IAVS: Intelligent Active Network Vulnerability Scanner

Mohammad H. Bazrafkan[1], Alireza Nowroozi[2], Toktam Ramezanifarkhani[3], Peyman Teymoori[3]

[1] Department of Network Security,
ICT Research Institute (ITRC), Tehran, Iran
mh.bazrafkan@itrc.ac.ir
[2] Department of Media Engineering,
IRIB University, Tehran, Iran
alirezanowroozi@iribu.ac.ir
[3] Department of Informatics,
University of Oslo, Norway
{toktamr|peymant}@ifi.uio.no

Abstract

Network security needs to be assured through runtime active evaluating and assessment. However, active vulnerability scanners suffer from serious deficiencies such as heavy scan traffic during the reconnaissance phase, uncertainty in the environment, and heavy reliance on experts. Generating a blind heavy load of attack packets not only causes usage of network resources, but it also increases the probability of detection by target defense systems and causes failure in finding vulnerabilities. Furthermore, environmental uncertainty increases pointless attempts of vulnerability scanners, which wastes time. Utilizing a decision-making method devised for uncertainty conditions, we present Intelligent Active Network Vulnerability Scanner (IAVS). IAVS is implemented as an extension on Hail Mary, the automatic execution mechanism in the Metasploit toolkit. IAVS learns from previous vulnerability exploitation attempts to select exploit codes purposefully. IAVS not only reduces the role of experts in the process of vulnerability testing, but it also decreases the volume of scanning requests during the reconnaissance phase by integrating the reconnaissance and exploitation phases. Our experimental results indicate a successful decrease in failed attempts. It is also demonstrated that improvements in the results of IAVS correspond directly to the rate of similarity among different vulnerabilities in systems of the target network; that is, the higher the similarity, the better the results of IAVS. Our experiments compared the results of IAVS and those of Hail Mary without the IAVS extension; these results show that IAVS improved Hail Marys successful attempts by around 37%.

1 Introduction

On the one hand, hackers are developing expertise and new tools to penetrate different systems; on the other hand, experts in information security are proposing new solutions to guarantee their systems' security. Based on statistics, the total average of data breaches is over increasing from $3.52 million in 2014, to $3.79 million in 2015, and to $3.92 million in 2019 reported in [13]. These numbers show the importance of information security as a promising field in the following years. Information security teams try to use different methods such as secure architecture design and implementation, risk management, cryptography, and security testing to save their systems and decrease the damage.

Penetration testing is a well-known method in information security applied to the final version of the product. It helps the information security experts analyze their systems from the hackers point of view and consequently find leakages and shortcomings of the system.

Active vulnerability scanning is a method that, in addition to scanning, exploits to check the targets vulnerabilities actively while simple penetration tests may omit the phase of target expoiltation. It is a method for actively evaluating and testing a network's security or an information system by accomplishing an attack from an attacker's perspective [27]. To date, a variety of techniques have been proposed for security testing, each consisting of various steps [7, 4, 2] with three common principles phases that are: 1) reconnaissance target system, 2) identifying vulnerabilities in the system, and 3) exploiting the identified vulnerabilities. Moreover, in traditional penetration testing methodologies, these phases are sequential; this, however, brings some deficiencies. The reconnaissance phase, for instance, could impose heavy traffic on the network. Occasionally, results obtained from a significant volume of this traffic, due to the absence of appropriate exploit codes, are inapplicable. Generating heavy attack traffic can lead to further problems, including 1) wasting network resources, and 2) triggering target networks defense systems, such as intrusion detection and prevention systems, anomaly detectors, and honeypots; this, as a result, interrupts the testing process [9], [5].

In this paper, a method, called Intelligent Active Network Vulnerability Scanner (IAVS), has been proposed for active vulnerability scanning. Active vulnerability scanning aims to overcome the large number of unsuccessful attempts in vulnerability scanners. IAVS proposes a solution to solve this problem. The Decision-making process of IAVS is performed based on the Partially Observable Markov Decision Process model (POMDP), which has been used in some penetration testing research [12, 24, 23]. Sarruate, Buffet, and Hoffmann [23] have described how to use POMDP and elaborate on the value functions of the POMDP to model network graphs for penetration testing. Besides, IAVS exploit codes are chosen intelligently, based on the nature of decision systems executing under uncertain conditions. IAVS intelligently and purposefully prioritizes the exploit codes based on prior attack experiences gained in the same network. Intelligently and purposefully here means that IAVS does not try all related and non-related exploits on a target, but by discovering the target, IAVS uses proper expolits for the target. In this method, exploit code is constructed by importance, risk, popularity, novelty, and vulnerability history. The research innovations are:

- decreasing dependency to cyber security experts,
- increasing the efficiency in identifying targeted network,
- running the identification phase and the attack phase interleaving,
- 37% reduction of unsuccessful execution of exploits.

In the rest of this paper, related research is reviewed in Section 2. Section 3 describes IAVS as our proposed method. Evaluation of the method is presented in Section 4 with the discussion on the procedure and the results. Finally, Section 5 concludes the paper.

2 Related Work

Modeling network and attack graphs are used in both manual and automatic penetration testing. Moreover, the Partially Observable Markov Decision Process model (POMDP) used in some penetration testing research is a basis for the decision-making process in IAVS. In this section, we give an overview of these models. Besides, IAVS is implemented as an extension on Hail Mary, and we will compare them in detail in the evaluation section.

Attack graphs are commonly used to analyze potential attacks in computer networks. An attack graph represents possible attack paths in computer network [12]. Attack graphs have a

wide range of applications in security issues, including secure network designs, risk assessment [17], assets assessment [25], intrusion detection systems [22], vulnerability assessment [10, 31], penetration testing [24, 23]. A major challenge in the notion of attack graphs concerns the limited knowledge regarding the environment and scalability. Other researchers suggest techniques to solve the problem of scalability [16, 32]. Some attack graphs begin from the start point and move towards the target [18, 30], while others spread in the opposite direction [26, 21]. Some attack graphs use model checking techniques. At first, model checking techniques were used to prove that targets were reachable from the start point [20, 21]. They were then used to extract all the possible routes for carrying out the attacks between two different modes [14, 26].

