Received April 12, 2019, accepted May 2, 2019, date of publication May 7, 2019, date of current version May 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2915091

An Improved Attack Path Discovery Algorithm Through Compact Graph Planning

ZANG YICHAO[1], ZHOU TIANYANG[1,2], GE XIAOYUE[1], AND WANG QINGXIAN[1,2]
[1]Information and Engineering University, State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
[2]National Engineering Technology Research Center of the National Digital Switching System, Zhengzhou 450001, China

Corresponding author: Zhou Tianyang (aipteamzhouty@aliyun.com)

This work was supported by the National Natural Science Foundation of China under Grant 61502528.

ABSTRACT Attack path discovery plays an important role in protecting network infrastructure. Unfortunately, previous attack path discovery algorithms are difficult for applying in reality because of the high computational complexity problem. To achieve effective attack path discovery, we proposed a compact graph planning algorithm to incorporate goal states related information into attack path discovery. Our model extracts goal states related information by calculating closure of goal states, and then construct planning graph structure given the closure, after which the backward search algorithm is used to extract the attack path solution. The experiments were done on the typical enterprise network, comparing the effectiveness of attack path discovery algorithms with existing known methods. The result turns out that our proposed compact graph planning algorithm shows great improvement in discovering attack paths.

INDEX TERMS Attack path discovery, graph planning, functional dependency theory, cyber security.

I. INTRODUCTION

With the increasing size and complexity of computer network, cyber security problem becomes more prominent than ever. Community Emergency Response Team(CERT) points out that global network security events increase exponentially from 2003 to 2018 [1]. To cope with the problem, attack paths discovery technology [2], which could discover hidden network vulnerabilities automatically, is widely adopted. As shown in Figure 1, attack paths discovery technology could not only find individual vulnerability existed in each host, but combinational vulnerabilities existed in whole network. Although tools like APT2 [3], Autosploit [4] and MulVAL [5] *etc*. have been developed to improve efficiency for discovering hidden attack paths, they are still far from reality because of two limitations. The first is that most of these tools, such as APT2, Autosploit and so on, aim at discovering individual vulnerabilities and fail in finding combinational vulnerabilities, and the second is that high computational complexity problem makes it hard to apply in large scale network.

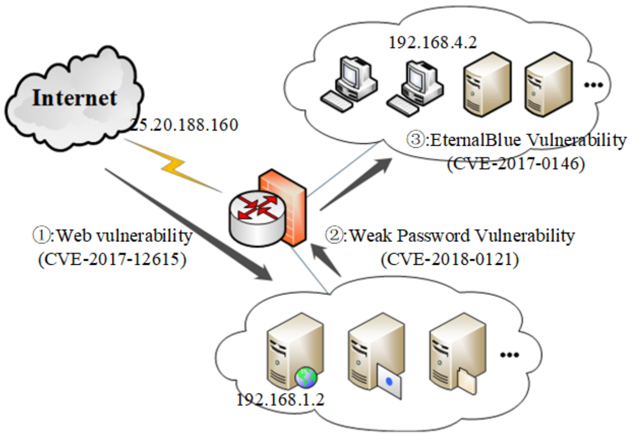
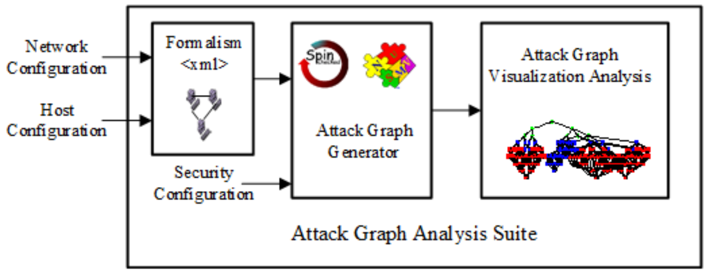


FIGURE 1. **Attack paths discovery in typical enterprise computer network (*Attack paths is the combination of individual vulnerability within each host:①②③*).**

To overcome these problem and improve efficiency of attack paths discovery, we propose a compact graph planning based attack paths discovery algorithm which could utilize penetration testing goal related information to prune redundant attack path branches based on a variant structure called ‘‘compact planning graph’’[6]. The rest of this paper is

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.



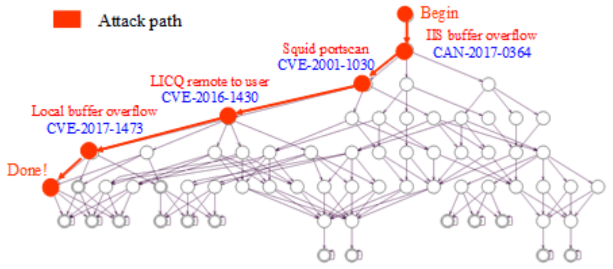

Attack path
Begin
IIS buffer overflow CAN-2017-0364
Squid portscan CVE-2001-1030
LICQ remote to user CVE-2016-1430
Local buffer overflow CVE-2017-1473
Done!

