning the optimal audit strategy using the tabular case Q-learning algorithm (see Figure 3 (a)). As described in Section II-A, the server executes the majority of the pentest (i.e. exploit and discovery actions) unless the client is instructed to perform $Exploit_3$ locally after remote $Exploit_2$ is executed.

The Group 1 tabular Q-learning experimental results show that the algorithm was able to learn the optimal sequence leading to the successful exploitation of the vulnerability after 1,000 episodes: `Initialization + OS_Win_Check +` $Exploit_2$ + $Exploit_3$. This solution was the optimal one for this experiment, as the execution of remote $Exploit_2$ gives `AgentPen` user privileges, which are then escalated to admin-level with local $Exploit_3$. This was more effective than executing the single remote $Exploit_1$ since $Exploit_2$ has a higher penetration rating that the former, which affected the reward value. Furthermore, the remote $Exploit_1$ affects the system process and is potentially more harmful to the target, which is not the aim of a pure penetration attack. Looking at

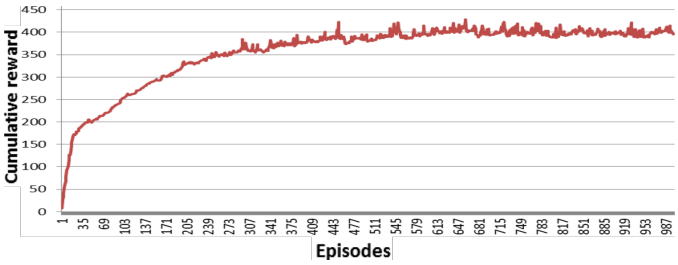

Fig. 4. Tabular Q-learning experiment performance

these results, it is apparent that Q-learning achieves the best possible strategy by maximizing the reward. Scalability and performance is addressed in group two of our experiments.

Figure 4 shows that the tabular Q-learning algorithm performance increased until roughly 545 epochs. This is because the acting policy initially chooses actions randomly, ignorantly, but as `AgentPen` learns, it is able to use past experiences to optimize exploring target system vulnerabilities. As it takes time to randomly explore most of the states, performance increases slowly. A plateau appears towards later episodes as `AgentPen` begins to act intelligently, choosing the most optimal action sequences it has learned. As a result, the best sequence of actions is always chosen, and the cumulative reward value stabilizes. In Group 1 experiments the most optimal hyper parameters for the Q-learning algorithm, using the softmax acting policy, were identified as $\gamma = \alpha = 0.8$.

These results show that tabular QL performs relatively well. Nevertheless, the algorithm performance is hindered by the random exploring actions at the start. Group 2 experiments reduce the state space to optimize `AgentPen`. While it is possible that "fewer experienced states" could hinder the learning process, that has not been observed in our experiments.

B. Group 2: RNN Approximated Q-Learning

Group 2 experiments were designed to build on the success of Group 1 model-free learning approach and address potential performance issues with the hypothesis that Q-learning, paired with an autoencoder based on an artificial neural network, could reduce the state space and, therefore, lessen the amount of time needed to find an optimal attack strategy. Unlike Group 1, Group 2 presents the attacker host with multiple targets. $Target_1$ is the first target to be penetrated. During this process `AgentPen` learns an attack strategy based on the learning feedback. Once a wealth of experience is gained, `AgentPen` is self-directed to penetrate $Target_2$ using only the knowledge it had accumulated.

In these experiments `AgentPen` derives the reward value directly from the environmental feedback while acting without a predefined transition function. The representation of the penetration testing process is also generalized for huge state spaces. This means that, instead of a fixed number of states, their general representation is defined by the feature set, which is related to the configuration definition of target system. This "configuration" is the combination of processes running and `dlls` used by the target. The learning performance of feature