Greenwald and Shanley [11] have attempted to introduce a method for automated generation of multi-step penetration test plans. Using the national vulnerabilities database (NVD), the algorithm derives different vulnerabilities to attack against a given system and categorizes them based on different preconditions. Therefore, any vulnerability is selected as the candidate for execution only if it meets all the required preconditions for successful execution. Exploit code in each phase is selected according to the probability of successful execution of different exploit codes.

Other fields of penetration testing (such as web applications) are using automation methods too. Many researches have been performed regarding web application scanners. For example, different commercial and free scanners are evaluated in [8, 29, 28]. Results of these evaluations, represent two major problems in scanners: weakness of scanners in crawling, and high number of false positive in scanners output. The first problem leads to inability of scanners to access all pages in a web application and the second problem leads to unreliability among users regarding the outputs of these tools.

In [6], four different methods of vulnerability assessment including basic manual penetration testing, organized manual penetration testing, static analysis and automatic penetration testing are compared to each other. The methods are used to detect vulnerabilities in web applications. Results clearly show that the best solution to find weaknesses and vulnerabilities is automatic penetration testing. Notwithstanding this fact, high rate of false positive is still one of the main problems in automatic penetration testing. For instance, in one of the evaluations in this research, about 40% of detected vulnerabilities were false positive.

POMDP which has been used in some penetration testing research [11, 24, 23, 34] is a basis for decision making process of IAVS. Sarraute, Buffet and Hoffmann [23] have fixed the scalability problem to some extent and have described how to use POMDP to model network graphs for penetration testing. In [24], they initially decompose the network graph into single machines based on a 4-level algorithm and then transport the machines' properties to the POMDP and derive the output. They further integrated the outputs yielded by different machines and obtain the best attack path. The suggested solution has the main precondition. They assume pen testers do their job in periodic times, and they know all detailed information about the target network, from the last time. However, in IAVS we do this process automatically.

3 Intelligent Active Network Vulnerability Scanner (IAVS)

Since active vulnerability scanning should be done in real situations, the organization's defense system must be active and so, it detects the scanner exploitation attempts, drop some of them and therfore, it will prevent scanner's activities and thus the network vulnerabilities cannot be detected completely. So, a more intelligent scanner is needed. In this section, we present IAVS. By establishing correspondence between the reconnaissance phase and the exploitation phase, and also deriving experience from past actions, IAVS not only reduces the broadcast traffic but

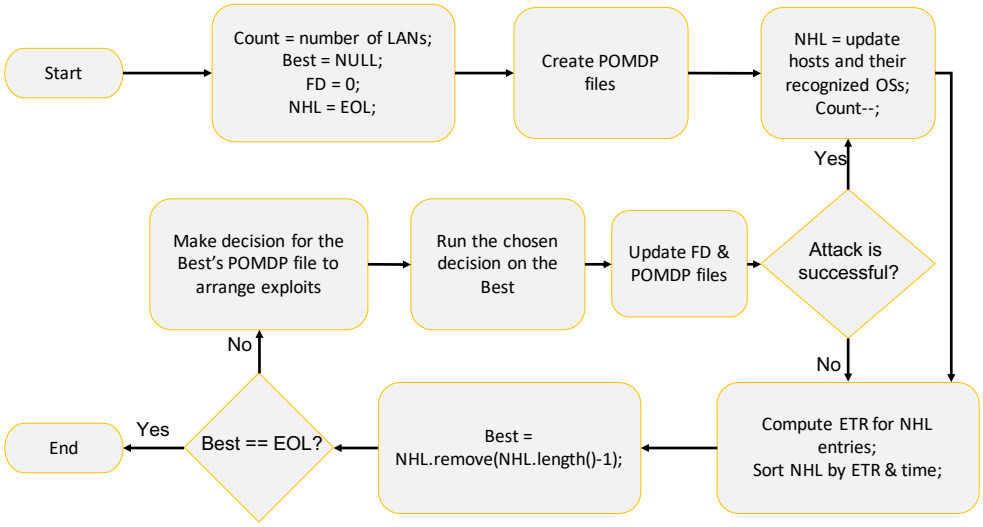

Figure 1: IAVS Flowchart

also reduces the probability of detection by the target networks defense systems. Figure 1 and Algorithm 1 illustrate the IAVS method and its pseudo code respectively. Now, we discuss how it works.

Variable initialization: Primarily, the variable Count needs to be initialized with the number of the target's organization's local networks that we are going to attack. This initial number will be updated based on the information we gain through the process. The Best variable is initialized with Null. The variable FD represents the total number of failed efforts during the method's execution, which is initialized with 0. NHL (Neighbor Host List) refers to the list of available machines, in which the value of End Of List (EOL) is stored at the start of execution. Then, after accessing a machine, all existing devices in the local network are added to NHL during the execution.

POMDP corresponding to all OSs: The list of various OSs in the organization must be available at the beginning. However, since the total number of OSs is limited, this assumption can be avoided by preparing POMDP files corresponding to all OSs.

3.1 POMDP Initialization Files

POMDP initialization files are one of the main inputs to IAVS. In this method, each type of OS has a POMPD file that is common between all machines using that OS.