FIGURE 2. Attack path discovery through attack graph technology (*The first subgraph shows process of attack graph generation and the second subgraph describes attack paths vulnerability analysis* [7].).

organized as follows: Section 2 introduces some related works about attack paths discovery technology, mainly containing attack graph based methods and classical planning based methods. Section 3 describes some preliminary knowledge about functional dependency theory which is the foundation of constructing compact planning graph. Section 4 shows the details of our proposed compact graph planning based attack paths discovery algorithm, containing closure calculation of goal states set, compact planning graph construction and attack paths extraction. Section 5 compares our proposed algorithm with existing methods and gives some explanations about the results. Section 6 concludes.

II. RELATED WORKS

A rich body of works have explored various ways to achieve effective attack paths discovery, which could be divided into two categories: attack graph based methods and classical planning based methods.

Attack graph model, shorted in AG, was firstly introduced by Sheyner et al. [7] to discover hidden network vulnerabilities. Generally, attack graph could be represented as tuple <*H, C, T, I, IDS*>, where *H* represents host information, containing ip, services, softwares and vulnerabilities, *C* represents connectivity information, *T* represents trust information among hosts, *I* represents initial network information and *IDS* describes attack behaviors that could be detected. As shown in Figure 2, there are two steps to discover attack paths through attack graph model, namely attack graph generation and vulnerability analysis. Based on attack graph, Swiler *et al.* adopted forward search strategy to discover hidden attack paths [8]. As forward search strategy would cause state space explosion problem, Zhang *et al.* shrinked state space to accelerate attack path discovery through backward search strategy on the basis of attack graph [9]. Singhal *et al.* combined attack graph model and probability analysis, coming up with probabilistic model to evaluate security risk of enterprise network [10]. Rather than state attack graph, Ou *et al.* proposed logical attack graph that could find dependency relation between vulnerabilities to avoid state explosion problem [11]. Though logical attack graph shows great advantage in shrinking state space, the result is hard to be interpreted. Sun *et al.* proposed a probabilistic approach and implemented a prototype system

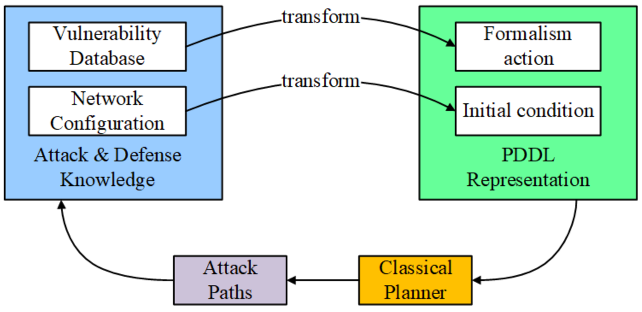


FIGURE 3. Classical planning based attack paths discovery (*Transform attack & defense knowledge into PDDL, and then use classical planner to discover attack paths based on transformed PDDL.*).

ZePro for zero-day attack paths identification [12]. Pan *et al.* came up with a systematic and automated approach to build a hybrid IDS to discovery attack paths for power systems [13]. Though this method improved efficiency a lot, they relied on experts' experience heavily, making it hard to apply in large scale network. Ding *et al.* presented a method which is based on the combination of finite state machine(FSM) and model checking technology(NuSMV) to analyse system safety [14]. To discover optimal attack path, Zhao *et al.* toke advantage of ant colony algorithm to optimize global security cost [15]. Wang *et al.* adopted heuristic search algorithm to generate attack graph and divided it into two stages: matching index table construction and attack graph construction [16]. Song *et al.* pruned branches of attack graph to discover attack path through greedy search strategy [17].

Aside from attack graph based attack paths discovery methods, classical planning provides another perspective for discovering attack paths. Usually, classical planning algorithms aim at discovering paths under deterministic, static, finite, fully observable, discrete circumstance [18]. The application of classical planning algorithms in cyber security domain firstly appeared in 2008 international planning competition(IPC), from which on, classical planning based attack path discovery became mainstream research direction [19]. As shown in Figure 3, there are two steps to implement attack paths discovery based on classical planning technology, first of which transformed the relevant network information, such as network topology, host configuration, vulnerability database *et al.* into planning domain definition language(PDDL) and the second toke advantage of classical planning algorithms, such as partial planning, hierarchical task network *etc*. to achieve attack paths discovery based on transformed PDDL [20]. The famous attack path discovery product that was developed based on classical planning is Core Impact automated penetration testing systems developed by SecureAuth corporation [21]. Boddy *et al.* demonstrated that the generation of attack paths for a simple but realistic web-based document control system with classical planning algorithms outperforms the prevailing state of the art in this area [22]. Roberts *et al.* extend the attack graph formalism by integrating user actions and supporting personalization that restrict the focus to those vulnerabilities present in a particular user/system combination. The enhanced attack graph was captured in PDDL with user specific attributes being included as facts, and it helped identify interventions that are most likely to reduce the system's susceptibility to attacks [23].

