2020 16th International Conference on Mobility, Sensing and Networking (MSN) | 978-1-7281-9916-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/MSN50589.2020.00086

Autonomous Security Analysis and Penetration Testing

Ankur Chowdhary, Dijiang Huang, Jayasurya Sevalur Mahendran, Daniel Romo, Yuli Deng, Abdulhakim Sabur
Arizona State University
Email: {achaud16, dijiang, jsevalur, dsromo, ydeng19, asabur}@asu.edu

***Abstract*—Security Assessment of large networks is a challenging task. Penetration testing (pentesting) is a method of analyzing the attack surface of a network to find security vulnerabilities. Current network pentesting techniques involve a combination of automated scanning tools and manual exploitation of security issues to identify possible threats in a network. The solution scales poorly on a large network. We propose an autonomous security analysis and penetration testing framework (ASAP) that creates a map of security threats and possible attack paths in the network using attack graphs. Our framework utilizes: (i) state of the art reinforcement learning algorithm based on Deep-Q Network (DQN) to identify optimal policy for performing pentesting testing, and (ii) incorporates domain-specific transition matrix and reward modeling to capture the importance of security vulnerabilities and difficulty inherent in exploiting them. ASAP framework generates autonomous attack plans and validates them against real-world networks. The attack plans are generalizable to complex enterprise network, and the framework scales well on a large network. Our empirical evaluation shows that ASAP identifies non-intuitive attack plans on an enterprise network. The DQN planning algorithm employed scales well on a large network $\sim$ 60-70(s) for generating an attack plan for network with 300 hosts.**

***Index Terms*—Penetration Testing, Cloud Network, Attack Graphs, Reinforcement Learning, Deep-Q Network (DQN), Internet of Things (IoT)**

I. Introduction

Penetration Testing (Pentesting) involves skilled cybersecurity professionals generating plans of attacks to find and exploit vulnerabilities in the networks and applications. The current procedure used in pentesting is semi-automated at best and requires significant human effort. The total cybersecurity spending by the year 2021 will be 1 Trillion USD [1], and the global pentesting market size is projected to grow from USD 1.7 billion in 2020 to USD 4.5 billion by 2025 [2]. The information security industry will experience a shortage of cybersecurity workforce by 3.5M by year 2021 [3]. About 65% of organizations have reported a shortage of cybersecurity staff, and 36% of organizations reported a lack of trained cybersecurity professionals.

The demand for continuous pentesting of network attack surface (growing at a polynomial rate) is difficult to be met by cyber-workforce (growing at a linear rate). To ensure that pentesting solutions provide comprehensive testing of the attack surfaces, the need for developing autonomous pentesting solutions has become very important. Artificial Intelligence (AI) [4] is defined as the study of intelligent agents. The intelligent agents perceive their environment and take necessary actions that maximized the chances of achieving the desired goal for the agent.

Some vulnerabilities are difficult to be discovered by pentesting alone such as Advanced Persistent Threats (APTs) [5]. Also, the target environment can have a robust adaptive defense mechanism where the network environment adapts to the attacker's actions such as Moving Target Defense (MTD) [6]–[8]. We do not consider the security vulnerabilities that are part of APTs in this work. We assume that the target environment does not present a deceptive view of the underlying infrastructure (no MTD mechanism in place).

AI can be applied when a machine mimics cognitive functions similar to the human brain, such as *learning*, *problem solving*, and *reasoning*. We utilize network service reachability and vulnerability information to construct an attack graph [9], [10], and host log information to create a state-action map of possible attack paths. The transition system helps guide the AI-solver for generating attack plans. The attack plans are validated by executing them on the real-world network.

Autonomous pentesting is an emerging line of research that has been considered by several existing research works. Obes *et. al.* [11] utilize attack graphs to generate a partially descriptive domain logic (PDDL) representation of the attack model, and generate an attack plan for pentesting tools. One key limitation of the approach is that not all vulnerabilities can be treated equally. Some security configuration related issues may be relatively easier to exploit, whereas security problems related to native components such as memory overflow require domain expertise and effort to exploit. Sarraute *et. al.* [12] encode the pentesting problem as a partially observable Markov decision process (POMDP). The authors use empirical evaluation on a pair of machines to validate the solution. The POMDP based solution does not scale well on a large network. Authors [13] capture uncertainty in the results of attack action using PDDL frameworks that include probabilistic effects. Reinforcement Learning (RL) provides an alternate framework where agents interact with the environment and learn the optimal policy using a trial-and-error approach. It has been successfully used in many games (Atari, Mario) with performance on par or better than human agents. RL based model for pentesting has been used in recent research works [14], [15]. The RL model used by most existing works utilizes the standard Q-Learning approach [16].