3.1.1 POMDP Description

The POMDP model is defined by a tuple $< S, A, O, T, o, r, b_0 >$. At each time step, the system is in some state $s \in S$ (the state space), the agent carries out an action $a \in A$ (the action space) that leads to 1) a transition to a state s' according to the transition function $T(s, a, s') = Pr(S'|s, a)$, 2) an observation $o \in O$ (the observation space) based on the observation function $o(s', a, o) = Pr(o|s', a)$, and 3) a scalar reward $r(s, a)$. b_0 is the initial probability distribution in each state. In this problem, the objective is to find the best action as to maximize reward based on past observations and actions. For more information, see [3]. APPL, which is used in

Algorithm 1: IAVS Pseudo Code

```
1: int count; // number of LANs and VLANs
2: string Best;
3: int FD = 0; // failed Attempts
4: string array NHL[] = EOL; // Neighbor host list
5: POMPD.CreateFile(X_1, X_2, ··· , X_n);
6: if isNewLAN == 1 then
7:    NHL[] = nmap.RecogActiveHosts(newLAN); // Recognize active machines in new
      LANs/VLANs
8:    count–;
9: end if
10: for index from 1 to Length do
11:    APPL.ETR(NHL[index]);
12: end for
13: sort(NHL by ETR and arrival time);
14: Best = NHL.remove(NHL.Length-1);
15: if Best == EOL then
16:    Exit;
17: end if
18: arrange = Appl.pomdpEval(Best); // Making decision for the Best's POMDP file
19: Attack.result =Attack.Order(arrange); // Attack Best, based on arrange Order
20: FD += Attack result.failed-attempts; // the count of failed executed exploit code will be
    added to FD
21: POMDP.UpdateFile(X_1, X_2, ··· , X_n); // Update POMDP file for each OS
22: if attack is successful then
23:    Goto 6
24: else
25:    Goto 10
26: end if
```

the implementation of IAVS, is a C++ toolkit to approximate POMDP planning. Originally, it is based on the SARSOP algorithm to solve discrete POMDPs.

3.1.2 Generating POMDP Initialization Files

Each POMDP file has the set of *states*, *actions*, *transition functions*, *observation functions*, *initial belief state*, *reward*, and *cost* as elements. POMDP elements are initialized based on the OS and the existing exploit codes. In the proposed method, each exploit code is mapped to a network port. Given that each port can be either open or close, it will have different states. Thus, exploit codes corresponding to each OS needs to be selected. Executing each exploit code has difficulties such as meeting the required preconditions for its successful execution, probability of detection by defensive systems, etc. The reward is provided for successful execution, which is proportional to the effects, such as meeting the conditions for executing further attacks, gained privileges, and etc., produced by the execution of exploit code in the target system. Therefore, selecting appropriate reward and cost values is a crucial phase.

States refer to different possible conditions in a particular machine. In penetration testing, different conditions of a given machine are presented by the state of ports (open or closed), and

vulnerability of the service executed on those ports (in case of being open). Two particular states are defined: (1) The "terminal" state, which is used to halt the attack's execution and freeze it before further damage. This is because maintaining the attack process appears to do more harm than good. (2) The "Agent Installed" state; a condition in which an agent is loaded on the target system as the result of the successful execution of exploit code.

Actions describe the set of actions that can be performed in each state. Performing Actions entail a change in state, receiving observation from the environment, and bearing cost (and reward in case of successful execution). There are three sets of actions in the proposed method: (1) Actions that are used in port scanning the number of actions relating to port scanning equals the number of ports for which there are exploit codes. (2) Actions that are related to the execution of exploit codes. Clearly, there is a unique action for each exploit codes. (3) There is also a special action named "terminate" that allows moving to the "Terminal" state.

Initial belief state b_0 denotes the probability distribution over the set of different states at the beginning of each execution. Indeed, it represents our pre-judgment of the target system. In the following, we argue that using past observations in future executions implies a change in the *Initial belief state*, corresponding with the past observations and actions.

Each action needs to have a specific reward and cost. Considering that the execution of actions relating to port scanning is identical for all ports, these actions are assigned a similar value. Determining the cost and reward of the exploit code execution is slightly different and requires initialization based on a specific criterion. Based on the Forum of Incident Response and Security Teams (FIRST) proposal, the Common Vulnerability Scoring System (CVSS) provides a universal open and standardized method for rating vulnerabilities. This scoring system has two significant metrics for scoring vulnerabilities: Exploitability and Impact. The former refers to the ease/difficulty of exploiting each vulnerability; the higher the score, the more easily vulnerability can be exploited. The latter measures the impacts of exploiting the vulnerability; the higher the score, the more significant damage is imposed on the target system because of exploiting the vulnerability [15]. These measures can determine the cost and reward of each exploit code. Based on this initialization, the cost of executing exploit codes is a value between 3000 and 12000, and the reward for the successful execution of exploit codes is between 5000 and 15000.These numbers calculated as a constant ratio multiply with the score of each vulnerability. The vulnerability's scores are computed using a well-known vulnerability scoring system called CVSS. In addition to the two preconditions described above, note that cost should not exceed reward for any given exploit code. The cost and reward of "Terminate" action is always 0.

Neighbor Host List (NHL): Following the generation of POMDP files based on the mentioned method, the Neighbor Host List (NHL) is developed. The list covers the entire neighboring active machines along with their corresponding OS. Neighboring is defined here in this way. Machines which are accessible from a machine are its neighbors. There are various tools available for detecting active machines, one of which is NMAP (Network Mapper) [19]. After detecting the neighboring machines and storing them in the NHL, the value of Count should be decreased by 1.

Expected Total Reward (ETR): ETR is given by $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where s_t and a_t denote the agent's state and action at time t, respectively. ETR is calculated for all OSs present in the NHL. APPL, the toolkit used here for approximate POMDP planning, has multiple outputs, one of them is the value of ETR. This output indicates the expected total reward, calculated on the basis of the values defined in the input file. In the proposed method, ETR refers to the total value of active scan against a machine. This value, which is itself a function of several variables, is calculated based on the number of successful and failed efforts.

In this method, due to keeping all values constant and making change in only the initial belief state, changes in ETR is a function of the changes in the initial belief state. After calculating ETR, the machines presenting NHL should be sorted.