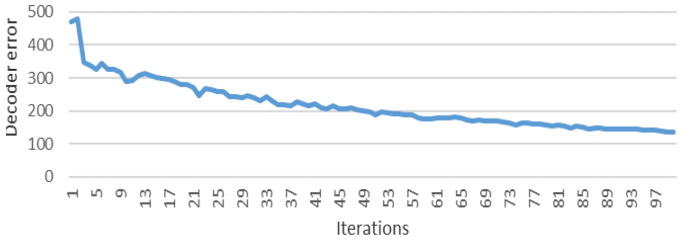

Fig. 5. Autoencoder learning process after 100 episodes/iterations

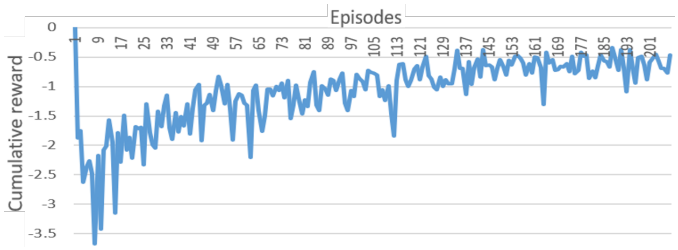

Fig. 6. RNN approximated Q-Learning performance

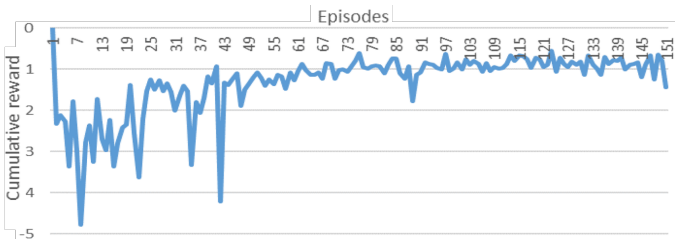

Fig. 7. Adapting RNN approximated QL performance using softmax policy

extraction, to support the state space reducing autoencoder architecture, is presented in Figure 5. It should be noted that the decoder error for the autoencoder learning process seen here was minimized relatively quickly to 126 after 100 episodes, demonstrating a successful mapping of features to a smaller dataset with minimal error. Further minimization of the error did not improve the following RNN learning processes, but significantly slowed down the system performance overall. Therefore, it was decided to use 100 learning epochs as the most optimal value for `AgentPen`. After the feature set was decreased, features were fed to the RNN approximator to learn a sequence of pentesting actions, as seen in Figure 6.

As the autoencoder learning process is a one-time performance cost per target (features are extracted once per host), it can help reduce overall overheads as `AgentPen` learns to penetrate multiple targets, even with some configuration changes. As a result of Group 2 first setup Figure 3 (b), the cumulative reward converged to a solution. This showed an optimal attack strategy, found during the learning process, and consisted of the following actions: `Initialization + Probe action (check OS) +` $Exploit_1$. This generates a new, bespoke, attack sequence when targeting a host it has not experienced before. After the new penetration sequence is learned, `AgentPen` self-directs to attack another target on the network with a client installed. `AgentPen` then acts according to its own experiences, gathered during the learning stage, after it extracts the feature set representing the new target host configuration. This feature set is, again, used as an input to the approximator in order to get the sequence of actions leading to successful penetration without starting the learning process all over again.

In Group 2 second setup Figure 3 (c), a second solution was found when $Target_2$ changed its configuration: `Initialize +` $Exploit_1$. While this sequence leads to a successful penetration, it might seem less optimal than the one first discovered in first setup Figure 3 (b) as probe actions to check OS were not used. However, it was still effective in getting admin privileges on the remote host. Given these results, we can see that the `AgentPen` was able to penetrate a new host, with which it had never interacted before. More importantly, it was able to use past experience accumulated during the previous learning iterations to adapt its attack the new $Target_2$ host.

After one successful penetration of a $Target_2$, the target configuration was changed mid-attack to test whether `AgentPen` could truly independently adapt and create another successful penetration strategy (Figure 3 (c)). The old configuration was updated in such a way that the old successful attack strategy, revealed during the second phase, no longer provided a successful target penetration. As a result, it utilized its learning process (Figure 7) in order to find this new successful attack strategy: `Initialize +` $Exploit_2$ `+` $Exploit_3$. Therefore, `AgentPen` did independently adapt its penetration strategy with minimal overheads.