An important characteristic of these domains is that features are often handcrafted, and states are low-dimensional and fully

978-1-7281-9916-0/20/$31.00 ©2020 IEEE
DOI 10.1109/MSN50589.2020.00086

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

observable. The Q-learning based solution for finding optimal attack plan will not work well for the domain with high-dimensions (a large scale network with multiple services and vulnerabilities). We utilize Deep-Q Network (DQN) based RL model to make policies learned more generalizable to high-dimensional state space and unseen states. DQN utilizes a deep neural network to parameterize the Q-learning function. In effect, the use of DQN architecture for generating a pentest plan can help in generating better plans for security professionals in a complex network. The reward and transition probability input for existing AI models used in the pentesting domain does not capture the network structure and real-world vulnerability distribution. In our framework, we utilize domain-specific modeling to ensure that reward distribution and transition probability associated with different actions are correlated with the severity of security vulnerability and difficulty for exploiting different vulnerabilities for a given network setup. We utilize the scan information from reconnaissance tools to generate an attack graph, which helps in the identification of dependencies between different vulnerabilities. The attack graph is parsed to obtain the parameters required for the RL framework. The policies obtained using the RL algorithm can help in guiding the pentest procedure.

The key contributions of this research work are as follows:

- ASAP framework identifies the dependencies between vulnerabilities, and network connectivity to provide domain-specific reward modeling. The transition probabilities are associated with a network structure, and access complexity and the reward values are associated with CVSS scores of vulnerabilities. In effect, the RL parameter modeling captures the real-world pentest assumptions - Section III-C.
- We utilize a Reinforcement Learning (RL) framework based on DQN to learn efficient pentest plans on a large scale network. The attack plans help in reducing the overhead of pentest significantly - order of seconds compared to manual pentest carried out over several hours/days.
- The optimal policies obtained using the RL framework helps in uncovering latent policies (hard to discover in manual pentest) that maximize the overall return of effort invested in a pentest. We utilize a case study - Section V-A to showcase that these policies may not be identified during a manual pentest.

The rest of the paper has been organized as follows. We discuss the related works in Section II, with emphasis on autonomous pentesting. Section III provides details on the attack graph, RL framework with domain-specific modeling. The implementation details of the ASAP framework and state graph generation algorithm have been discussed in Section IV. We provide experimental analysis on an enterprise network to showcase the efficiency and practicality of the ASAP framework in Section V. Finally, we conclude the paper in Section VI and provide directions for the future.

II. Related Work

Cyber threats have increased in frequency dramatically. Currently, there is a high demand for cybersecurity manpower, and organizations are trying to find alternative solutions to automate most of the security-related tasks. If penetration testing can be done autonomously, it can relieve the burden facing the cyber-workforce. In most attempts to automate penetration testing, an attack plan was implemented. Earlier work for attack planning was able to use PDDL description language to bridge penetration testing and planning [11]. This also allowed an increase in flexibility for integration for penetration testing systems but was unable to handle uncertainty [12]. Uncertainty could be many things in the network, such as missing updates or new services. Other methods such as Partially observable Markov decision process (POMDP) solvers did not scale well for large-sized networks but also needed accurate probability distributions, as stated in [12]. An attempt to divide the network into segments or clusters seemed promising to resolve the scalability issue with POMDP. Using MDP was beneficial since the exploit action was not known ahead of time, which allows it to be a stochastic environment. The expected reward or value from POMDP or MDP would determine what route a hacker might take when attempting to gain access to the system—being able to assign an MDP mapping allowed for Reinforcement Learning methods (RL) to be used in the optimal attack path problem. RL in autonomous penetration testing creates the best policy through its interactions in its environment without prior knowledge. Using an RL environment can capture the complexity and uncertainty of penetration testing [15]. Previous work using RL [14], [15], [17], [18] have mostly been proven successful with relatively small networks. Although RL has shown the ability to find the optimal attack path, it requires an accurate model of exploit outcomes, and realistic training simulator for agents [14]. Moreover, Zennaro and Erdodi [19], have used RL in the context of capture the flag competition to solve cybersecurity challenges and obtain the secret flag. All of the previously mentioned works did not use the vulnerability information in the network, and they do not build a correlation between those vulnerabilities and the transition probabilities like this paper. Also, this paper utilizes DQN to learn efficient pentesting plans that will be scalable on large networks, which other works failed to achieve.

III. System and Models

A. Attack Model

Consider a multi-tiered network under the scope of network pentest in Figure 1. The goal of the pentest is to target the user's Personally Identifiable Information (PII) present on the database server behind the firewall. The Web Server is present on the public network. The attack can be carried out using two different paths, a) *Attack Path 1* involves exploiting a known vulnerability on Web Server and using elevated privileges on Web Server to target vulnerability present on the Database Server, b) *Attack Path 2* involves setting up a malicious website