Sorting machines in NHL and choosing the Best machine: Accordingly, different OSs are initially ordered based on the ETR, and then different machines with an OS are sorted based on their arrival time. As such, on top of the list exists a machine that not only is a component of the OS with the highest ETR value, but also is imported in the NHL later than the other machines of the same OS type. This machine is considered as the Best machine. In the next step, the machine is removed from the list and put in the variable named Best. Since the value of minus ten is always the last entry in the NHL, selecting it as the Best suggests that there is no machine in the NHL and the execution should be terminated.

Actively scanning the Best machine: After selecting the most appropriate machine, it should be scanned actively. By regulating the best order of exploit attempts based on the input values, APPL tools determine how to attack the target machine. Therefore, the Best machine's POMDP file is given to APPL as the input and output, i.e. the order of executing the Actions, are derived using Pomdpeval.exe that is a tool in APPL.

Up to this point, the order of executing the actions is determined. Now, by applying this derived order of actions, it is time to test if the target machine is vulnerable or not. The vulnerability testing process continues until an exploit code manages to install the agent in the target machine. Executing this action terminates in two ways: either one of the exploit codes is successfully executed, or none of the exploit codes are successful. The process ends by examining all exploit codes. In the next step, it is essential to change the POMDP files and update the FD value using this step's output.

3.2 Updating the POMDP files

As mentioned earlier, 'states' indicate different possible conditions in a particular machine. It is assumed that probabilities of presence in different states are either equal or different. Presence Probability distribution in each state is performed by the 'initial belief state'. Such probability represents our preconceptions regarding the target machine. Since initially there is no information about the target machine, probabilities in each state are considered to be equal. However, after observation, probabilities will be changed. 'Initial belief state' for an OS in the first iteration is as {0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05}. These are twenty equal beliefs which their sum is one in the begining. 'Initial belief state' for the mentioned OS in the 30^{th} iteration is as {0.04274, 0.04274, 0.11950, 0.04274, 0.04274, 0.02819, 0.04274, 0.04274, 0.03115, 0.04274, 0.04274, 0.08654, 0.04274, 0.04274, 0.03115, 0.04274, 0.04274, 0.10514, 0.04274, 0.04274}. As it seen, the sum of beliefs is one, again. In IAVS, due to the fact that all input values remain constant, changes in output is a function of changes in the "initial belief state". Changes in the '*Initial belief state*' are calculated according to the experience of attack against previous machines.

During attacking a machine, executing exploit codes continues until a successful installation of an agent in the target system. so, exploit codes are divided into three types: (Type 1) exploit codes that are executed but failed and the executing exploit codes continues; (Type 2) exploit codes that are executed and managed to load an agent in the target system successfully and executing exploit codes terminates; and (Type 3) the rest of the exploit codes that are not executed.

Using past observations means that the ratio of states that lead to success in exploit codes

of Type 1 to the rest of states decreases, yet this ratio for exploit codes of Type 2 increases. Probability of states corresponding to success of Type 3 codes does not change substantially; yet, given the fact that the sum of probabilities of different states is always "1", the probability of these states will undergo some changes.

In the last step, it is necessary to make decisions regarding going along the path. In case where none of the exploit codes is successful, another machine needs to be selected among the machines in the NHL. This implies that the machine is secured from the vulnerabilities considered in the corresponding POMDP file. If the attack ends with executing an exploit code, then the process should continue.

4 Evaluation

We compare our approach with the proposed method and the current active vulnerability scanning methods for practical evaluations. Hail Mary, provided by Armitage, is one of those most powerful tools using a semi-automated approach, identifies the entire exploit codes corresponding to a target, and then rates and runs them. The evaluation explains the scenarios as examples of the method and the comparison between IAVS and Hail Mary is shown over these different scenarios at the rest of this section.

In Hail Mary, exploit codes are initially classified according to six reliability levels: excellent, great, good, normal, average, and low [1]. Then, starting by the higher levels, the exploit codes available in each level are arranged based on the time they are added to Metasploit, and executed in order. Therefore, the first exploit code corresponds to the highest level added to the Metasploit later than the rest of codes at the same level.

The key assumption in IAVS is similarities between systems of a network and the associating vulnerabilities. This kind of similarity, which is popular in real-world networks, is reported in [33]. We define the term "beneficial similarity" more precisely as the recurrence of vulnerability in two different machines in a network before identifying all the machines' vulnerabilities in-between these two machines. The ratio of beneficial similarity to the entire vulnerable machines constitutes the "percentage of beneficial similarity". How IAVS functions in the networks with different similarities are demonstrated later in this paper.

For evaluation, we consider two evaluation sets, each with four test scenarios. The first evaluation set is explained in more detail to show how IAVS works. In the first evaluation set (1^{st} to 4^{th} test scenarios), the selected network for evaluation includes 59 machines. In the second evaluation set, network of the 5^{th} and 6^{th} scenarios has 74 machines and network of the 7^{th} and 8^{th} scenarios has 78 machines. Since IAVS starts from a machine of the target network. Given N machines in the target network, IAVS ends after taking $N-1$ iterations. The experiments have been carried out on three networks with eight vulnerability distributions. As we will discuss further in the results section, the higher percentage of beneficial similarity results in further improvement in the method's execution.