In spite of importance of previous studies, there are still some limitations in existing attack paths discovery methods. For instance, the attack graph based method shows simple way to discover attack paths, but the computational complexity limits its application. Classical planning based attack paths discovery avoids this by using well studied planning algorithms and achieves better result, but it does not take deep consideration into penetration testing goal states related information, which influences efficiency(time consumption *etc.*) of discovering attack paths greatly. In order to achieve effective attack paths discovery on the basis of existing studies, there are three challenges: How to extract essential information according to goal states? How to model the process of attack paths discovery? How to incorporate those information into model so as to achieve effective attack paths discovery? To conquer these challenges, we proposed a compact graph planning based attack paths discovery algorithm. Firstly, a closure calculation algorithm is proposed to extract all related states according to penetration testing goals. Then a variant compact planning graph is proposed to model process of attack paths discovery. Lastly, the extracted information is incorporated into compact planning graph to prune irrelevant states so as to achieve effective attack paths discovery.

III. PRELIMINARY

A. FUNCTIONAL DEPENDENCY THEORY

Functional dependency theory [24] originates from relational database theory, and it's a kind of constraint relation between sets of attributes from a database. In other words, functional dependency is a constraint that describes the relationship between attributes in a relation, which could help find all dependency attributes in a database. The detailed formalism of functional dependency theory is shown as follows:

Definition 1 (Relation pattern): Relation pattern $R(U)$ is attributes tuple, where each $u \in U$ is specific attribute. The quantization of relation R is denoted as r, and $r[u]$ represents attribute value.

Definition 2 (Functional Dependency): A functional dependency(FD) $\alpha \rightarrow \beta$ holds if for all pairs of tuples t, u in any legal instance satisfy the following equation:

$$if \;\; t[\alpha] = u[\alpha] \;\; then \;\; t[\beta] = u[\beta] \tag{1}$$

where α and β denote sets of attributes separately. $\alpha \rightarrow \beta$ is called attributes set α determined attributes set β or β is determined by α.

Definition 3 (Logically Implied): A function dependency $\alpha \rightarrow \beta$ is logically implied if all instances satisfying FD set F also satisfy $\alpha \rightarrow \beta$. We denote the set of all function dependency that logically implied by F as F^+, namely *closure of functional dependency set F*.

Definition 4 (Closure of a Set of Attributes): Closure of a set of attributes X with respect to FD set F is the set X^+ of all attributes that are functionally determined by using F^+.

Given that X, Y and Z are sets of attributes in a relation R, there are several properties of functional dependencies, among which there are three which are most important, called Armstrong's axioms [25]:

- **Reflexivity**. Functional dependency $X \rightarrow Y$ holds if Y is subset of X.
- **Augmentation**. If $X \rightarrow Y$, then $XZ \rightarrow YZ$ where $XZ = \{e | e \in X \; or \; e \in Y\}$ and YZ is similar.
- **Transitivity**. Functional dependency $X \rightarrow Z$ holds if both $X \rightarrow Y$ and $Y \rightarrow Z$ hold.

These three rules are axiomatization of functional dependencies that could be proved sound and complete, and any other rules could be inferred based on these rules. Several important secondary rules are introduced as follows:

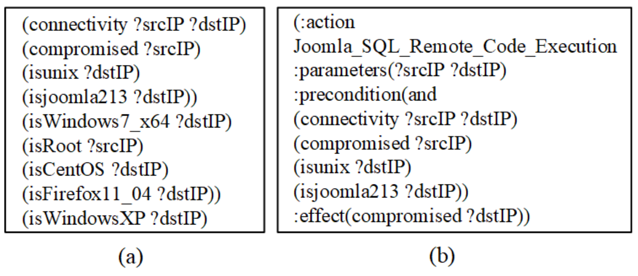
- **Union**. Functional dependency $X \rightarrow YZ$ holds if both $X \rightarrow Y$ and $X \rightarrow Z$ hold.
- **Decomposition**. If $X \rightarrow YZ$, then functional dependency $X \rightarrow Y$ and $X \rightarrow Z$ hold as well.
- **Pseudotransitivity**. Functional dependency $WX \rightarrow Z$ holds if both $X \rightarrow Y$ and $WY \rightarrow Z$ hold.
- **Composition**. Functional dependency $XZ \rightarrow YW$ hold if $X \rightarrow Y$ and $Z \rightarrow W$ hold separately.

As functional dependency theory could lay well foundation for formalizing penetration testing process, we showed an example for better understanding.

B. EXAMPLE

Given a relational pattern *R(U)=R(SID, SNAME, CID, GRADE, CNAME, TNAME, TAGE)*, where each attribute represents student ID, student name, class ID, grade, class name, teacher name and teacher age separately. From attribute set U, we could see that there are five obvious functional dependencies that make up for functional dependency set F that are shown as follows:

- *SID* $\rightarrow$ *SNAME*: Student name is totally determined by student ID.
- *CID* $\rightarrow$ *CNAME*: Class name is totally determined by class ID.



```
(connectivity ?srcIP ?dstIP)
(compromised ?srcIP)
(isunix ?dstIP)
(isjoomla213 ?dstIP))
(isWindows7_x64 ?dstIP)
(isRoot ?srcIP)
(isCentOS ?dstIP)
(isFirefox11_04 ?dstIP))
(isWindowsXP ?dstIP)
```

(a)

```
(:action
Joomla_SQL_Remote_Code_Execution
:parameters(?srcIP ?dstIP)
:precondition(and
(connectivity ?srcIP ?dstIP)
(compromised ?srcIP)
(isunix ?dstIP)
(isjoomla213 ?dstIP))
:effect(compromised ?dstIP))
```

(b)

FIGURE 4. **Formalism (*The left one describes state spaces and the right one describes exploitation action space.*).**

- *CID* → *TNAME*: Class ID could determine teacher name solely.
- *TNAME* → *TAGE*: Teacher age is totally determined by teacher's name.
- *(SNAME, CNAME)* → *GRADE*: Student name and class name could point out grade deterministicly.