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

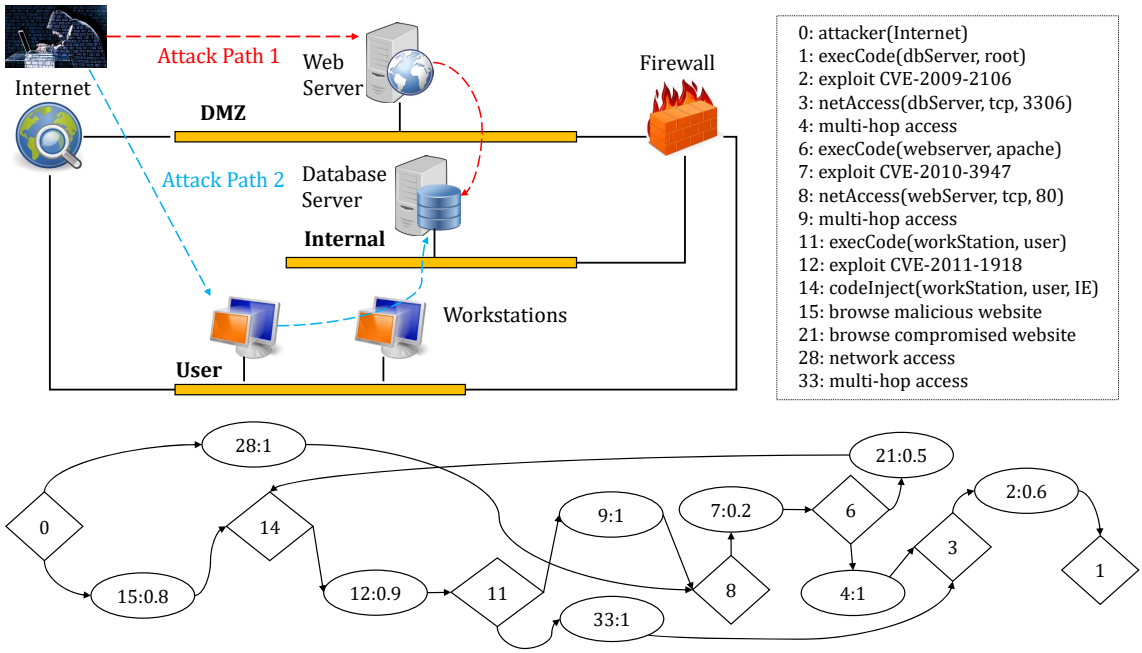

Figure 1: The progression of attacks in case of a network pentest. The target environment in scope has sensitive data stored in SQL backend. The goal of pentest is to access the datastore.

(spear phishing style attack), which when visited by company users, can lead to code injection vulnerability on a user's workstation, and eventually penetration testing team can use workstations as a pivot to target vulnerability on the Database Server.

Manual penetration testing: Current penetration testing is done manually. The pentest broadly involves the following:

1) **Scoping and Reconnaissance:** This involves creating a scope for the security assessment and scanning of the network. The pentester gathers information about the target environment using reconnaissance tools such as nmap [20].
2) **Vulnerability Analysis:** The target environment can have known vulnerabilities, which can be identified using vulnerability scanning tools such as Nessus [21], OpenVAS [22], or unknown vulnerabilities, which require exploration by the pentester using tools such as Burpsuite [23].
3) **Exploitation and Reporting:** This phase involves the attack using payloads based on custom scripts, or the use of tools such as metasploit [24] to exploit the vulnerability. This phase also covers evidence gathering and creation of a report explaining the findings in detail.

The value obtained from the pentest depends on the skillset of security professionals. Identification and exploitation of software and configuration flaws also require some understanding of system functionality, access control, and data flow. In a time-bound security assessment, the creation of a mapping between vulnerabilities and how to exploit them can be challenging, especially if the assessment scope is quite large. Moreover, each pentester has a cognitive bias formed by experience, for instance, the first instinct of the pentester might be to check authentication flaws, followed by access control issues, session management, datastore issues, application logic flaws, etc. If the tester can cover only a few of these focus areas in a time-bound test, it is easier to miss some otherwise easy to find issues. This serves as a motivation for the use of autonomous tools and techniques in the field of pentensting.

B. Attack Graph

Attack graphs help the network administrator to analyze the connectivity between network services, vulnerabilities present on the services, and dependencies between the vulnerabilities [9], [10], [25]. The paths in the attack graph showcase the progression of attacks in the system. Based on the analysis of the attack graph in Figure 1, the initial state of the attacker is *attackerLocated(internet)*, and goal of attacker is to execute code on Database Server, i.e., *execCode(dbServer, root)*. An attack graph can be defined as $G = (N, E)$ consists of nodes N representing vulnerabilities, attacker's privilege state, and edges E leading to transition between different privilege states in the network. A detailed description of generating scalable attack graphs has been provided by Chowdhary *et. al.* [6].

C. Reinforcement Learning

This paper utilizes an artificial intelligence algorithm based on reinforcement learning to identify the attack path that maximizes the reward value for the pentester. Reinforcement learning refers to a problem where an agent learns from an environment by trial-and-error interactions [26]. We observe such scenarios in security assessments where the security

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

professionals utilize several variants of attack payloads against a vulnerability before identifying a valid proof of concepts to showcase that it can be exploited. The reinforcement learning problem can be solved using statistical techniques and dynamic programming to estimate the utilities for taking actions in state space. Formally, reinforcement learning can be defined using tuple (S, A, R, τ, Π).

- S=$\{s_1, s_2, s_3, .., s_n\}$ represents the system states. The states in our model represent the privilege of the pentester, e.g., s_1 = (userm ftp), s_2 = (root, ftp), which means attacker had user privilege on FTP service in state s_1, and root privilege in state s_2.
- A=$\{a_1, a_2, ..., a_k\}$ represent the action taken in current state. We utilize vulnerability exploitation as actions in our model. For instance attacker can transition from (user, Web Server) to (root, Web Server) by taking action a_1 = exploit (CVE-2010-3947), which corresponds to a vulnerability present of FTP service.
- $\tau(s_1, a_1, s_2)$ represent the state transition probability. By taking action a_1 in state s_1 the pentester can transition to s_2 with a certain probability, depending on how difficult it is to exploit a certain vulnerability.
- $R(s_1, a_1)$ is the immediate reward obtained by an agent for taking action a_1 in state s_1. We utilize the Common Vulnerability Scoring System (CVSS) [27] to assign reward value for (state, action) pairs in our reinforcement learning model.
- Π known as a policy which is a strategy employed by the agent to learn next action based on current state. We employ a well-known model-free reinforcement learning algorithm Q-learning [16] to learn the agent's policy.