4.1 The First Evaluation Set: 1^{st} to 4^{th} Test Scenarios

Figure 2 presents a schematic representation of the network of these four test scenarios with the type of OS and the vulnerable service (0 represents the machine is safe). X1, X2, X3, and X4 are the four OSs used in these tests, corresponding to Ubuntu, Windows XP, Windows 7, Windows 2003, respectively. In the POMDP files associated with the current implementation, two exploit codes are common between X2 and X3 and two exploit codes common between X2

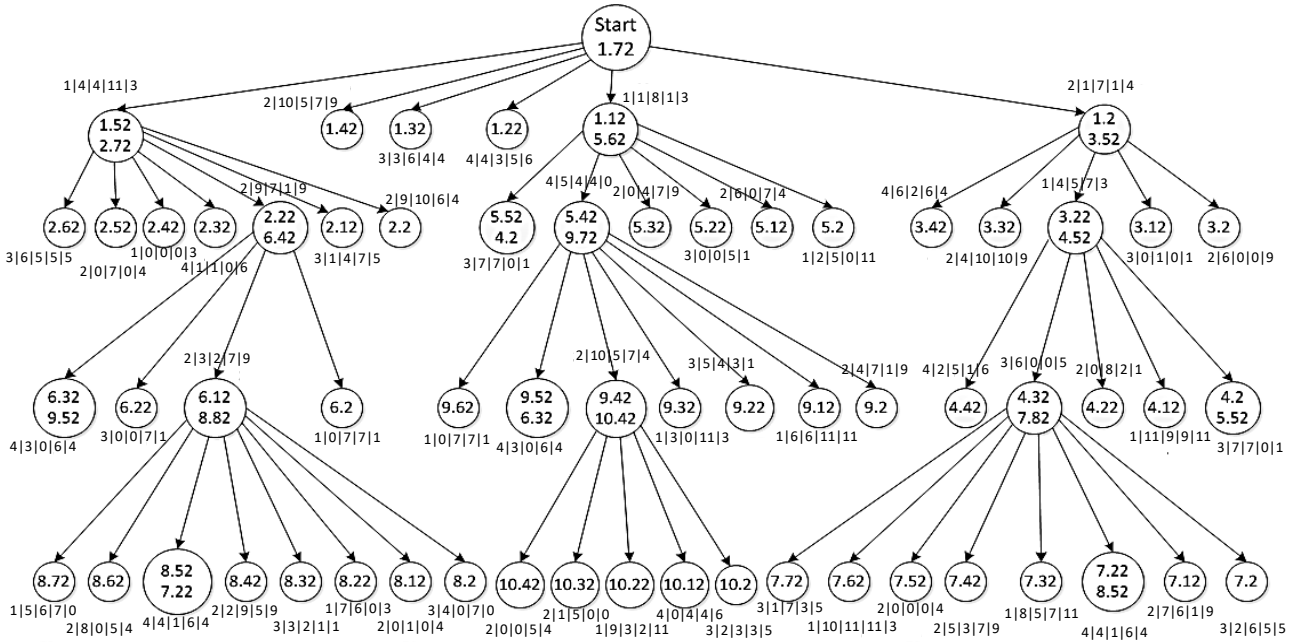

Figure 2: Network graph for the first four test scenarios

and X4 to demonstrate the effect of executing common exploit codes in different OSs on each other.

In Figure 2, numbers associating to the different scenarios are separated by "|" that the first number represents the OS type and the next number shows the vulnerable service of the machine in the 1^{st}, 2^{nd}, 3^{rd} and 4^{th} scenarios, respectively. It should be noted that the algorithm scans all of the nodes in the network(graph in Figure 4), so after complete execution, all nodes are scanned, but the difference is in node selection and attack execution order. Table 1 provides descriptions of each scenario.

To compare the results of IAVS with Hail Mary, the number of failed exploit codes executed by them for each OS calculated and named FD; lesser FD better performance. Safe machines are not vulnerable against their corresponding exploit codes of POMDP file, and unsafe machines are vulnerable against at least one of the used exploit codes in the POMDP file. X1, X2, X3, and X4 are four used OS in the network. Each row of Table 1 shows the failed attempts for each safe and unsafe machine. For example, in the first row of Table 1, there are 58 failed attempts (failed exploit codes) regarding unsafe machines and 22 failed attempts regarding safe machines. The total column shows the sum of safe and unsafe failed attempts. A comparison between the results of executing IAVS and Hail Mary in the first four test scenarios is shown in Table 1. Only in the first test, IAVS result is not better than Hail Mary due to the lack of similarity. This situation is rare in real-world networks. More details on the execution of IAVS using ETR (Expected Total Reward) measure in these four test scenarios are illustrated in Table 1.

Figure 3 illustrates changes in ETR for each four OSs (X1, X2, X3, and X4) over 58 iterations of IAVS in the first test scenario. As shown in Figure 3, increasing in the number of failed

Table 1: IAVS vs. Hail Mary-Results in the 1st, 2nd, 3rd and 4th Test Scenarios

Scenario no.	**Test Scenario 1**			**Test Scenario 2**			**Test Scenario 3**			**Test Scenario 4**		
No. of machines	59			59			59			59		
Beneficial similarity	0%			33%			62%			87%		
Hail Mary	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**
X1	80	22	58	100	22	78	107	33	74	82	22	60
X2	130	60	70	125	50	75	142	40	102	124	10	114
X3	55	28	27	67	28	39	66	21	45	58	7	51
X4	25	6	19	23	6	17	33	6	27	38	6	32
IAVS	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**
X1	87	22	65	82	22	60	56	33	23	41	22	19
X2	148	60	88	119	50	69	89	40	49	44	10	34
X3	75	28	47	60	28	32	50	21	29	23	7	16
X4	33	6	27	19	6	13	19	6	13	17	6	11
Improvement	-18%			11%			38%			59%		

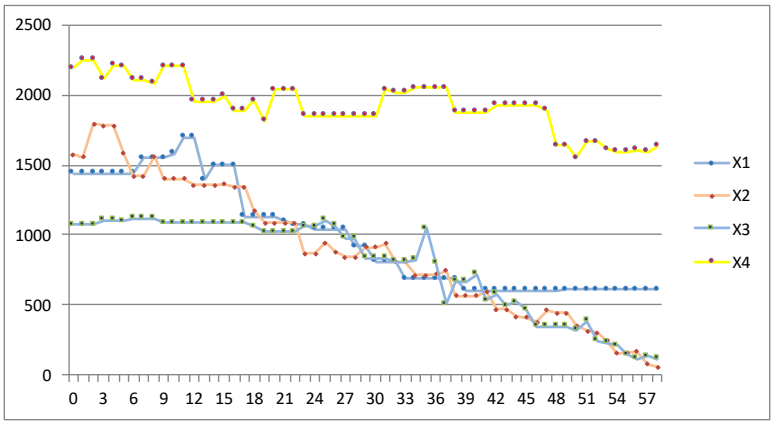

Figure 3: First Test Scenario-ETR Changes for Four OSs in 58 Iterations of IAVS

attempts, which are due to the lack of similarity, has led to a significant decrease in ETR in all OSs (ETR represents the total value of attack on an OS). It indicates that the "expected total reward" had experienced a substantial decrease, which is because not only past observations did fail to result in improving the results, but also misled IAVS.

