Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

NIG-AP: a new method for automated penetration testing*

Tian-yang ZHOU[1], Yi-chao ZANG[†‡1], Jun-hu ZHU[2], Qing-xian WANG[1]

[1]*State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China*

[2]*China National Digital Switching System Engineering and Technological R&D Center, Zhengzhou 450001, China*

[†]E-mail: zangyeechao@sina.com

Received Sept. 3, 2018; Revision accepted Feb. 1, 2019; Crosschecked Sept. 4, 2019

Abstract: Penetration testing offers strong advantages in the discovery of hidden vulnerabilities in a network and assessing network security. However, it can be carried out by only security analysts, which costs considerable time and money. The natural way to deal with the above problem is automated penetration testing, the essential part of which is automated attack planning. Although previous studies have explored various ways to discover attack paths, all of them require perfect network information beforehand, which is contradictory to realistic penetration testing scenarios. To vividly mimic intruders to find all possible attack paths hidden in a network from the perspective of hackers, we propose a network information gain based automated attack planning (NIG-AP) algorithm to achieve autonomous attack path discovery. The algorithm formalizes penetration testing as a Markov decision process and uses network information to obtain the reward, which guides an agent to choose the best response actions to discover hidden attack paths from the intruder's perspective. Experimental results reveal that the proposed algorithm demonstrates substantial improvement in training time and effectiveness when mining attack paths.

Key words: Penetration testing; Reinforcement learning; Classical planning; Partially observable Markov decision process

https://doi.org/10.1631/FITEE.1800532 **CLC number:** TP393.08

1 Introduction

Over the past decades, penetration testing has played an important role in assessing network security by seeking possible exploitable vulnerabilities in applications or operating systems. However, it is still a domain-specific area, making it hard for non-experts to carry out regular and systematic security tests. Corporations or organizations spent a great deal of money inviting security analysts to conduct penetration testing of their networks, which incurs a significant time cost as well. Automated attack planning, however, can indicate the direction along which to cope with these vulnerabilities.

Automated attack planning (Futoransky et al., 2010), also known as "cyber security planning" in the artificial intelligence planning community (Zhuang et al., 2017), aims at automatically finding all possible attack paths that seek to retrieve sensitive information. Schneier (1999) first proposed an attack tree model to discover attack paths where the root node represents the goal of the attack and child nodes represent requirements of satisfying the parent nodes. The limitation of the attack tree model lies in its single-objective-oriented attack path discovery, which does not accommodate multi-objective scenarios. To solve this problem, Sheyner et al. (2002) devised an attack graph model, which represents the collection of possible penetration scenarios in a specific computer network, where each penetration scenario is a sequence of actions taken by the intruders. Roberts et al. (2011) combined personal

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61502528)

ORCID: Yi-chao ZANG, http://orcid.org/0000-0002-1791-586X

psychology and the attack graph model to implement personal vulnerability analysis. Though the attack graph model is widely used in security evaluations, it can be applied only to small network penetration testing scenarios because of the state explosion problem. To overcome this problem, Obes et al. (2013) transformed the penetration scenario into the plan domain definition language (PDDL) and adopted a classical planning algorithm to find attack paths. The penetration scenario is composed of two parts, the "problem.pddl" and "domain.pddl" files (Fox and Long, 2003), where the former is used to describe the network configuration information and the latter is used to describe vulnerability and action information. Khan and Parkinson (2017) used PDDL to implement automated vulnerability assessment. PDDL has achieved great success in the commercial product "Core Impact" (Core Security, 2019), but there are still some limitations to this type of method, in which it needs perfect information about a network scenario and it yields uncertainty. To take uncertainty into account in a network penetration testing scenario, Sarraute et al. (2013) adopted partially observable Markov decision processes (POMDPs) to formalize the attack planning problem so that action uncertainty can be incorporated into attack path generation. Inspired by previous works, Alexander Pretschner (2017) introduced POMDP into industrial control systems, trying to automatically verify the security of industrial control systems. Meanwhile, Shmaryahu et al. (2017) modeled penetration testing as a partially observable contingent problem and devised a contingent planning tree algorithm to plan the attack paths. Instead of introducing new models, Sarraute et al. (2011) merged probability into classical planning algorithms to find the optimal attack paths in nondeterministic scenarios. Even though uncertainty is considered, solving POMDP for a large network is still unfeasible (a network with 20 computers is already unfeasible for the POMDP solver). To overcome this problem, Sarraute et al. (2012) built a 4AL decomposition algorithm to slice a large network into smaller ones according to the network structure and solve each of them by POMDP. Even though the POMDP model has achieved great success in the academic domain, it is based on a rigid hypothesis whereby the network structure and software configuration remain unchanged between any two penetration tests, which makes it hard to be applied to realistic penetration scenarios. There are some other research works that have contributed to the field of automated attack planning from an engineering perspective. Samant (2011) offered a fast, reliable, and automated testing tool to perform a protocol-oriented security test. Stefinko and Piskuzub (2017) developed an expert system composed of an inference engine and a knowledge base to accomplish modern automated penetration testing. Backes et al. (2017) adopted mitigation analysis to implement a holistic security assessment in a simulation penetration test scenario, but it lacked a solid theoretical understanding. Steinmetz (2016) proposed an attack planning algorithm from the perspective of resource constraints.

In spite of the importance of the previous studies, it is still impossible for these attack planning methods to achieve penetration testing without prior knowledge. For instance, PDDL-based methods need an intact network structure and host configuration information, and POMDP-related methods need parts of that information. None of these methods can discover attack paths without prior knowledge. To achieve realistic penetration testing, there are two challenges: how to drive an agent to penetrate automatically and intelligently and how to choose the best attack action when faced with a specific situation. To deal with these challenges, we propose a network information gain based attack planning (NIG-AP) algorithm. First, network information gain is proposed to formalize the penetration testing. Then a reinforcement learning model is reconstructed to guide discovery of the attack paths based on a cumulative network information gain.

In this study, we propose the NIG-AP algorithm, in which there is no need for prior network structure, software configuration information, and domain knowledge to discover the attack paths. The algorithm can intelligently retrieve essential penetration information in the penetration testing.