Not only these obvious functional dependencies, there are still some other functional dependencies that could be inferred based on Armstrong's axiom rules which are shown as follows:

- *(SID, CID)* → *GRADE*. As *GRADE* is determined by *SNAME* and *CNAME*, *SNAME* is determined by *SID* and *CNAME* is determined by *CID*, Armstrong's axiom pseudotransitivity promises that *GRADE* is determined by *SID* and *CID*.
- *CID* → *TAGE*. Given *CID* → *TNAME* and *TNAME* → *TAGE*, it is easy to prove that *TAGE* is determined by *CID* on the basis of Armstrong's axioms transitivity rule.

The closure of functional dependencies is all functional dependencies that hold in relational pattern $R(U)$. For example, the closure of attributes set {*GRADE*} is {*GRADE, SID, CID, CNAME*}, denoted as $\{GRADE\}^+$.

IV. METHODOLOGY

In this section, we proposed a compact graph planning based algorithm to achieve effective attack paths discovery through calculating closure of penetration testing goal states set. There are three parts in this section: Formalism and closure calculation, compact planning graph construction and solution extraction.

A. FORMALISM AND CLOSURE CALCULATION

Attack paths discovery could be formalized as tuple $< S, A, G >$, where state space S is composed of host information, network topology information and access privilege *etc.* action space A is set of attack action, each of which contains predicates, action name and effects, goal states set G is penetration testing goals, satisfying $G \subseteq S$.

Figure 4 shows an example of state and action element in PDDL representation, from which we could see that there are 9 states on the left side and a *Joomla_SQL_Remote_Code_Execution* action on the right side. In order to transform attack paths discovery problem into functional dependencies representation, state space S is formalized as attribute set, where each element represents specific state, action $a \in A$ is formalized as functional dependency $x \rightarrow y$, where $x = a.preconditions$ and $y = a.effects \subseteq S$ represent precondition and effect states of action a separately. Goal states set G, where each element g represents specific penetration testing goal, could be formalized as subset of state set S. Given goal states set G, the first step is to calculate closure set of G which contains all possible preconditions of g that satisfy functional dependencies transformed from action space.

After formalizing attack paths discovery problem into PDDL, PDDL parser will transform conceptual states and actions into concrete actions and states, which is a heavily time-consuming process. Taking action *Joomla_SQL_Remote_Code_Execution* for example, there will be 16 actions given object sets{192.168.1.2, 192.168.1.0/24, 192.168.4.2, 192.168.4.0/24} because the action is composed of 2 parameters: ?srcIP and ?dstIP, whose combinatorial number is 16. Each combination will turn conceptual states into a new concrete action. And it is the same to states space. After transformation, state space S is composed of all transformed concrete states and the same to action space A. It's easy to find that there will be many unreasonable states and actions, such as *connectivity 192.168.1.2 192.168.1.0/24*, *isjoomla213 192.168.1.0/24* and so on. As there are many irrelevant states in whole state and action space, it's essential to remove those irrelevant states and actions so as to accelerate the efficiency of attack paths discovery. Traditional graph planning algorithm does not take deep consideration into pruning those irrelevant states, so that the attack paths efficiency is relatively low. Further, as action is totally determined by states, redundant states points out redundant actions implicitly. What only need to be done is pruning irrelevant states. In other words, it's essential to pick out those states that are relevant to goal states, namely closure of goal states set. Functional dependency theory shows the way to calculate closure of goal states set. There are two steps for closure calculation, the first of which iterates functional dependencies of action space to find the preconditions of goal states and the second takes the preconditions as new goals to calculate recursively until no new goal state appears. The closure calculation method is summarized in algorithm 1 which could promise the result closure set is sound and complete, namely any states that are relevant to goal states are all included in result closure set. The details of sound and complete proof of closure set could be referenced by [26], [27].