The policy obtained from reinforcement learning formulation is the attack plan for the pentest. The plan guides the security professional what next action to take once they have obtained a certain privilege in the network by exploiting a security vulnerability. We define some additional notation for Q-learning. $Q^*(s, a)$ is expected discounted reward for taking action a in state s. $V^*(s) = max_a Q^*(s, a)$ is the optimal value obtained by taking best action initially, and continuing to choose best actions in subsequent states. The value of Q-function can be estimated using recursive equation defined below. The value γ is discount factor which characterises agent's preference for future reward compared to immediate reward.

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} \tau(s, a, s') max'_a Q^*(s', a') \quad (1)$$

The Q-learning as shown in equation above helps in finding an optimal policy to maximize the expected reward value. However, it suffers from a lack of generality. The Q-learning agent will not be able to determine the action for states it has not seen. To deal with this situation, we utilize Deep-Q Network (DQN) [28], a neural network-based solution to estimate the Q-value function. The approximate value function is parameterized as $Q(s, a; \theta_i)$ using deep convolutional neural network, where θ_i represents the model parameters of the Q-network at iteration *i*. Q-network follows an experience replay approach, where agent's experience $e_t = \{s_t, a_t, r_t, s_{t+1}\}$, are stored in the dataset $D_t = \{e_1, e_2, .., e_t\}$. During the learning phase, the Q-learning updates are applied to experience samples drawn uniformly and randomly from a pool of stored samples [29].

Table I: CVSS Score and Access Complexity Description of vulnerabilities in example network in Figure 1

VM	Vulnerability	CVE	CVSS	AC
Web Server	Multi-hop Access	CVE-2010-3947	4.3	MEDIUM
DB Server	Code Execution	CVE-2009-2106	8.5	LOW
Workstation	Code Injection	CVE-2011-1918	7.5	LOW

Domain Specific Modeling: The CVSS system is well-known for tracking known vulnerabilities. Each vulnerability is assigned a score that determines the severity of the vulnerability. Moreover, there is metadata associated with each vulnerability, such as Access Complexity (AC), which determines how easy or difficult it is to exploit it. For instance, if AC is low, it is easier to exploit the vulnerability. The range of CVSS score is $(0, 10]$, and AC = `{LOW, MEDIUM, HIGH}`.

Table I captures the CVSS metrics for the network presented in threat model described in Figure 1. It requires some software configuration knowledge and security skills to exploit Web Server vulnerability; hence access complexity is `MEDIUM` for this vulnerability. Also, the vulnerability is not as severe as a database compromise, so its CVSS score is *4.3*.

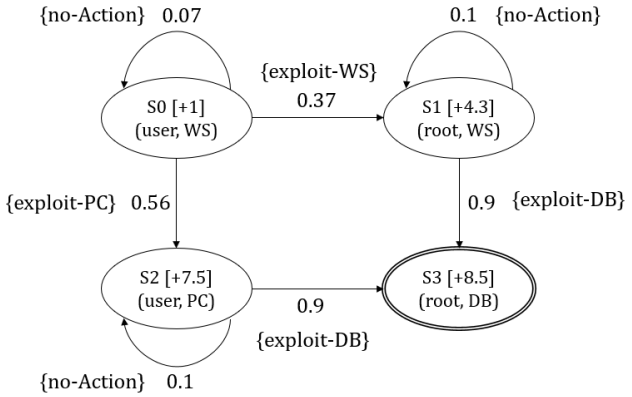

Figure 2: State transition diagram representing privilege states, possible actions, and state transition probabilities used as input for reinforcement learning algorithm

Next, in Figure 2, we describe the state-transition diagram for the threat model in question. We consider four states, S0={user, WS} where the security team has access to user role on the Web Server, S1={root, WS} where the team can compromise the web vulnerability to get root privilege on Web

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

Server. There are one or more possible actions in each state. For instance, in the state S0, the pentester can choose to take {no-Action}, exploit workstation {exploit-PC}, or exploit Web Server, i.e., {exploit-WS}. We assign low-probability to {no-Action}. Each vulnerability exploitation action in a state has provided probability value. We map the access complexity for each exploit to probability value, i.e., `AC={LOW:0.9, MEDIUM:0.6, HIGH:0.3}`. If there are more than one actions in a given state, the probability values are normalized accordingly, e.g., in the state S1, τ(S0, exploit-WS, S1) = $\frac{0.6}{0.6+0.9+0.1} \sim 0.37$, since exploiting workstation (PC) has LOW access complexity, and exploiting Web Server has MEDIUM access complexity. The state S3={root, DB} is the goal state in the model; hence we utilize double circle to represent this state. Each state is also associated with a reward value, which corresponds to the CVSS score of vulnerability, e.g., state S3 has a reward value of [+8.5] since the CVSS score of Code-Execution vulnerability on DB Server is *8.5* as can be seen from Table I. We utilize the model as input for the DQN model, and the result is an optimal penetration testing policy for the underlying network.