2 Preliminaries

Automated attack planning is an interdisciplinary domain, involving cyber security and intelligent planning domains. In this section, we present some background on automated attack planning, covering mainly penetration testing and deep reinforcement learning.

2.1 Penetration testing

Penetration testing aims to obtain control over specific computers in a target network. Usually it starts with a controlled computer, where hackers try to gather computer and network information through scanning or exploit operations. After gathering information, hackers choose the best action to penetrate the target network, which we call an "exploit." Fig. 1 shows an example of penetration testing. The steps of penetration testing are as follows:

1. Gathering information

This is an essential step in penetration testing and usually takes a long time. The collected information contains opened ports, services, operating system version, and vulnerability information, obtained through scan tools such as Nmap and X-Scan. The information is the basis for choosing the proper exploitation action.

2. Exploitation

Given the essential penetration information gathered above, hackers choose the corresponding exploitation action to attack and control the target computer according to the intended penetration objective. A typical exploit action contains various vulnerability exploits, such as the structured query language (SQL) injection and binary vulnerability exploits.

3. Privilege escalation

This is a pivotal step to continue the subsequent attacks. The privilege that hackers achieved in the last step is usually fairly constrained, which can limit the hackers' choice of action. By adopting privilege escalation, the available action space can be enlarged and contribute to the hacker's subsequent attack choices.

4. Trace elimination

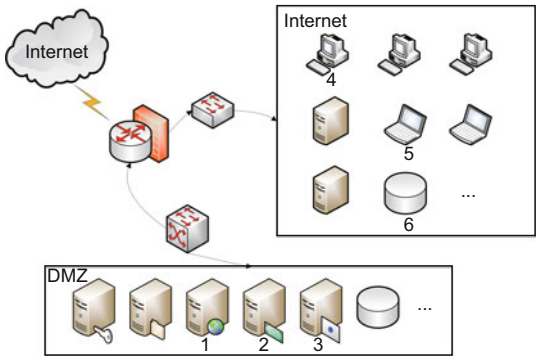

Fig. 1 Typical penetration testing scenario

After achieving control of the victim computer, attack traces (security or operation logs) left behind in the computer have to be eliminated, so that the hackers can safely log on to the computer again. Trace elimination can be done by deleting security logs and other log information files.

2.2 Deep reinforcement learning

As shown in Fig. 2a, reinforcement learning (Sutton and Barto, 1998) is a kind of trial-of-error algorithm. The agent or learner is not told what to do but takes action guided by numerical reward. Formally, reinforcement learning can be represented as (S, A, R, P, γ), where S represents the state space, A represents the action space, R represents a reward function, P represents the state transition probability function, and $\gamma \in (0, 1)$ is a discount factor. The objective of reinforcement learning is to optimize policy $\pi(a|s)$ so as to maximize the cumulative reward shown as

$$E(G_t|\pi) = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n}, \quad (1)$$

where R_t is instant reward attained at time t and G_t is cumulative reward retrieved from that time on to the end of the episode $t + n$. A Q-learning algorithm is often used to solve the above objective function (Szepesvári, 2010). In Q-learning, the cumulative reward during this process can be represented as

$$Q^{\pi}(s, a) = E[G_t|s_t = s, a_t = a, \pi]. \quad (2)$$

Within the policy space, there exist some policies π^* that satisfy $Q^*(s, a) \geq Q^{\pi}(s, a)$, where * is the optimal policy and $Q^*(s, a)$ is the optimal state action value function following the Bellman optimality equation shown as

$$Q^*(s, a) = E_{s' \sim S}[r + \gamma \max_{a'} Q(s', a')|s, a]. \quad (3)$$

According to the Banach fixed-point theorem (Mnih et al., 2013), $Q(s, a)$ will iterate to an

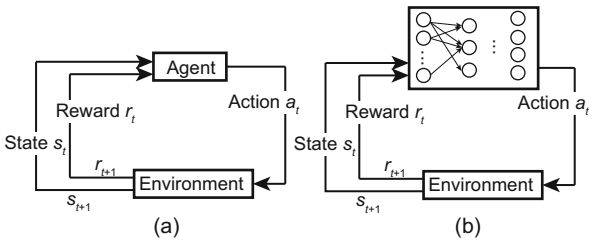

Fig. 2 Reinforcement learning (a) and deep reinforcement learning (b)

optimal value, so that we can infer the optimal policy according to the formula shown below after the policy iteration:

$$\pi^*(a|s) = \underset{a}{\operatorname{argmax}}\, Q(s,a). \tag{4}$$

However, with the increase of the sizes of S and A, the iterative process for solving the optimal policy becomes unfeasible. One natural way to deal with this problem is to adopt a nonlinear function approximator, such as an artificial neural network, to represent the state action value function. After incorporating a deep neural network into reinforcement learning (Fig. 2b), it becomes deep reinforcement learning (DRL) (Mnih et al., 2015). The Adam algorithm (Kingma and Ba, 2014) is usually adopted to train a deep neural network, whose update formula is shown as

$$\theta_t \leftarrow \theta_{t-1} - \alpha\hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon), \tag{5}$$

where $\hat{m}_t$ is a bias-corrected first moment estimate, $\hat{v}_t$ is a bias-corrected second raw moment estimate, α is the step size, ϵ is the correction parameter, and θ_t is the parameter of the deep neural network. The original $Q(s,a)$ table will be replaced with a neural network fitting function whose update formula is much more efficient.

3 Methodology

In this section, we will introduce the network information gain into attack planning and a network information gain based attack planning algorithm to plan attack paths in a target network without prior knowledge.

3.1 Network information gain

Penetration testing is a systematic engineering. Actions taken by intruders maximize the information entropy of a target network composed of two parts: host information and network information entropies. The host information entropy is made up of four parts: (1) Operating system (OS) information is used to describe the information about the operating system type, version, and language packages; (2) Application information is used to describe the information about the installed software; (3) Port information is used to describe the services that a target host has opened; (4) Protection mechanism information is used to describe the activated protection mechanism information of the target host.