B. COMPACT PLANNING GRAPH CONSTRUCTION

After calculating closure of goal states set, compact planning graph is constructed layer by layer that is composed of two kinds of nodes, proposition nodes and action nodes. Proposition nodes describes available states currently, containing host information, network topology information *etc.* and action nodes describes available actions given current

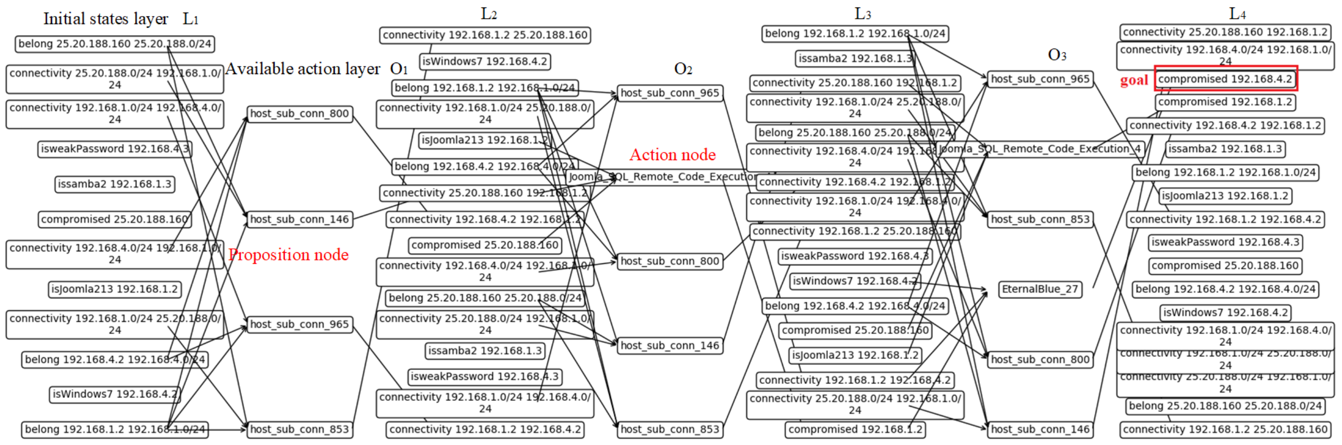


FIGURE 5. **Example of compact planning graph extension (*Graph starts from initial states and extends by iterating action and state space until reaching goal states set or stabilizing.*).**

Algorithm 1 Closure Calculation Algorithm

```
Input:
    S: state space, contains combination of host, network
    topology and access privilege information
    G: goal states set, satisfying $g \in G \subseteq S$
    A: action space, satisfying $\beta \rightarrow \gamma \in A$ where $\beta, \gamma \in S$
    represent preconditions and effects of specific action
Output:
    closure of goal states set: $G^+$

1: $G^+ = G$
2: repeat
3:     for each $\beta \rightarrow \gamma \in A$:
4:         if $\beta \subseteq G^+ : G^+ \leftarrow G^+ \cup \gamma$
5: until $G^+$ not changed
6: return $G^+$
```

states, containing scanning action, exploitation action *etc.* As shown in Figure 5, compact planning graph starts from L_1 which represents the initial states containing *compromised 25.20.188.160*, *connectivity 25.20.188.0/24 192.168.1.0/24*, *isWindows7 192.168.4.2*, *isjoomla213 192.168.1.2* and so on. Now consider iteration i where $i \geq 2$, the action set O_i is the set of all actions whose preconditions are subset of L_{i-1}, *Joomla_SQL_Remote_Code_Execution* in this case, and the set L_i is the newly extended layer that unions effects of all actions in O_i, for example, *compromised 192.168.1.2* in L_3. Also, all states in previous layer should be included in next layer without modification because these states could be used in further steps which is called maintenance action denoted as e. The iteration continues until the compact planning graph stabilizes, which means that $L_{i+1} = L_i$ or encountering goal state set G. As complete state set iteration is a time-consuming process, we use closure of goal state set G^+ to prune those useless branches as so to achieve effective compact planning graph construction.

While extending compact planning graph, it is of great beneficial to record collision pairs within same layer. There are two types of collision pairs, namely action collision pairs and state collision pairs. Action collision pairs indicate that two contradictory states hold within same layer, for example, CVE-2017-7494 will crash samba service and any other vulnerabilities of samba service could not be exploited in the same layer. For each layer, a pair, $o, o' \in O_i$, of actions is defined to be collision if any of these conditions is met:

- **Inconsistent effects**: An effect of o is contradictory to an effect of o'. For example, initial states set contains state ¬ *connectivity 25.20.188.160 192.168.4.2*, *connectivity 25.20.188.160 192.168.4.2* holds through port forward technology after compromising 192.168.1.2, which is contradictory to maintenance action e.
- **Interference**: An effect of o interferes preconditions of o'. For instance, *samba_is_know_pipename* vulnerability CVE-2017-7494 would crash normal service, making it unavailable to other vulnerabilities.

Not only action pair, but state pair may collide as well. Any pair, $l, l' \in L_i$, of states is defined to be collision if the following condition is met:

- **Contradiction**: l and l' form contradictory pair. For example, ¬ *connectivity 25.20.188.160 192.168.4.2* collides with *connectivity 25.20.188.160 192.168.4.2*.

Collision pairs appear within layer, and it is computed layer by layer during compact planning graph construction process. Though layer extending will be a multi-step process, attack paths discovery will limit attack path length. Supposing there are M IP address in a specific network, the maximum attack path length will be $M-1$ at most because once an host is compromised there is no need to attack it again. We summarized compact planning graph construction process in algorithm 2.