Whereas IAVS tries to prioritize the execution of successful exploit code, vulnerabilities of the system are arranged so that after the successful execution of exploit code, that exploit code will never be successful unless all the other exploit codes are executed successfully for one time. These rare synthesized conditions are among the conditions under which using the IAVS leads to an increase in failed attempts, which are the worst cases of IAVS. The following describes how in other scenarios in the presence of beneficial similarity, the use of the IAVS results in a considerable decrease in failed attempts and increase effectiveness.

As shown earlier, the improvement occurred by IAVS is directly associated with the rate of similarity between vulnerabilities in the target network, such that in the network without beneficial similarity, failed attempts increased by about 18%. In contrast, in the network with 87% similarity, the use of IAVS resulted in 67% decrease in failed attempts. It is also essential that the improvement of IAVS results over those of Hail Mary, even in a network with 33%

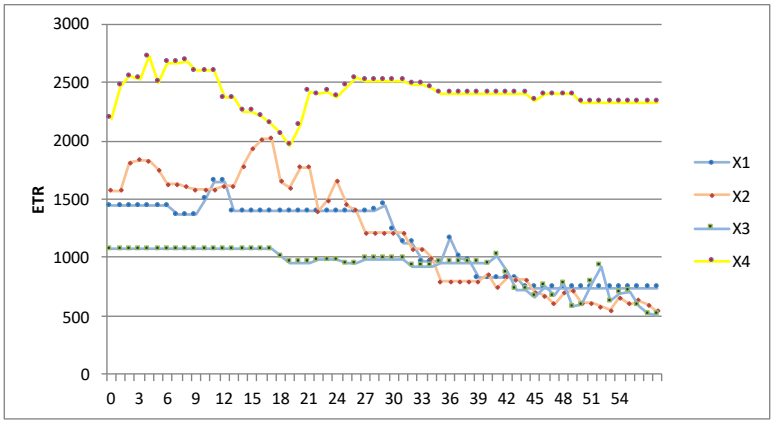

Figure 4: Second Test Scenario-ETR Changes for Four OSs in 58 Iterations of IAVS

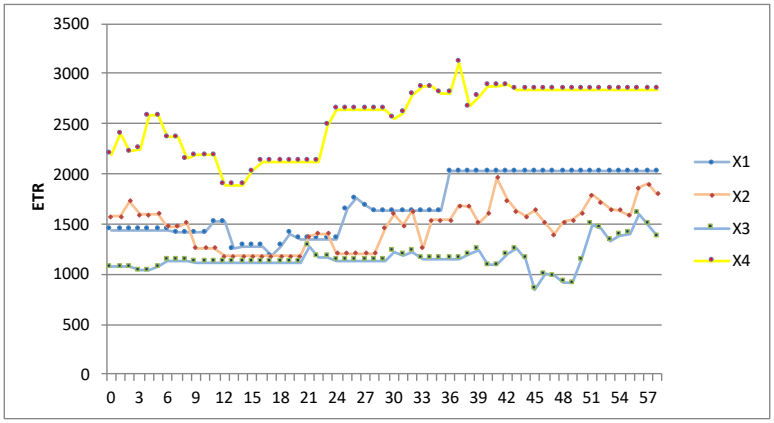

Figure 5: Third Test Scenario-ETR Changes for Four OSs in 58 Iterations of IAVS

beneficial similarity. This implies that in the presence of a slight similarity, which is a normal condition, IAVS leads to improved results. A decrease in failed attempts suggests a decrease in traffic broadcasted to the target network. Note that the traffic avoided in this phase is the attack traffic. Even the slightest decrease could significantly impact the probability of stimulation of defense systems in the target network. IAVS automatically uses observations of past attacks, rates exploit codes, and, after executing attacks, uses the obtained output as experience in future attacks.

In the process of IAVS execution, the ETR value of different OSs changes. Figures 4, 5, and 6 show changes in the ETR value of all OSs over 58 iterations of execution. Figure 4 corresponds to the network with 33% beneficial similarity, Figure 5 to the network with 62% beneficial similarity, and Figure 6 to the network with 87% beneficial similarity.

Experiments demonstrate that increase in the ETR value of different OSs is directly associated with beneficial similarity; the higher the beneficial similarity, the higher the value of ETR. It is due to a decrease in the number of failed attempts, which increases the probability of states leading to successful exploit codes; thus, the lower the number of failed attempts, the higher the value of ETR. It suggests an inverse correlation between variation in ETR value and

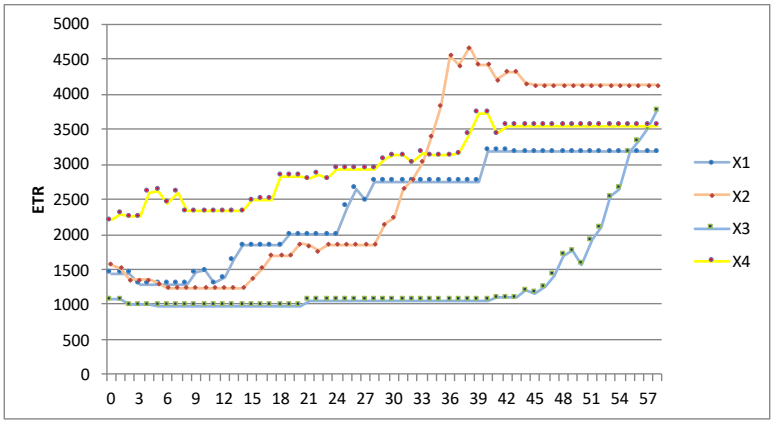

Figure 6: Fourth Test Scenario-ETR Changes for Four OSs in 58 Iterations of IAVS

the number of failed attempts.