The detailed host information can be formalized as vector $[\boldsymbol{P}_{\rm os}, \boldsymbol{P}_{\rm app}, \boldsymbol{P}_{\rm port}, \boldsymbol{P}_{\rm pro}]$, where $\boldsymbol{P}_{\rm os}$ illustrates the operating system probability distribution, where each element represents the probability of the corresponding operating system installed on the host. The other three vectors are not exclusive, which means that one specific computer can have multiple elements (for example, one host could have opened ports 80 and 22 and installed Firefox and IE at the same time). Thus, it is necessary to normalize these vectors.

Given a general vector, information entropy (Liang and Shi, 2004) is adopted to represent the exposure state of the victim computer, which is calculated as

$$\begin{aligned} H(\boldsymbol{P}) = &-\sum_{k=1}^{M}\sum_{j=1}^{|p_k|}(p_{kj}\log p_{kj}+(1-p_{kj})\log(1-p_{kj})) \\ &-\sum_{i=1}^{|\boldsymbol{P}_{\rm os}|} p_i\log(p_i), \end{aligned} \tag{6}$$

where $\boldsymbol{P}_{\rm os}$ is the operating system vector and M is the set of the other three vectors. From the formula above, it is stated that the information entropy is high at the very beginning as an intruder knows nothing about the target host, and it decreases as the intruder comes to know an increasing amount of information about the victim computer by scanning or exploiting actions. When the intruder takes over the victim computer, there is no uncertainty in the general vector, meaning that there is only 1 or 0 in the vector, and the information entropy H decreases to 0. Given the formula, it is easy to prove that no matter what action is taken, the information entropy after an action will be no greater than the information entropy before the action, because the penetration testing action can decrease uncertainty. We prove the idea in Theorem 1.

Theorem 1 Assume that the exposure state of the target computer can be represented as a general vector $\boldsymbol{P} = [\boldsymbol{P}_{\rm os}, \boldsymbol{P}_{\rm app}, \boldsymbol{P}_{\rm port}, \boldsymbol{P}_{\rm pro}]$. Let $\boldsymbol{P}_0$ ($\boldsymbol{P}_1$) be the general computer vector before (after) action a. $H(\boldsymbol{P}_0)$ ($H(\boldsymbol{P}_1)$) is the corresponding information entropy. Then the information gain $\Delta H = H(\boldsymbol{P}_0) - H(\boldsymbol{P}_1) \geq 0$ regarding action a is always true.

Because the exposure state of the target

computer $\boldsymbol{P}=[\boldsymbol{P}_{\rm os},\boldsymbol{P}_{\rm app},\boldsymbol{P}_{\rm port},\boldsymbol{P}_{\rm pro}]$ is composed of four parts and every two of them are independent, if we can prove that for each vector in $\boldsymbol{P}$ the above formula is satisfied, then it will be easy for us to prove the theorem. Let $\boldsymbol{P}_{\rm os}^0$ ($\boldsymbol{P}_{\rm os}^1$) be the operating system vector before (after) action a. For each element p in $\boldsymbol{P}_{\rm os}^0$ which satisfies $p\geq 0.5$ ($p\leq 0.5$), the corresponding element after action a in $\boldsymbol{P}_{\rm os}^1$ will satisfy $p'\geq p\geq 0.5$ ($p'\leq p\leq 0.5$). At the same time, because the network information gain $H(p)=\sum_{j=1}^{|\boldsymbol{p}|}(p_j\log p_j+(1-p_j)\log(1-p_j))$ is monotonous, then $H(\boldsymbol{p}_{\rm os})\geq H(\boldsymbol{p}'_{\rm os})$ holds. It is the same for the other three parts. As the information entropy is the sum of the four parts, the theorem holds true.

To guide the agent to automatically take the best action, we need to encourage those actions that can successfully gain more information from the victim host, and punish those actions that do not contribute to their further penetration. Given network information entropy, we adopt network information gain (Lee and Lee, 2006) as the signal for evaluating actions taken by the agent, whose formula is shown as

$$\Delta H=H(\boldsymbol{P}_{\rm before})-H(\boldsymbol{P}_{\rm after}),\tag{7}$$

where $H(\boldsymbol{P}_{\rm before})$ is network information entropy before action and $H(\boldsymbol{P}_{\rm after})$ is network information entropy after action. There will be three kinds of situations for calculating network information gain: (1) The uncertainty of the victim computer decreases but it is not eliminated after taking actions such as OS detection and port scan, so the information gain from the action is the difference between two probability distributions. (2) The victim computer is controlled after action, such as a vulnerability exploit; in this case, the information gain is the information entropy of the state before action. (3) The action has no influence on the state of the victim computer and the probability distribution is the same after action, so the information gain is 0.

$$\Delta H=\begin{cases}H(\boldsymbol{P}_{\rm before})-H(\boldsymbol{P}_{\rm after}), & \text{Situation (1)},\\ H(\boldsymbol{P}_{\rm before}), & \text{Situation (2)},\\ 0, & \text{Situation (3)}.\end{cases}\tag{8}$$

The reward r for an action of an agent will be composed of two parts $r_{\rm gain}$ and $r_{\rm cost}$, satisfying $r=r_{\rm gain}+r_{\rm cost}$. $r_{\rm gain}$ is information gain, which conveys information to judge an action. Compared with the original constant reward, $r_{\rm gain}$ is much more flexible and will guide our agent to choose a better action so as to achieve much more cumulative reward. $r_{\rm cost}$ is action cost. There are two reasons for setting this term: one is to limit the number of actions to avoid an endless loop, and the other is to guide the agent to find the attack paths that are as good as possible. $r_{\rm cost}$ is calculated based on the common vulnerability scoring system (CVSS) and the formula is shown as

$$r_{\rm cost}=\text{sigmoid}(\text{Score}_{\rm cvss}\cdot\text{Score}_{\rm period}),\tag{9}$$

$$\text{Score}_{\rm period}=\begin{cases}70, & t\leq 60,\\ 90, & 60<t\leq 730,\\ 50, & 730<t\leq 1825,\\ 30, & t>1825,\end{cases}\tag{10}$$

where $\text{Score}_{\rm cvss}$ is the CVSS value, $\text{Score}_{\rm period}$ represents action timeliness, and t is the disclosure time of vulnerability.