C. ATTACK PATHS EXTRACTION

As shown in Figure 6, the compact planning graph could be searched for a solution at this point once compact planning

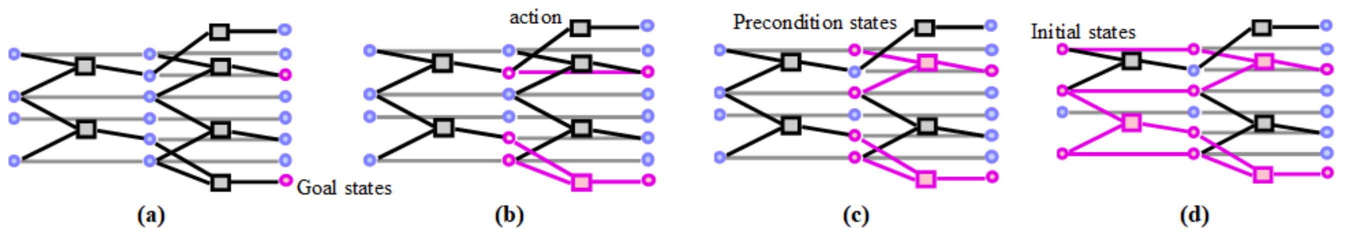


FIGURE 6. Attack path extraction (*(a) represents goal states in L_3; (b) finds that one precondition for goal states could not be satisfied; (c) retries another action in the same layer; (d) discovers attack paths successfully.*).

Algorithm 2 Compact Planning Graph Extension

Input:
goal states set G, closure of goal states set G^+, action space A, host number M, old graph *Graph*, layer *step*

Output:
compact planning graph: *newGraph*

```
1:  repeat
2:      if $step \le M - 1$ then
3:          for each $a \in A$:
4:              if $a.preconditions \subseteq L_{i-1}$ and $a.effects \subseteq G^+$ :
5:                  $O_i.add(a)$;
6:                  $L_i.add(a.effects)$;
7:          $L_i = L_i \cup L_{i-1}$
8:          $newGraph.add(O_i)$
9:          $newGraph.add(L_i)$
10:     else
11:         return None
12:     end if
13: until $G \subseteq L_i$ or $L_{i-1} = L_i$
14: return $newGraph$
```

graph has been constructed up to L_i. Here, depth-first based backward search strategy [28] is adopted to extract solution from graph. Given goal states set G, and there are three conditions for solution extraction from L_i which could be described as follows:

- $G \nsubseteq L_i$. In this case, there is no attack paths solution given current network information because goal states are not included in layer L_i and no new state will appear in further layers.
- $G \subseteq L_i$ and $L_i \neq L_{i+1}$. In this case, an attack paths solution may exists. If backward search could not find a solution, then it can be extended further by adding O_i and L_{i+1}. The extended graph can then be searched for a solution plan recursively.
- $G \subseteq L_i$ and $L_i = L_{i+1}$. In this case, an attack paths solution may exists. If backward search could not find a solution, then there is no solution given current network information because no new state will appear in further layers.

Attack paths discovery algorithm derived from the graph planning algorithm which interleaves graph extension and solution search. Either a solution is reported at some layer or the algorithm correctly reports that no available attack path exists after the compact planning graph stabilizes. Rather than a fully specified plan, the extracted solution is a layered plan, which is a special form of partial plan [29]. All of solution actions are distributed in some specific layers, and the layered plan could be represented as follows:

$$(R_1, R_2, \ldots, R_k) \tag{2}$$

where each R_i is a set of available exploitation actions. Within any specific R_i, the available actions are noncontradictory and may be applied in any order without interfering the implementation of final penetration testing goal states. The only constraint is that every action in R_i must be applied before any actions in R_{i+1} for each i from 1 to k in any specific attack path solution.

Algorithm 3 Attack Paths Discovery Algorithm

Input:
goal states set G, state space S, initial states set I_0, action space A, Host number M.

Output:
attack paths: $< R_1, R_2, R_3, \cdots, R_k >$

```
1:  $G^+ \leftarrow Closure(G)$ /* closure calculation in alg.1 */
2:  $step \leftarrow 1$
3:  while $step \le M - 1$ do
4:      $newGraph \leftarrow extendGraph(G, A, M, graph, step, G^+)$
        /* compact planning graph extension in alg.2 */
5:      $solution \leftarrow backwardSearch(newGraph)$ /* solution
        extraction based on depth-first backward search */
6:      if $solution$ is None then
7:          $graph \leftarrow newGraph$
8:          $step \leftarrow step + 1$
9:      else
10:         return $solution$
11:     end if
12: end while
13: return None
```

In order to obtain a fully specified attack path, the layered plan needs to be linearized by specifying a cause-effect order for the actions that is consistent with layer constraints. For example, weak password action(CVE-2018-0121) should be

TABLE 1. Time consumption of attack path discovery algorithms over increasing host number on 50 separate experiments.