X2 and X3 OSs share two exploit codes. Thus, if an attack occurs on either of these two OSs, an experience is obtained regarding these two exploit codes, its effects will also be reflected in the POMDP file of the other OS. For instance, in Figure 5 that corresponds to ETR changes in the scenario with 62% beneficial similarity, by selecting only X3 from iterations 45th to 58th, X2 has also been changed. It is similar for exploit codes common in X2 and X4. In the 6th iteration of the test scenario with 87% beneficial similarity, for example, by selecting X4 from the 6th iterations, the POMDP files corresponding to both OSs (X2 and X4) have been changed.

Variation in the ETR value of different OSs will change the selected machine in IAVS. Occasionally, multiple failed attempts in a machine or unavailability of a machine using the desired OS result in a change in the selected machine. In such cases, IAVS searches for a machine with another OS. Figure 7 shows the selected machine in every 58 iterations of IAVS execution in the network with 87% beneficial similarity.

Up to the 33rd iteration, X4 had the highest ETR; yet, after successfully executing several consecutive iterations of X2, the ETR of this OS was increased, making it the OS with the highest ETR value. As stated before, sometimes IAVS cannot select a machine from an OS with the highest ETR, which is due to the absence of a machine with the desired OS in NHL. Indeed, it explains the difference between the selected OS and the OS with the highest ETR (which should have been selected but due to its absence in NHL another machine has been selected) occurring 46 times.

4.2 The Second Evaluation Set: 5th to 8th Test Scenarios

In this experiment, four different test scenarios were evaluated in two networks with various machines and different vulnerability distribution. The first network used for the 5th and 6th scenarios has 74 machines, and the second network which used for the 7th and 8th scenarios has 79 machines. The results of using Hail Mary and IAVS in these four scenarios are reported in Table 2. Results had shown that even when the network graph is changed, the results are compatible with previous evaluations, and FD decreased with Beneficial similarity growth.

One of the main problems in POMDP is scalability, in which an increase in the number of states results in a considerable decrease in the speed of the method. The aforementioned

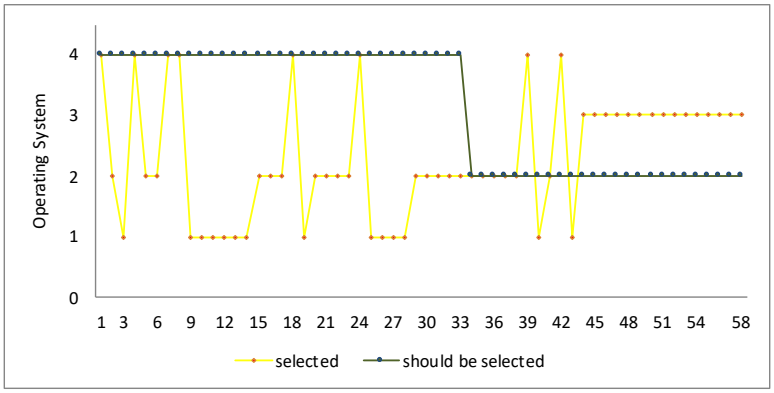

Figure 7: Selected OS Type in Each Iteration

Table 2: IAVS vs. Hail Mary-Results in the 5^{th}, 6^{th}, 7^{th} and 8^{th} Test Scenarios

Scenario no.	**Test Scenario 5**			**Test Scenario 6**			**Test Scenario 7**			**Test Scenario 8**		
No. of machines	73						79					
Beneficial similarity	34%			62%			35%			61%		
Hail Mary	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**
X1	148	33	115	158	33	125	125	33	92	155	22	133
X2	159	30	129	146	30	116	195	30	165	193	40	153
X3	72	21	51	66	14	52	81	21	60	62	14	48
X4	42	12	30	33	12	21	39	12	27	47	12	35
IAVS	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**	**Total**	**Safe**	**Unsafe**
X1	107	33	74	82	33	49	98	33	65	77	22	55
X2	119	30	89	78	30	48	132	30	102	97	40	57
X3	50	21	29	28	14	14	47	21	26	37	14	23
X4	29	12	17	29	12	17	34	12	22	20	12	8
Improvement	-18%			11%			38%			59%		

scalability problem of POMDP causes this problem in IAVS. This overhead can be reduced using the improved versions of POMDP.

5 Conclusion

This paper proposed an intelligent active network vulnerability scanner (IAVS) to solve some common problems in active vulnerability scanning, including generating heavy blind vulnerability tests during the reconnaissance phase and firm reliance on experts. IAVS executes the reconnaissance and exploitation phases in parallel, and thus avoids the generation of heavy attack traffic during reconnaissance. Not only decreasing the resulting traffic prevents network resources waste, but it also reduces the probability of the scanner detection by the target network's defense systems. Such a decrease in traffic is achieved in two ways: (a) identification of ports for which there exists exploit codes and (b) goal-oriented execution of exploit codes. Taking advantage of the POMDP model and APPL tools, recognized as an implementation of the model, IAVS reduces the vulnerability testing's dependency on cyber security experts. POMDP is a decision-making model under uncertainty. Evaluations performed over eight dif-

ferent scenarios on three different networks show that the higher the similarity, the better the method's results. IAVS is implemented on the best and well-known open source vulnerability assessment and exploitaion tool called Hail Mary, and IAVS could improve Hail Mary about 37% of failed attempts in networks with 50% beneficial similarity. Using IAVS resulted in a decrease in broadcast traffic and the number of failed attempts. So, it finds the answer faster with lower cost. There is no report a scanner claim doing this facility. Finding the level of beneficial similarity in real-world networks and computing the best, average, and worst case of IAVS can help us improve the proposed method.