Network information entropy concerns mainly connectivity among computers and a scan operation is usually adopted to test computer connectivity. Thus, we adopt a scan operation whenever there are no useful actions for an agent to take, and the total information gain will increase when there are available computers that an agent can penetrate.

3.2 Network information gain based automated attack planning algorithm

Penetration testing can be seen as a typical Markov decision process (MDP) whose trajectory is shown as follows:

$$\begin{aligned}&\{S_0,A_0,R_1,S_1,A_1,R_2,\ldots,\\ &\ S_i,A_i,R_{i+1},\ldots,S_n,A_n,R_{n+1}\},\end{aligned}\tag{11}$$

where S_i represents the victim state, A_i represents the action taken by the agent given state S_i, and R_{i+1} represents the reward for action A_i under state S_i. n denotes the end of the episode. Penetration testing aims at compromising target hosts within a limited period of time, which can be formalized as Eq. (1), where R_t is the same as ΔH, representing the network information gain at time t, and γ is a discount factor within $(0,1)$. S represents the combination of the operating system, opened services, ports, protection mechanisms, and whether the computer has been compromised, for example, M0-win2000-p445-SMB-compromised and

M1-win2003-p80-HTTP-uncompromised. Because different policies lead to different cumulative rewards, the goal of MDP is to find the best policy π that maximizes cumulative rewards G_t, which is shown as

$$\max\ E[G_t|\pi]. \tag{12}$$

Given the objective function, a Q-learning algorithm is adopted to find the best policy. The policy is a mapping function which maps a state to an action, and the mapping function can be represented as a tabular form, where each row stands for a host state and each column stands for an action. The value stands for the contribution value of adopting a corresponding action under a specific state to reach the final goal. According to the Bellman equation, the update formula of $Q(s,a)$ for one specific computer is shown as

$$Q^*(s,a)=E_{s'\sim S}[\Delta H-r_{\text{cost}}+\gamma \max_{a'} Q(s',a')|s,a]. \tag{13}$$

To update the parameters of a deep neural network to fit $Q(s,a)$, the Adam algorithm is adopted, and the update formula is shown as

$$\theta_t \leftarrow \theta_{t-1} - \alpha\hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon), \tag{14}$$

where $\hat{m}_t$ and $\hat{v}_t$ are calculated as

$$\hat{m}_t \leftarrow (\beta_1 \cdot m_{t-1}+(1-\beta_1)\nabla_\theta Q_t(\theta_{t-1}))/(1-\beta_1^t), \tag{15}$$

$$\hat{v}_t \leftarrow (\beta_1 \cdot v_{t-1}+(1-\beta_2)\nabla_\theta Q_t(\theta_{t-1})^2)/(1-\beta_2^t), \tag{16}$$

where α is the step size, ϵ is the correction parameter, and β_1 and β_2 are the exponential decay rates for moment estimates.

The method mentioned above aims at discovering an effective exploitation action in a specific host, and it cannot be generalized to a network scenario. To extend a specific host to a network scenario, an observed computer set Φ is created to save the detected computers. When Φ is empty or actions chosen by an agent have no influence on the information gain, there will be a scan action to discover new available computers. When there are multiple actions that will influence the information gain, we choose the action that contributes the most to the cumulative reward according to policy π. Calculating the state transition probability is hard in a manual setting, so the Monte Carlo method is used to estimate the state transition probability during the training stage. At the very beginning of the training stage, a large number of random action sequences are chosen to retrieve the available knowledge hidden in penetration testing. The discount factor γ is set to 0.9 to place a greater weight on the long-term reward. The algorithm for automated attack planning is summarized in Algorithm 1.

Algorithm 1 Network information gain based automated attack planning

Require: $S, A, \gamma, \alpha, \Phi, Q(\theta), m_0, v_0, \beta, \epsilon$
Ensure: π^*
Initialize θ, m_0, v_0 arbitrarily
If $\neg\Phi$ and action a contributes to H
  Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
      Choose action a from s using current policy derived from $Q(s,a)$
      Take action a, observe s', and calculate reward $r = H(s) - H(s')$
      $m_t \leftarrow \beta_1 \dot{m}_{t-1} + (1-\beta_1)\dot{\nabla}_\theta Q_{t-1}(\theta)$
      $v_t \leftarrow \beta_2 \dot{v}_{t-1} + (1-\beta_2)\dot{\nabla}_\theta Q_{t-1}(\theta)^2$
      $\hat{m}_t \leftarrow m_t/(1-\beta_t)$
      $\hat{v}_t \leftarrow v_t/(1-\beta_t)$
      $\theta_t \leftarrow \theta_{t-1} - \alpha\hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon)$
      $Q_t(s,a) \leftarrow Q_{t-1}(s,a) + \alpha(r - Q_{t-1}(s,a) + \gamma \max_{a'} Q_{t-1}(s',a'))$
      $s \leftarrow s'$
    Until s is end
Else
  Take scan action and update Φ, and continue update $Q(s,a)$
End if

4 Experiment

The effectiveness of NIG-AP is tested on a machine running an Intel Core i7 CPU at 2.5 GHz with 16 GB RAM. The experiment network scenario is constructed based on full virtualization technology according to typical enterprise networks operating in the real world, where the operating systems and applications run in the same way as on physical machines.

To collect the required scenario data, Nessus (Beale et al., 2004) is used to scan each subnet of the enterprise network to gather information about the host Internet protocol, opened services, operating system, and vulnerability information. Even though the various vulnerabilities detected by Nessus

can be falsely detected or are unimportant for further penetration, it is still reproduced in the experimental scenario. There are two reasons for doing so: the first is that reproducing a network scenario vividly is important for verifying the effectiveness of the attack planning algorithm, and the second is that avoiding useless vulnerability exploitation is a metric in evaluating an automated attack planning algorithm. All of the scanned hosts will be reproduced in the experimental scenario, and there are also some random hosts assigned to the demilitarized zone (DMZ) area and intranet area to evaluate the scalability of attack planning algorithms. We compare our proposed algorithm with POMDP and the forward search planner, and the goal is to recognize effective attack paths.