Host number	7	8	9	10	11	12	13
Graph Planning	2.875±0.359	2.861±0.340	2.952±0.339	3.015±0.342	3.073±0.337	3.141±0.379	3.180±0.383
MulVAL	0.308±0.055	0.368±0.083	0.309±0.060	0.311±0.066	0.325±0.075	0.319±0.062	0.310±0.063
Forward Search	2.138±0.633	2.133±0.599	2.178±0.587	2.187±0.589	2.242±0.601	2.302±0.655	2.350±0.685
Compact Graph Planning	**1.930**±0.249	**1.890**±0.212	**1.867**±0.220	**1.850**±0.224	**1.895**±0.205	**1.921**±0.190	**1.862**±0.177
Host number	14	15	16	17	18	19	20
Graph Planning	3.270±0.371	3.393±0.393	3.383±0.417	3.430±0.409	3.510±0.433	3.554±0.423	3.646±0.434
MulVAL	0.317±0.061	0.317±0.070	0.317±0.062	0.323±0.067	0.306±0.058	0.319±0.072	0.314±0.058
Forward Search	2.367±0.666	2.438±0.649	2.484±0.665	2.492±0.662	2.562±0.714	2.616±0.810	2.638±0.718
Compact Graph Planning	**1.903**±0.221	**1.933**±0.154	**1.925**±0.193	**1.959**±0.187	**1.930**±0.147	**2.003**±0.220	**2.004**±0.220
Host number	21	22	23	24	25	26	27
Graph Planning	3.682±0.453	3.751±0.452	3.910±0.458	3.866±0.467	3.943±0.480	4.013±0.497	4.053±0.482
MulVAL	0.321±0.094	0.313±0.058	0.307±0.048	0.306±0.055	0.310±0.057	0.309±0.055	0.309±0.065
Forward Search	2.673±0.811	2.761±0.927	2.770±0.794	2.797±0.790	2.847±0.762	2.859±0.744	2.921±0.777
Compact Graph Planning	**1.989**±0.164	**2.047**±0.206	**2.025**±0.218	**2.050**±0.258	**2.068**±0.221	**2.074**±0.210	**2.073**±0.207
Host number	28	29	30	31	32	33	34
Graph Planning	4.183±0.589	4.182±0.481	4.342±0.548	4.357±0.482	4.386±0.540	4.432±0.558	4.461±0.527
MulVAL	0.327±0.069	0.325±0.072	0.316±0.067	0.298±0.048	0.319±0.085	0.317±0.069	0.313±0.066
Forward Search	2.952±0.771	2.994±0.819	3.100±1.122	3.211±1.587	3.156±0.979	3.132±0.823	3.184±0.803
Compact Graph Planning	**2.060**±0.220	**2.042**±0.252	**2.108**±0.216	**2.124**±0.247	**2.119**±0.201	**2.120**±0.228	**2.126**±0.215
Host number	35	36	37	38	39	40	41
Graph Planning	4.557±0.565	4.663±0.607	4.741±0.594	4.916±0.592	4.925±0.615	4.984±0.567	5.061±0.668
MulVAL	0.317±0.081	0.304±0.060	0.303±0.049	0.309±0.073	0.316±0.065	0.334±0.153	0.312±0.068
Forward Search	3.248±0.847	3.332±0.873	3.370±0.906	3.430±0.973	3.449±0.902	3.510±0.899	3.544±0.917
Compact Graph Planning	**2.113**±0.194	**2.178**±0.248	**2.169**±0.240	**2.146**±0.284	**2.236**±0.218	**2.141**±0.199	**2.214**±0.222
Host number	42	43	44	45	46	47	48
Graph Planning	5.092±0.607	5.165±0.621	5.247±0.609	5.270±0.630	5.380±0.646	5.501±0.639	5.462±0.622
MulVAL	0.322±0.070	0.319±0.071	0.315±0.067	0.305±0.063	0.320±0.063	0.329±0.078	0.326±0.087
Forward Search	3.599±0.938	3.636±0.953	3.665±0.935	3.685±0.946	3.723±0.969	3.812±0.988	3.837±0.982
Compact Graph Planning	**2.186**±0.201	**2.181**±0.258	**2.228**±0.220	**2.208**±0.256	**2.266**±0.267	**2.241**±0.236	**2.317**±0.281
Host number	49	50	51	52	53	54	55
Graph Planning	5.522±0.642	5.581±0.670	5.708±0.686	5.712±0.700	5.787±0.688	5.880±0.688	5.933±0.735
MulVAL	0.319±0.076	0.330±0.095	0.319±0.096	0.316±0.069	0.305±0.058	0.303±0.051	0.307±0.059
Forward Search	3.905±0.989	3.944±1.026	3.966±0.996	4.020±1.043	4.035±1.049	4.078±1.054	4.180±1.117
Compact Graph Planning	**2.259**±0.323	**2.263**±0.282	**2.284**±0.286	**2.392**±0.325	**2.317**±0.261	**2.248**±0.259	**2.348**±0.301

before eternal blue action(CVE-2017-0146) and after web SQL injection action(CVE-2017-12615) in Figure 1. And we summarize compact planning graph based attack path discovery algorithm as follows, where *Closure(G)* stands for closure calculation process for penetration testing goals, *extendGraph* represents compact graph planning extension process described in algorithm 2.