IV. Implementation

We utilized vulnerability scanners [21], [22] to scan the vulnerabilities, as can be seen from ASAP architecture in Figure 3. The scan information, along with host configuration, the network topology, is passed to the attack graph generator. The attack graph generator generates an attack graph using MulVAL [30].

Algorithm 1 State Graph Generator

```
1:  procedure PARSE-ATTACK-GRAPH(G{N,E})
2:      CVE-Dict ← {}
3:      S ← {}                                   ▷ State Graph
4:      for n ∈ G.nodes() do
5:          n1,n2 ← ∅
6:          for pred ∈ G.pred(n) do    ▷ Extract Predecessors
7:              if {attacker, execCode, netAccess} ∈ n then
8:                  n1 ← S.add_node(n)
9:              end if
10:             if vulExists ∈ n then
11:                 n.getAttrib(cve)
12:                 CVE-Dict.add(n, cve)
13:             end if
14:         end for
15:         for pred ∈ G.pred(n) do      ▷ Extract Successors
16:             if {attacker, execCode, netAccess} ∈ n then
17:                 n2 ← S.add_node(n)
18:             end if
19:         end for
20:         S.add_edge(n1, n2, CVE-Dict(n1))
21:     end for
22: end procedure
```

A. State Graph

The attack graph is stored in the form of an XML file, which can be parsed to extract key parameters for the reinforcement learning algorithm. As shown in algorithm 1, we parse the attack graph and extract the predecessors, and successors of each node lines *6,15*. If the extracted nodes are privilege nodes, e.g., {execCode, netAccess} - lines *7,16*, we store them as nodes of state graph *S*. The state graph is a *directed* graph representing key privilege transition in a network. Additionally, if the predecessor node that leads to exploit is a vulnerability node {vulExists}, we extract the attributes of the vulnerability such as CVSS score and access complexity by a call to *n.getAttrib()*, and store the information in CVE-Dict for reference - lines *10-13*. After identification of predecessor and successor nodes of interest, we add an edge between them and mark edge-label as vulnerability attributes line *20*. The information from the state graph *S* is used for populating the transition probabilities of the state transition matrix, and numeric reward values for the reward matrix.

B. Attack Plan Generation and Validation

The parameters of the state graph help in the creation of input structure {S, A, R, τ, Π} for reinforcement learning (RL) algorithm. The edge attribute from each node serves as action, e.g., if node n1 has two outgoing edges to n2, n3, the actions are vulnerability exploits that lead to transition (n1,n2) and (n1,n3). The transition probabilities are normalized access complexity values as described in Section III-C. Additionally, we add a low-probability self-loop for each node except goal node to indicate the case where pentester takes no action. The reward matrix is populated based on attributes of each destination node in the edge of the directed graph *S*. The value of the reward is CVSS score [31] of the vulnerability that led to transition (n1 $\rightarrow$ n2). Additionally, we use the ELK server [32] to obtain evidence of successful exploit. The logs can also help in discovering information about unknown vulnerabilities such as weak session management, and unencrypted network transmission. The threat information along with state graph information is passed to the RL plan generator as shown in Figure 3. Once the attack plan is generated, we validate the plans using python wrapper for well known Metasploit framework [24]. The validation of the pentest thus obtained can be used for suggesting the necessary remediation plan to the target organization. Based on the patches deployed by the network infrastructure or security team of the target organization, the attack graph can be updated and the system can be retested. This creates an end-to-end autonomous security assessment and feedback cycle.

V. Performance Evaluation

In this section, we discuss the experimental analysis performed for establishing usability and scalability of the ASAP framework. The network scan was performed using standard tools like Nessus [21], and OpenVAS [22]. The information was provided to attack graph generation tool MulVAL [30]. We utilized the network configuration, vulnerability parameters