4.1 Network scenario

As shown in Fig. 3, the experiment network structure is composed of two parts, the DMZ area and an intranet area, and connected by a firewall and a router. The firewall rules allow the internet host to visit the DMZ host and the intranet host to visit the DMZ host, but the rules do not allow the internet host to visit the intranet host.

Given the network structure shown above, the basic idea of penetration testing consists of three steps: (1) The hacker breaks into the DMZ area via SQL injection, obtaining sensitive information about the router password; (2) The hacker controls the router given the sensitive information attained in the previous step; (3) The hacker discovers the vulnerabilities of the intranet hosts, and exploits them to take control of the target computer.

4.2 Action space

Actions are retrieved from the open source penetration test framework "Metasploit" (Holik et al., 2014), which includes mainly two types of action, information gathering and vulnerability exploit. First, log into Metasploit remotely using the manager interface. Second, read the whole database and extract the essential information. Third, construct the uniform resource locator (URL) based on the retrieved information to crawl the customer premise equipment (CPE) information. Finally, store that information in the database. The information gathering actions contain mainly port scan and OS scan operations, while the vulnerability exploit actions comprise concrete vulnerability exploitations such as

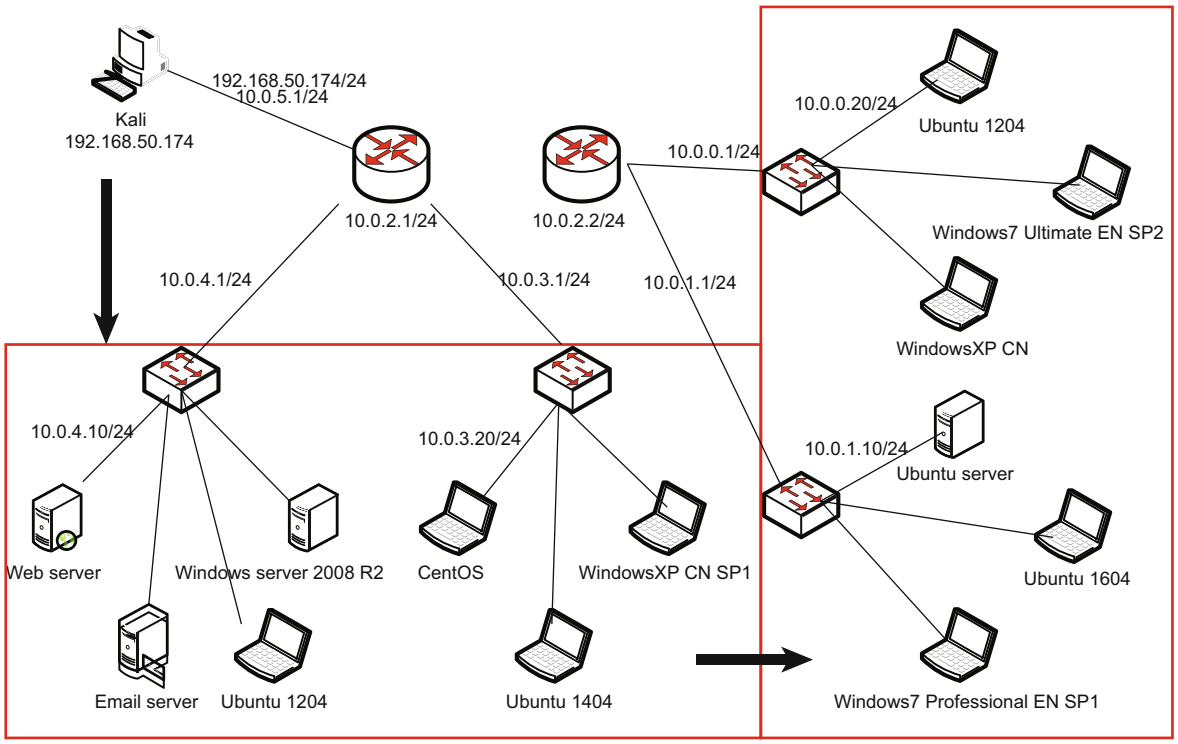

Fig. 3 Experimental scenario

an SQL injection vulnerability exploitation against Joomla. As shown in Fig. 4, each action is composed of three parts: parameters, predicates, and effects, where the parameters represent the variables used in the PDDL description, predicates represent the preconditions of executing an action, and effects represent the outcomes after executing this action. There are in total 1798 actions extracted from Metasploit by using "msfrpcd."

4.3 Experimental results

POMDP is implemented based on the APPL toolkit (Kurniawati et al., 2008), forward search is implemented based on fast forward (FF) (Baulcombe, 1999), and NIG-AP is implemented based on mdptoolbox (Chadès et al., 2014).

The experimental results are separated into two parts: the first compares POMDP with NIG-AP in terms of convergence, and the second compares POMDP with the forward search planner and NIG-AP in terms of effectiveness. NIG-AP is used in the experiments with various optimization algorithms, including the Q-learning optimization algorithm (NIG-AP(Q)), the relative value iteration optimization algorithm (NIG-AP(RVI)), the value iteration algorithm (NIG-AP(VI)), the policy iteration algorithm (NIG-AP(PI)), the modified policy iteration algorithm (NIG-AP(PIM)), and the value iteration Gerchberg-Saxton (GS) algorithm (NIG-AP(VIGS)).

As shown in Fig. 5, it is easy to find that our NIG-AP planner has a good performance in convergence. Though POMDP performs better at the beginning when compared with NIG-AP, they share a similar performance after the 3rd iteration, which means that they possess similar attack path planning precision. The error threshold is reached after four iterations, which means NIG-AP performs as well as POMDP in planning attack paths.

Apart from precision performance, it can be seen from Fig. 6 that NIG-AP(VI), NIG-AP(RVI), and NIG-AP(PIM) share similar training time along with the increasing size of the host number, and these algorithms perform better than NIG-AP(Q) and NIG-AP(VIGS). Nevertheless, the training time is still acceptable, and for 14 states, the training time is limited to within 0.5 s, and NIG-AP(RVI), NIG-AP(VI), NIG-AP(PI), and NIG-AP(PIM) are limited to within 0.02 s.