References

[1] Metasploit reliability rankings. https://community.rapid7.com/community/metasploit/blog/2012/12/18/5-tips-to-ensure-safe-penetration-tests-with-metasploit. Accessed: 21-Mar-2020.

[2] Owasp: The open web application security project. https://www.owasp.org. Accessed: 21-Mar-2020.

[3] Partially Observable Markov Decision Processes. http://www.pomdp.org. Accessed: 21-Mar-2020.

[4] PTES: The Penetration Testing Execution Standard. www.pentest-standard.org. Accessed: 21-Mar-2020.

[5] Sultan Alneyadi, Elankayer Sithirasenan, and Vallipuram Muthukkumarasamy. A survey on data leakage prevention systems. *Journal of Network and Computer Applications*, 62:137–152, 2016.

[6] Andrew Austin and Laurie Williams. One technique is not enough: A comparison of vulnerability discovery techniques. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 97–106. IEEE, 2011.

[7] Marta Barceló, Peter Klee, Vincent Ip, Waidat Chan, Russ Spooner, Miguel Angel Dominguez Torres, Rich Jankowski, Anton Chuvakin, Efrain Torres, Michael S Hines, et al. Open-source security testing methodology manual. 2000.

[8] Adam Doupé, Marco Cova, and Giovanni Vigna. Why johnny cant pentest: An analysis of black-box web vulnerability scanners. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131. Springer, 2010.

[9] Gilberto Fernandes, Joel JPC Rodrigues, Luiz Fernando Carvalho, Jalal F Al-Muhtadi, and Mario Lemes Proença. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70(3):447–489, 2019.

[10] Laurent Gallon and Jean-Jacques Bascou. Cvss attack graphs. In *2011 Seventh International Conference on Signal Image Technology & Internet-Based Systems*, pages 24–31. IEEE, 2011.

[11] Lloyd Greenwald and Robert Shanley. Automated planning for remote penetration testing. In *MILCOM 2009-2009 IEEE Military Communications Conference*, pages 1–7. IEEE, 2009.

[12] Todd Heberlein, Matt Bishop, Ebrima Ceesay, Melissa Danforth, CG Senthilkumar, and Tye Stallard. A taxonomy for comparing attack-graph approaches. *Online] http://netsq. com/Documents/AttackGraphPaper. pdf*, 2012.

[13] IBM. Cost of a data breach report. https://www.all-about-security.de/fileadmin/micropages/Fachartikel_28/2019_Cost_of_a_Data_Breach_Report_final.pdf. Accessed: 21-Mar-2020.

[14] Somesh Jha, Oleg Sheyner, and Jeannette Wing. Two formal analyses of attack graphs. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, pages 49–63. IEEE, 2002.

[15] Peter Mell, Karen Scarfone, and Sasha Romanosky. A complete guide to the common vulnerability scoring system version 2.0. In *Published by FIRST-forum of incident response and security teams*, volume 1, page 23, 2007.

[16] Sanjeeb Nanda and Narsingh Deo. A highly scalable model for network attack identification and path prediction. In *Proceedings 2007 IEEE SoutheastCon*, pages 663–668. IEEE, 2007.

[17] Steven Noel, Sushil Jajodia, Lingyu Wang, and Anoop Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1):135–147, 2010.

[18] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.

[19] Ram Sandesh Ramachandruni and Prabaharan Poornachandran. Detecting the network attack vectors on scada systems. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 707–712. IEEE, 2015.

[20] CR Ramakrishnan and R Sekar. Model-based analysis of configuration vulnerabilities 1. *Journal of Computer Security*, 10(1-2):189–209, 2002.

[21] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 156–165. IEEE, 2000.

[22] Sebastian Roschke, Feng Cheng, and Christoph Meinel. High-quality attack graph-based ids correlation. *Logic Journal of IGPL*, 21(4):571–591, 2013.

[23] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Penetration testing== pomdp solving? *arXiv preprint arXiv:1306.4714*, 2013.

[24] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Pomdps make better hackers: Accounting for uncertainty in penetration testing. *arXiv preprint arXiv:1307.8182*, 2013.

[25] Reginald E Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *European Symposium on Research in Computer Security*, pages 18–34. Springer, 2008.

[26] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.

[27] Chaitra N Shivayogimath. An overview of network penetration testing. *International Journal of Research in Engineering and Technology*, 3(7):408–13, 2014.

[28] Larry Suto. Analyzing the effectiveness and coverage of web application security scanners. *San Francisco, October*, 2007.

[29] Larry Suto. Analyzing the accuracy and time costs of web application security scanners. *San Francisco, February*, 2010.

[30] Laura P Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, volume 2, pages 307–321. IEEE, 2001.

[31] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296. Springer, 2008.

[32] Anming Xie, Zhuhua Cai, Cong Tang, Jianbin Hu, and Zhong Chen. Evaluating network security with two-layer attack graphs. In *2009 Annual Computer Security Applications Conference*, pages 127–136. IEEE, 2009.

[33] Su Zhang, Xinming Ou, and John Homer. Effective network vulnerability assessment through model abstraction. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 17–34. Springer, 2011.

[34] Tian-yang Zhou, Yi-chao Zang, Jun-hu Zhu, and Qing-xian Wang. Nig-ap: a new method for automated penetration testing. *Frontiers of Information Technology & Electronic Engineering*, 20(9):1277–1288, 2019.