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

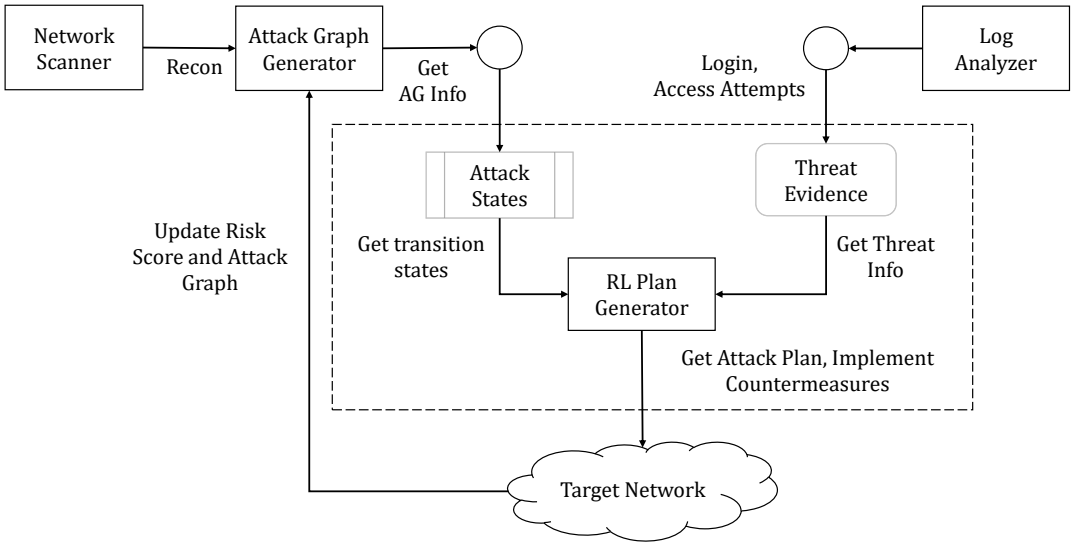

Figure 3: ASAP architecture and data flow. The scanning information is passed to attack graph generator. The core threat analysis component based on Reinforcement Learning utilizes attack graph and log information to create attack plan.

such as CVSS score, access complexity to create a transition, and reward matrix using Algorithm 1. The python wrapper of the network library [33] was utilized for attack graph parsing and state graph generation. The output of the algorithm was represented using Yang Modeling Language (YAML) file. We utilized PyTorch framework [34] for encoding DQN agent [28].

A. *Case Study: Pentesting Enterprise Network*

We simulated an enterprise network with an industrial control system (Subnet2), and IoT devices (Subnet3). The network comprised of 16 hosts, three networks (Net1-3). A mixture of Windows and Linux systems were utilized for experimental simulation. The network is comprised of four key services (SSH, FTP, HTTP, and SMTP), as shown in Figure 4. The pentesting team was provided the goal of compromising email information, i.e., vulnerability present on SMTP service and infiltration on IoT subsystem using vulnerability present in the gateway machine. Each pair in the network architecture depicts the order of network, host. For generality we considered five sub-networks for encoding the (network, host) information pair, e.g, (5,0) represents *subnet 5*, *host 0*.

Figure 5 provides information on the runtime of the DQN agent over the enterprise network state-space representation. We varied the discount factor from $\gamma = 0.6$ to $\gamma = 0.99$. The lower value of the discount factor means that the reward value of terminal states does not have a substantial impact on current state reward value, and a higher value of discount factor means higher reward value in terminal states affects the value of reward obtained by an agent in the current state. We also utilized different batch sizes (BS) to observe the impact on learning optimal policy.

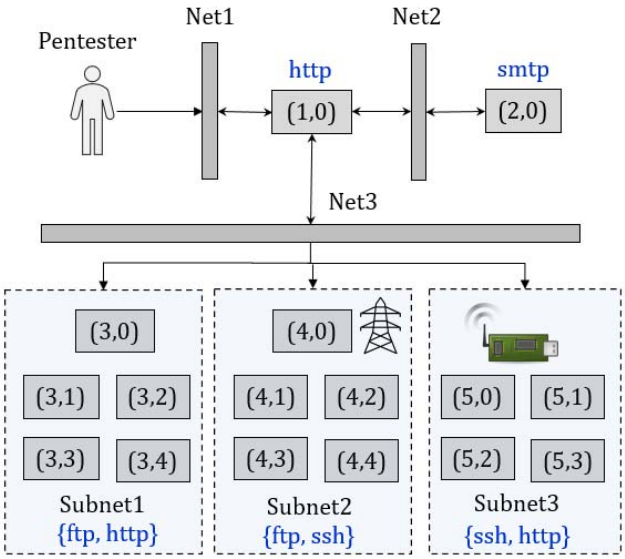

Figure 4: A case study of using DQN based planning algorithm to perform a penetration test on an enterprise network with Industrial Control System (ICS) and Internet of Things (IoT) infrastructure

The DQN algorithm achieved convergence fairly quickly for different values of γ. The convergence time fluctuated between 2.5(s) and 4(s) for value of $\gamma = \{0.6, 0.7\}$. We observed fast convergence for $\gamma = 0.8$. The convergence can be attributed to the fact that agent attributes a higher probability on the exploration of future states. For values of

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

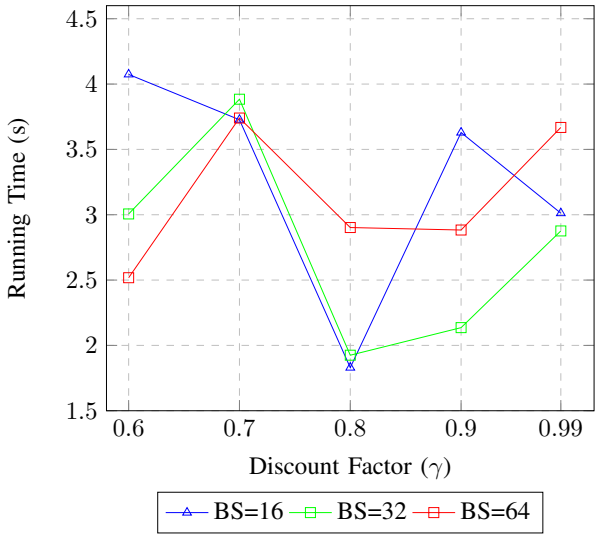

Figure 5: Experimental Analysis of DQN based Reinforcement Learning framework for identification of optimal pentesting policy with varying range of discount factor γ and batch sizes (BS)