As for POMDP, we illustrate the time consumption performance in Table 1, from which we can see that NIG-AP consumes less planning time, while POMDP consumes much time in planning attack paths. When the host number reaches three, the APPL toolkit cannot solve the problem. This illustrates that the effectiveness of POMDP is very low, and it is hard for POMDP to be generalized to large network scenarios; however, NIG-AP can effectively find attack paths in large network scenarios.

To compare the effectiveness of the planners, the

```
Information gathering action

(: action
Os_scan
: parameters (?srcIP?dstIP)
: precondition ( and
(connectivity ? srcIP ? dstIP)
(compromised ? srcIP))
: effect (and (osType > dstIP)(increase (total – cost)5)))
```

```
Vulnerability exploitation action

(: action
Joomla_Content_History_SQLi_Remote_Code_Execution
: parameters (?srcIP?dstIP)
: precondition ( and
(connectivity ? srcIP ? dstIP)
(compromised ? srcIP))
(isunix ? dstIP)
(or (isjoomla213 ? dstIP))
:effect(and(compromised ? dstIP)(increase(total – cost)75)))
```

Fig. 4 Action example

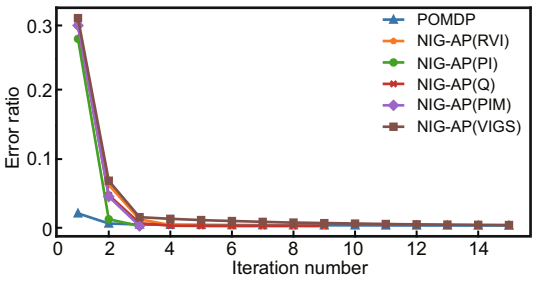

Fig. 5 Iteration performance

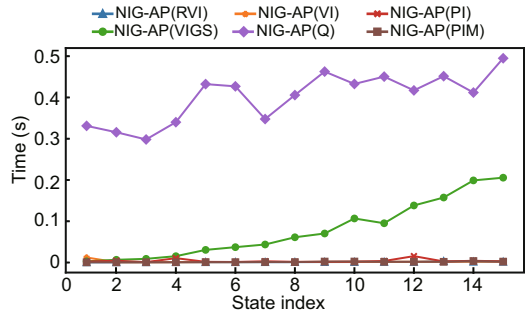

Fig. 6 Time performance

effectiveness metric is defined as

$$e = \frac{N_{\text{correct}}}{N_{\text{state}}}, \tag{17}$$

where N_{state} is the host state number while N_{correct} is the number of correct actions considering the specific state shown in the output policy file. All of the gathered vulnerability information is reproduced in the scenario, but only some contribute to further penetration according to artificial analysis. An effective attack planning algorithm can recognize those useless vulnerabilities, find valid attack paths as soon as possible, and generate less network traffic. We record an effectiveness metric on a large complex network scenario, after restricting the number of available vulnerability exploits to 10. The experimental results are shown in Table 2, from which we can see that NIG-AP outperforms POMDP and FF in terms of effectiveness. The lowest effectiveness with NIG-AP is about 0.36, which is far beyond that of FF, whose effectiveness ratio is about 0.1.

Fig. 7 shows the effectiveness ratio of different algorithms. The effectiveness of NIG-AP outperforms that of FF and POMDP along with the increasing size of the state number. FF is a deterministic planner that iterates all possible attack paths in order, so the effectiveness ratio of FF is constant, and the effectiveness of FF is completely determined by the order of exploitation actions in the database. POMDP shows great performance at the very beginning, but it is hard for POMDP to solve a large state scenario, so there is no effectiveness ratio after the 2nd state because of the solution complexity problem. Compared with POMDP, NIG-AP converges faster, which makes NIG-AP more generalized in large complex network scenarios. FF plans attack paths by combining all possible actions that satisfy the preconditions, and the solution space is very large. POMDP and NIG-AP reduce the space using

Table 1 Training time of different planners for four state numbers

Planner	Training time (s)			
	1	2	3	4
NIG-AP(VI)	0.0121	0.0007	0.0006	0.0007
NIG-AP(RVI)	0.0004	0.0005	0.0006	0.0008
NIG-AP(Q)	0.3312	0.3157	0.2982	0.3401
NIG-AP(PI)	0.0025	0.0044	0.001	0.0106
NIG-AP(PIM)	0.0005	0.0006	0.0007	0.0007
NIG-AP(VIGS)	0.0028	0.0066	0.0089	0.0153
POMDP	5.62	27.43	–	–

NIG-AP: network information gain based automated attack planning; NIG-AP(VI): NIG-AP with value iteration; NIG-AP(RVI): NIG-AP with relative value iteration; NIG-AP(Q): NIG-AP with Q-learning; NIG-AP(PI): NIG-AP with policy iteration; NIG-AP(PIM): NIG-AP with modified policy iteration; NIG-AP(VIGS): NIG-AP with value iteration Gerchberg-Saxton; POMDP: partially observable Markov decision process

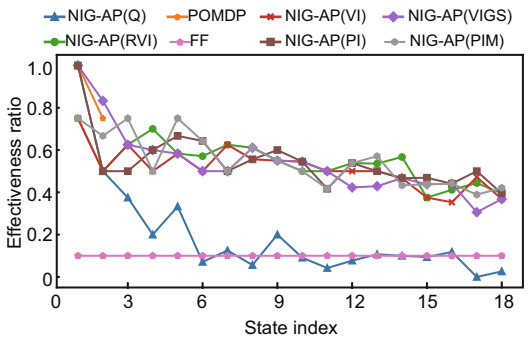

Fig. 7 Performance of discovering correct action

Table 2 Attack action discovery effectiveness of different planners

Planner	Attack action discovery effectiveness													
	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NIG-AP(Q)	1	0.5	0.375	0.2	0.333	0.071	0.125	0.056	0.2	0.091	0.042	0.077	0.107	0.1
NIG-AP(VI)	0.75	0.5	0.625	0.5	0.583	0.5	0.625	0.556	0.55	0.545	0.5	0.5	0.5	0.467
NIG-AP(PI)	1	0.5	0.5	0.6	0.667	0.643	0.5	0.556	0.6	0.545	0.417	0.538	0.5	0.467
NIG-AP(VIGS)	1	0.833	0.625	0.6	0.583	0.5	0.5	0.611	0.55	0.545	0.5	0.423	0.429	0.467
NIG-AP(PIM)	0.75	0.667	0.75	0.5	0.75	0.643	0.5	0.611	0.55	0.5	0.417	0.538	0.571	0.433
NIG-AP(RVI)	0.75	0.5	0.625	0.7	0.583	0.571	0.625	0.611	0.55	0.5	0.5	0.538	0.536	0.567
POMDP	1	0.75	–	–	–	–	–	–	–	–	–	–	–	–
FF	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