IV. Discussions

The main contribution of the paper is its approach to automating the computer security audit process using Q-learning to create an intelligent audit agent. In most other methods, the attack strategy is developed manually by a computer security expert. Such approaches cannot handle huge state spaces or difficult environmental models used during assessment [37], [38]. In some circumstances, security experts use semi-automatic tools to support manually created attack strategies, however more intelligent tools are needed as technology becomes more prevalent, and the skill gap increases.

Other approaches apply machine learning with environmental models [39]. Unfortunately, by automating the decision-making process this way, the planning approach generates problems. Building models cost time and computations, particularity when using exhaustive approaches. In cases where the model was not developed properly, the attack strategy of the audit agent is not reactive and every change to the host demands new additional simulations in order to rebuild the environment model. Additionally, modern penetration testing systems are not able to accumulate experience in real time.

This paper aims to provide scalable, autonomous, auditing without these drawbacks and the results in Section III demonstrate that it is possible by applying model-free machine learning algorithms and state space reductions. In our approach, people have been fully excluded from the attack

decision-making process, and can instead focus on higher-level strategies, ethics, etc [40]. The method presented achieved this despite the fact that multiple steps were needed for a successful attack and the environment could change unexpectedly. Therefore `AgentPen` is able to derive attack strategies itself without cyber security officer involvement. It also learns the penetration strategy in real-time with accumulated experience. `AgentPen` also mitigates the problem of huge state space that other exhaustive approaches depend on - it uses approximators to efficiently define all possible host configurations. This allows for derivation of successful attacks strategies even for new targets. The developed deep learning architecture works universally for any quantity and quality of the chosen feature set. Thus, `AgentPen` is significantly scalable and adjustable.

There are several future research directions for this work. Different ways of feature representation could be developed for optimization. Additional features can be added to the set, describing a host's network position in relation to the attack path. This could make the agent learn attack strategies based on the target's network environment as well. The proposed deep learning architecture leaves freedom to experiment with different feature extraction and learning mechanisms. For example, instead of RNNs, different types of neural networks can be used. The back-propagation algorithm can also be substituted, or the activation functions, to improve learning performance. The reward function can also be altered for different situations, for example factoring in ethics and laws.

V. Conclusions

A major challenge in modern computer security is automating audit processes so analysts can reliably, and efficiently, analyse the growing number of cyber-threats. While most existing audit tools rely on a predefined attack strategy, require an environment model of the target, and depend heavily on human expertise to use the various tools correctly, our proposed approach has none of these constraints and has been shown to be a successful, adaptive and learning audit tool. Unlike previous tools, it can therefore make autonomous decisions on which attack strategy to implement, even when facing unforeseen changes. As `AgentPen` accumulates knowledge, it is able to learn new optimal attack sequences, explore new host environments, and improve its own internal penetration testing logic. With the positive experimental results, we believe that the model-free Q-based learning approach proposed, enhanced with an approximator, can inform analysts of the events leading to security breaches, highlighting potential vulnerabilities, and reducing the amount of menial tasks a typical penetration tester would need to do for the same result.

References

[1] C. Chen, Z. Zhang, S. Lee, and S. Shieh, "Penetration testing in the iot age," *Computer*, vol. 51, no. 4, pp. 82–85, April 2018.

[2] A. Enache and V. V. Patriciu, "Intrusions detection based on support vector machine optimized with swarm intelligence," Applied Computational Intelligence and Informatics, 2014.

[3] H. Kam, P. Menard, D. Ormond, and P. Katerattanakul, "Ethical hacking: Addressing the critical shortage of cybersecurity talent," PACIS, 2018.

[4] McAfee, "2019 threat predictions report, mcafee," 2019.

[5] Nessus, http://www.tenable.com/, Accessed: 2020.

[6] CoreImpact, http://www.coresecurity.com/, Accessed: 2020.

[7] Nexpose, http://www.rapid7.com/, Accessed: 2020.

[8] Metasploit, http://www.metasploit.com/, Accessed: 2020.