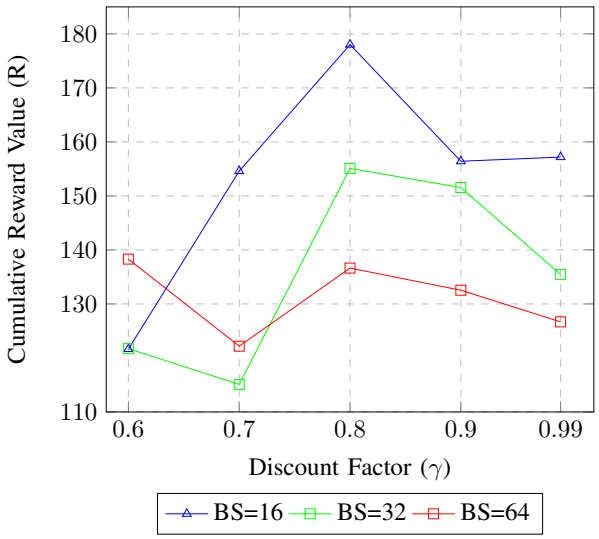

Figure 6: Experimental Analysis of DQN reward obtained by variation of discount factor γ and batch sizes (BS)

$\gamma = \{0.9, 0.99\}$ the agent fails to exploit the current states, which serve as a pivot for future states; hence the algorithm takes more time on convergence as can be observed from Figure 5. Moreover, we observed that with BS=16 the agent achieved faster convergence compared to BS=$\{32, 64\}$. This may, however, not be the case for a large scale network. We observed a similar pattern for Cumulative Reward Value (R). As can be seen in Figure 6, the agent achieves higher reward value for $\gamma = 0.8$. For the BS=16, the agent achieved reward $\sim$ 178, a reward value of 155 for BS=32, and reward value $\sim$ 136 for BS=64. The reward value diminishes considerably for higher values of γ for all batch sizes. The reward value oscillates between $\sim \{128, 158\}$ for $\gamma = \{0.9, 0.99\}$. Thus, the $\{\gamma = 0.8$, BS=16$\}$ provides optimal convergence time and reward values for performing pentest on the simulated enterprise network. We observed the optimal policy Π for the pentest is $(e\_http, (1, 0)) \rightarrow (e\_smtp, (2, 0)) \rightarrow (e\_http, (3, 1)) \rightarrow (e\_ssh, (5, 0))$. The intuition in case of manual pentest is to first try to exploit sub-networks with multiple services, such as $Subnet\{1\text{-}3\}$, i.e., $(e\_http, (3, 1))$ and $(e\_ftp, (3, 2))$ once the pentester is able to obtain initial access to the network via exploitation of $(e\_http, (1, 0))$. However, the ASAP framework proves that this may not be an ideal approach, and the exploitation of SMTP service before targeting $Subnet\{1\text{-}3\}$ is a better approach (plan) for obtaining the highest cumulative reward. This can be explained by the fact that while there are more services associated with ICS and IoT environment subnets, they might be difficult to exploit, compared to targeting subnets with a smaller number of services which are relatively easier to exploit.

B. Scalability Analysis on a Large Network

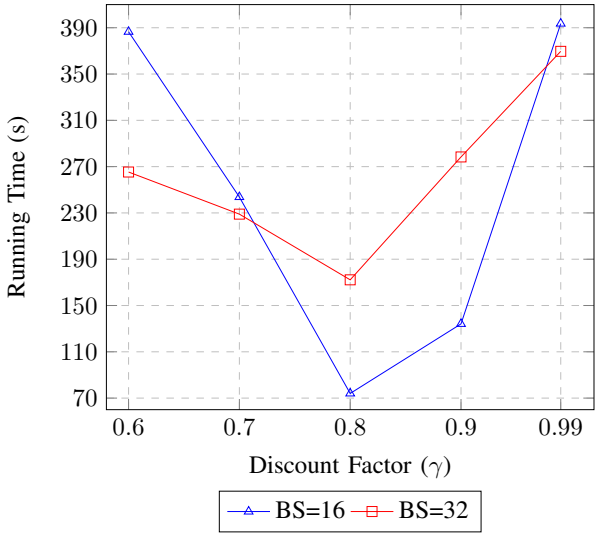

Figure 7: Scalability analysis of ASAP framework on a large scale network with 300 hosts and evenly distributed vulnerabilities

We utilized a simulated flat network with 300 hosts and three vulnerabilities (CVE-2011-0411, CVE-2013-2566, CVE-2011-1431) uniformly and randomly distributed amongst the target hosts. The access complexity of vulnerabilities was {MEDIUM, HARD}. The goal of the experiment was to showcase the scalability of the framework in an environment where deciding between exploration vs. exploitation is not trivial. In a real-world environment, when a network is pretty well secured, the pentesting team may often end up investing too much time trying to triage a difficult to exploit security issue and end up with the low return of investment (ROI)

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.

for the pentest. ASAP framework was able to provide an optimal attack plan within $\sim$ 70(s) for this network when BS size was set to 16. This is a significant improvement over research works utilizing autonomous methods for performing pentesting [12], where the pentest took $\sim$ 300(s) for 7 hosts. Even for the worst-case scenario's where γ value is too low or too high, the algorithm converges and generates an attack plan in $\sim$ 350-400(s). The experiment not only generalizes the parameter settings {BS=16, γ=0.8}, but also saves effort on performing pentest manually by several orders of magnitude. Hence, this experiment establishes the scalability of ASAP framework on a large network.