NIG-AP: network information gain based automated attack planning; NIG-AP(VI): NIG-AP with value iteration; NIG-AP(RVI): NIG-AP with relative value iteration; NIG-AP(Q): NIG-AP with Q-learning; NIG-AP(PI): NIG-AP with policy iteration; NIG-AP(PIM): NIG-AP with modified policy iteration; NIG-AP(VIGS): NIG-AP with value iteration Gerchberg-Saxton; POMDP: partially observable Markov decision process; FF: fast forward

uncertainty information from the training data. As a classical planner, FF can find only one path because it cannot deal with uncertainty from a network scenario. To iterate all possible attack paths, FF changes the reward according to feedback from the scenario. We call it path iteration in the classical planning domain, and it is clear that FF is not suitable for real penetration testing scenarios, but POMDP and NIG-AP can find the greatest number of possible attack paths based on the uncertainty information. However, POMDP can be applied only to small network scenarios. Moreover, Fig. 7 shows that not all optimization algorithms contribute to NIG-AP. NIG-AP(Q) is not as good as the other NIG-AP algorithms, and the effectiveness ratio decreases to 0 when there are 16 states. However, our proposed NIG-AP(VI), NIG-AP(RVI), NIG-AP(PI), NIG-AP(PIM), and NIG-AP(VIGS) still show good performance, and it is more applicable to realistic penetration testing scenarios.

Furthermore, we explore the relationship between the total penetration testing time (including payload search time, penetration time, and shell establishment time) and the number of subnetworks. The host number within each subnetwork is restricted to two, and only the policy iteration optimization algorithm is applied to NIG-AP. We gradually increase the number of subnetworks to record the total penetration testing time. Because POMDP is not suitable for realistic penetration testing, only FF and NIG-AP are compared. The results are shown in Fig. 8, from which we can see that the total time exponentially increases with respect to the number of subnetworks. When the number of subnetworks is less than eight, the total time is still acceptable; otherwise, the total time increases so rapidly that the penetration task cannot be completed in a given time. It is the path-combinational explosion that results in such a phenomenon. Supposing that the number of subnetworks is $|A|$ and the available penetration action is limited to $|O|$, the complexity is $|O|^{|A|}|A|!$, which explains the exponentially increasing phenomenon. However, our NIG-AP algorithm can effectively eliminate invalid attack paths according to the training action sequences, so the total penetration time is less than that of the method in which all possible attack paths are iterated.

Network traffic plays an important role in evaluating attack plans, such that the less attack traffic

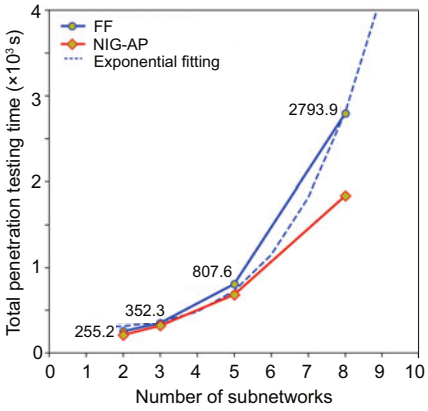

Fig. 8 Relationship between the total penetration testing time and the number of subnetworks

there is, the more efficient the plan will be. We collect attack traffic in the router through a TcpDump and visualize the traffic based on Wireshark. The traffic statistics for FF and NIG-AP are shown in Fig. 9, from which we can see that FF generates huge network traffic during the whole penetration process, while there is less traffic in our proposed planning algorithm. Furthermore, the traffic fluctuates much more with FF when compared with NIG-AP. The average packet speed generated by FF is 260 packets/s, but 190 packets/s by our proposed algorithm. Thus, not only the packet speed, but also the number of error packets generated by our algorithm are smaller compared with FF. The error packet means that the agent fills the packets with the wrong parameters. This demonstrates that our algorithm can choose the proper action to conduct effective penetration testing compared with FF.

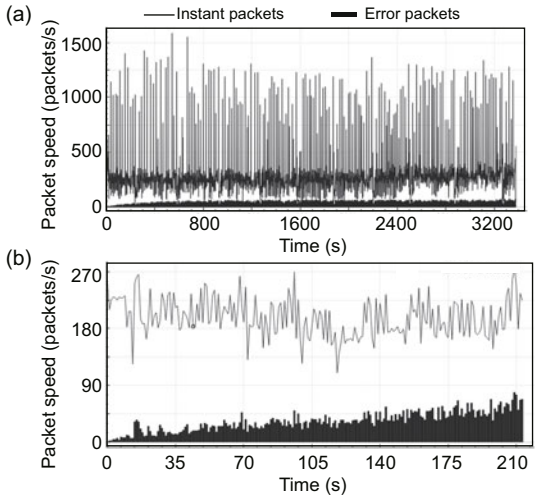

Fig. 9 Network traffic comparison: (a) FF; (b) NIG-AP

5 Conclusions and future work

In this study, we proposed a network information gain based automated attack planning (NIGAP) algorithm to automatically conduct penetration. First, we analyzed the state-of-the-art attack planning algorithms, including the attack tree, attack graph, classical planning, and POMDP. Second, we devised an attack planning algorithm based on reinforcement learning to automatically discover attack paths without prior knowledge of the scenario network. Finally, a comparison experiment was conducted and the results demonstrated that our proposed algorithm can automatically discover attack paths without prior information. We hope that our research can drive the industrial applications and inspire other researchers as well.