[9] Sankalp Singh, J. Lyons, and D. M. Nicol, "Fast model-based penetration testing," Winter Simulation Conference, Dec 2004.

[10] X. Li, X. Han, and Q. Zheng, "Study on model-based security assessment of information systems," in *Computing and Intelligent Systems*, Y. Wu, Ed. Springer Berlin Heidelberg, 2011.

[11] M. Buchler, J. Oudinet, and A. Pretschner, "Semi-automatic security testing of web applications from a secure model," SERE, 2012.

[12] G. Vezzani, A. Gupta, L. Natale, and P. Abbeel, "Learning latent state representation for speeding up exploration," CoRR, 2019.

[13] A. Ajith, J. Ravi, S. Sugata, and H. SangYong", "Scids: A soft computing intrusion detection system," in *Distributed Computing*. Springer, 2005.

[14] S. Mukkamala, A. H. Sung, A. Abraham, and V. Ramos, "Intrusion Detection Systems Using Adaptive Regression Splines," SAO, 2004.

[15] A. Enache and V.-V. Patriciu, "Intrusions detection based on support vector machine optimized with swarm intelligence," SACI, 2014.

[16] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys Tutorials*, 2019.

[17] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," Security and Privacy, 2000.

[18] A. Lomuscio and W. Penczek, "Ldyis: a framework for model checking security protocols," *Fundam. Inform.*, vol. 85, pp. 359–375, 01 2008.

[19] D. Rohit and R. K. Yvonne, "More scalable ltl model checking via discovering design-space dependencies (d3)," in *Springer*, 2018.

[20] G. Marco, C. Alessandro, M. Cristian, T. Stefano, and R. K. Yvonne, "Model checking at scale: Automated air traffic control design space exploration," in *Computer Aided Verification*. Springer, 2016.

[21] M. Babar, M. H. Roos, and P. H. Nguyen, "Machine learning for agile and self-adaptive congestion management in active distribution networks," EEEIC / I CPS Europe, June 2019.

[22] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[23] C. Watkins and P. Dayan, "Q learning: Technical note, machine learning," 8, 279-292, Boston MA: Kluwer Academic Publishers., 1992.

[24] M. Irodova and R. Sloan, "Reinforcement learning and function approximation," pp. 455–460, 01 2005.

[25] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *International Conference on Machine Learning*, ser. ICML. New York: ACM, 2008.

[26] X. Xu, L. Zuo, and Z. Huang, "Reinforcement learning algorithms with function approximation: Recent advances and applications," *Information Sciences*, vol. 261, pp. 1 – 31, 2014.

[27] H. N. Mhaskar and N. Hahm, "Neural networks for functional approximation and system identification," *Neural Comput.*, 1997.

[28] V. Pascal, L. Hugo, L. Isabelle, B. Yoshua, and M. PierreAntoine, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *JMLR*, 2010.

[29] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "'transforming auto-encoders', lecture notes in computer science," UofT, 2011.

[30] K. G. Lore, A. Akintayo, and S. Sarkar, "Llnet: A deep autoencoder approach to natural low-light image enhancement," *CoRR*, 2015.

[31] L. Li, Z. Deng, and B. Zhang, "A fuzzy elman neural network," 05 2000.

[32] D. Samek, "Elman neural networks in model predictive control," ECMS, 2009.

[33] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni, *Metasploit: The Penetration Tester's Guide*. San Francisco: No Starch Press, 2011.

[34] CVE, "database," available at cve.mitre.org, Accessed: 2020.

[35] J. Guo, "Backpropagation through time," Unpubl. ms., Harbin Institute of Technology, 2013.

[36] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[37] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," CoRR, 2013.

[38] Y. Stefinko, A. Piskozub, and R. Banakh, "Manual and automated penetration testing. benefits and drawbacks," TCSET, 2016.

[39] C. Sarraute, G. Richarte, and J. L. Obes, "An algorithm to find optimal attack paths in nondeterministic scenarios," CoRR, 2013.

[40] A. Rahman and L. Williams, "A bird's eye view of knowledge needs related to penetration testing," in *HotSoS*, 2019.