VI. Conclusion

We introduced an autonomous security analysis framework that helps in reducing the manual effort invested in penetration testing. The framework utilizes network configuration and vulnerability information first to generate an attack graph, and then extract the attacker's privilege based on state transition information in the form of a Reinforcement Learning (RL) framework. ASAP utilizes domain-specific reward and transition probability modeling to capture the real-world settings inherent in a pentesting exercise. The attack plans generated using ASAP help uncover latent attack paths that might have remained undiscovered in a manual pentest. The framework generalizes well to different kinds of network scenarios and scales well on a large network compared to existing research. We have not considered scenarios associated with unknown vulnerabilities, including APT scenarios and pentest drills involving both the red team and blue team (purple team) scenarios in the scope of this paper. We plan to explore these scenarios as a part of future work. The code and data for this research work can be found at `https://github.com/ankur8931/asap.git`

VII. Acknowledgement

This research is supported in part by following research grants: Naval Research Lab N0017319-1-G002, NSF DGE-1723440, OAC-1642031.

References

[1] C. Sausalito, "Global cybersecurity spending predicted to exceed 1 trillion$ from 2017-2021," 2019.
[2] MarketsandMarkets, "Penetration testing market by component," 2020.
[3] C. Sausalito, "Cybersecurity talent crunch to create 3.5 million unfilled jobs globally by 2021," 2019.
[4] D. L. Poole, A. K. Mackworth, and R. Goebel, *Computational intelligence: a logical approach*. Oxford University Press New York, 1998, vol. 1.
[5] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019.
[6] A. Chowdhary, S. Pisharody, and D. Huang, "Sdn based scalable mtd solution in cloud network," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, 2016, pp. 27–36.
[7] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A survey of moving target defenses for network security," *IEEE Communications Surveys & Tutorials*, 2020.
[8] A. Chowdhary, S. Sengupta, D. Huang, and S. Kambhampati, "Markov game modeling of moving target defense for strategic detection of threats in cloud networks," *arXiv preprint arXiv:1812.09660*, 2018.
[9] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 336–345.
[10] A. Sabur, A. Chowdhary, D. Huang, M. Kang, A. Kim, and A. Velazquez, "S3: A dfw-based scalable security state analysis framework for large-scale data center networks," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 473–485.
[11] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," *arXiv preprint arXiv:1306.4044*, 2013.
[12] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing== pomdp solving?" *arXiv preprint arXiv:1306.4714*, 2013.
[13] C. Sarraute, G. Richarte, and J. Lucángeli Obes, "An algorithm to find optimal attack paths in nondeterministic scenarios," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011, pp. 71–80.
[14] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *arXiv preprint arXiv:1905.05965*, 2019.
[15] M. C. Ghanem and T. M. Chen, "Reinforcement learning for efficient network penetration testing," *Information*, vol. 11, no. 1, p. 6, 2020.
[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[17] M. C. Ghanem and T. M. Chen, "Reinforcement learning for intelligent penetration testing," in *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. IEEE, 2018, pp. 185–192.
[18] S. Chaudhary, A. O'Brien, and S. Xu, "Automated post-breach penetration testing through reinforcement learning," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–2.
[19] F. M. Zennaro and L. Erdodi, "Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular q-learning," *arXiv preprint arXiv:2005.12632*, 2020.
[20] G. Lyon, "Nmap security scanner," *línea] URL: http://nmap.org/[Consulta: 8 de junio de 2012]*, 2014.
[21] R. Rogers, *Nessus network auditing*. Elsevier, 2011.
[22] S. Rahalkar, "Openvas," in *Quick Start Guide to Penetration Testing*. Springer, 2019, pp. 47–71.
[23] J. Kim, "Burp suite: Automating web vulnerability scanning," Ph.D. dissertation, Utica College, 2020.
[24] D. Kennedy, J. O'gorman, D. Kearns, and M. Aharoni, *Metasploit: the penetration tester's guide*. No Starch Press, 2011.
[25] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 198–211, 2013.
[26] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
[27] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Security & Privacy*, vol. 4, no. 6, pp. 85–89, 2006.
[28] Z. Yang, Y. Xie, and Z. Wang, "A theoretical analysis of deep q-learning," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 486–489.
[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[30] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: A logic-based network security analyzer." in *USENIX security symposium*, vol. 8. Baltimore, MD, 2005, pp. 113–128.
[31] K. Scarfone and P. Mell, "An analysis of cvss version 2 vulnerability scoring," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2009, pp. 516–525.
[32] W. Sholihah, S. Pripambudi, and A. Mardiyono, "Log event management server menggunakan elastic search logstash kibana (elk stack)," *JTIM: Jurnal Teknologi Informasi dan Multimedia*, vol. 2, no. 1, pp. 12–20, 2020.
[33] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.
[34] N. Ketkar, "Introduction to pytorch," in *Deep learning with python*. Springer, 2017, pp. 195–208.

Authorized licensed use limited to: The University of Arizona. Downloaded on October 30,2023 at 23:44:11 UTC from IEEE Xplore. Restrictions apply.