A further step in this study would be incorporating temporal information about the vulnerabilities into the planning algorithm. The timeliness of a vulnerability influences much of its application. For example, 0-day vulnerability is much more useful in penetration tests than those vulnerabilities discovered some time ago. When training the agent, the update algorithm should pay much more attention to those effective and aging vulnerabilities. One possible method of dealing with this problem is to set the reward function relating to time, so that the training algorithm is predisposed to recent training examples. Another method is to rank the vulnerabilities according to a time-stamp, so that new exploits would always be considered first.

The other research direction we are currently pursuing is using transfer learning to generate the attack scheme. Because there are various kinds of network scenarios, we cannot iterate every scenario to train our agent. One possible way to solve this problem is adopting transfer learning in implementing knowledge transfer to apply our agent to unknown network scenarios. Since penetration testing knowledge is similar in various scenarios, transfer learning shows us the way to train an agent with fewer scenario examples, and remains effective.

Compliance with ethics guidelines

Tian-yang ZHOU, Yi-chao ZANG, Jun-hu ZHU, and Qing-xian WANG declare that they have no conflict of interest.

References

Alexander Pretschner AS, 2017. Automated Attack Planning Using a Partially Observable Model for Penetration Testing of Industrial Control Systems. MS Thesis, Technische Universität München, München, Germany.

Backes M, Hoffmann J, Künnemann R, et al., 2017. Simulated penetration testing and mitigation analysis. https://arxiv.org/abs/1705.05088v1

Baulcombe DC, 1999. Fast forward genetics based on virus-induced gene silencing. *Curr Opin Plant Biol*, 2(2):109-113. https://doi.org/10.1016/S1369-5266(99)80022-3

Beale J, Meer H, van der Walt C, et al., 2004. Nessus Network Auditing: Jay Beale Open Source Security Series. Elsevier, Amsterdam, the Netherlands.

Chadès I, Chapron G, Cros MJ, et al., 2014. MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. *Ecography*, 37(9):916-920. https://doi.org/10.1111/ecog.00888

Core Security, 2019. Core Impact Penetration System. https://www.secureauth.com/products/penetration-testing/core-impact [Accessed on Feb. 23, 2019].

Fox M, Long D, 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J Artif Intell Res*, 20:61-124. https://doi.org/10.1613/jair.1129

Futoransky A, Notarfrancesco L, Richarte G, et al., 2010. Building computer network attacks. https://arxiv.org/abs/1006.1916

Holik F, Horalek J, Marik O, et al., 2014. Effective penetration testing with metasploit framework and methodologies. IEEE 15th Int Symp on Computational Intelligence and Informatics, p.237-242. https://doi.org/10.1109/CINTI.2014.7028682

Khan S, Parkinson S, 2017. Towards automated vulnerability assessment. 27th Int Conf on Automated Planning and Scheduling, p.33-40.

Kingma DP, Ba J, 2014. Adam: a method for stochastic optimization. https://arxiv.org/abs/1412.6980

Kurniawati H, Hsu D, Lee WS, 2008. SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Brock O, Trinkle J, Ramos F (Eds.), Robotics: Science and Systems IV. MIT Press, Massachusetts, USA, Chapter 10.

Lee C, Lee GG, 2006. Information gain and divergence-based feature selection for machine learning-based text categorization. *Inform Process Manag*, 42(1):155-165. https://doi.org/10.1016/j.ipm.2004.08.006

Liang JY, Shi ZZ, 2004. The information entropy, rough entropy and knowledge granulation in rough set theory. *Int J Uncert Fuzzy Knowl Syst*, 12(1):37-46. https://doi.org/10.1142/S0218488504002631

Mnih V, Kavukcuoglu K, Silver D, et al., 2013. Playing Atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602

Mnih V, Kavukcuoglu K, Silver D, et al., 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529-533. https://doi.org/10.1038/nature14236

Obes JL, Sarraute C, Richarte G, 2013. Attack planning in the real world. https://arxiv.org/abs/1306.4044

Roberts M, Howe A, Ray I, et al., 2011. Personalized vulnerability analysis through automated planning. Proc Int Joint Conf on Artificial Intelligence, p.50-57.

Samant N, 2011. Automated Penetration Testing. MS Thesis, San Jose State University, California, USA.

Sarraute C, Richarte G, Lucángeli Obes J, 2011. An algorithm to find optimal attack paths in nondeterministic scenarios. 4th ACM Workshop on Security and Artificial Intelligence, p.71-80. https://doi.org/10.1145/2046684.2046695

Sarraute C, Buffet O, Hoffmann J, 2012. POMDPs make better hackers: accounting for uncertainty in penetration testing. 26th AAAI Conf on Artificial Intelligence, p.1816-1824 .

Sarraute C, Buffet O, Hoffmann J, 2013. Penetration testing == POMDP solving? https://arxiv.org/abs/1306.4714

Schneier B, 1999. Attack trees. *Dr Dobb's J*, 24(12):21-29.

Sheyner O, Haines J, Jha S, et al., 2002. Automated generation and analysis of attack graphs. IEEE Symp on Security and Privacy, p.273-284. https://doi.org/10.1109/SECPRI.2002.1004377

Shmaryahu D, Shani G, Hoffmann J, et al., 2017. Partially observable contingent planning for penetration testing. 1st Int Workshop on Artificial Intelligence in Security, p.33-40.

Stefinko Y, Piskuzub A, 2017. Theory of modern penetration testing expert system. *Inform Process Syst*, 148(2):129-133. https://doi.org/10.30748/soi.2017.148.25

Steinmetz M, 2016. Critical constrained planning and an application to network penetration testing. 26th Int Conf on Automated Planning and Scheduling, p.141-144.

Sutton RS, Barto AG, 1998. Reinforcement Learning: an Introduction. MIT Press, Cambridge, London.

Szepesvári C, 2010. Algorithms for Reinforcement Learning. Morgan & Claypool Publishers, San Rafael, Argentina.

Zhuang YT, Wu F, Chen C, et al., 2017. Challenges and opportunities: from big data to knowledge in AI 2.0. *Front Inform Technol Electron Eng*, 18(1):3-14. https://doi.org/10.1631/FITEE.1601883