V. EXPERIMENT

A. EXPERIMENT SETUP

Experiment was carried out on typical enterprise network to compare our algorithm against existing attack paths discovery algorithms. As shown in Figure 7, the experiment network structure is composed of three parts: DMZ area, workspace area and inner service area that are connected by a firewall and router. The firewall rules allows the internet host to visit web server in DMZ area, but not workspace hosts and inner services. Hosts in workspace could visit not only DMZ area but inner service area. Given the network structure shown above, the basic idea of penetration testing is composed of three steps: First, the hacker breaks into the DMZ area via SQL injection. Then, the hacker discovers the vulnerabilities of the workspace hosts, and exploits them to get login user and password of file server. And eventually hacker obtained sensitive information on file server based on achieved information.

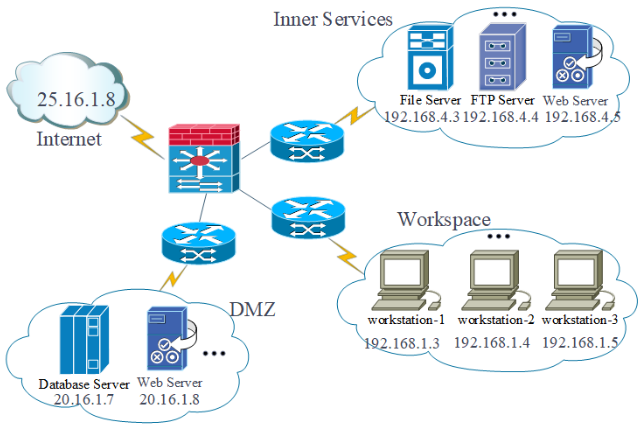


FIGURE 7. Experiment network.

The baseline approaches we compare in the experiment includes:

Forward search algorithm [30]: Forward search starts with initial states and finds all available action successors until resulting states contain goal states.

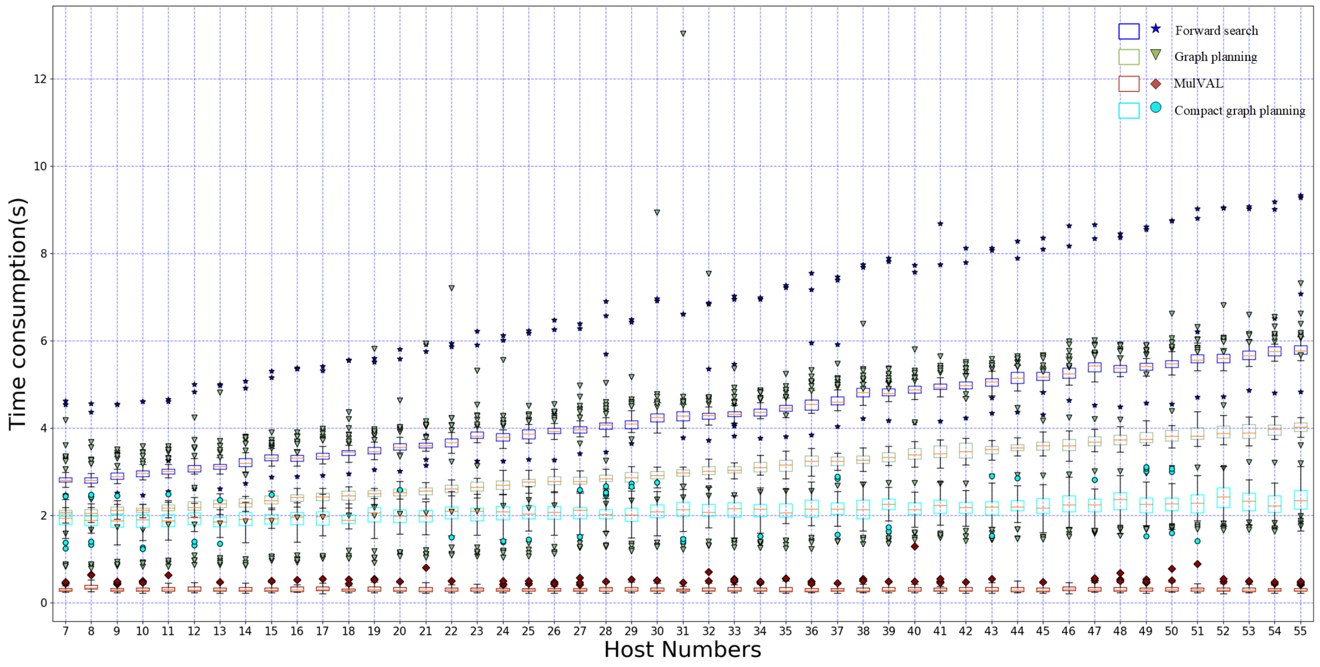


FIGURE 8. **Visualization of time consumption with attack path discovery algorithms over increasing host on 50 separate experiments (*Box represents mean value and standard deviation for individual attack paths discovery algorithms while discrete points represent outliers.*).**

Graph planning algorithm [31]: Graphplan is a general-purpose planner for STRIPS-style domains. There are two core steps in graph planning, containing planning graph construction and solution extraction.

MulVAL [5]: MulVAL stands for "Multi-host, Multi-stage Vulnerability Analysis Language". It is a research tool for security practitioners and system administrators to better manage the configuration of an enterprise network such that the security risks are appropriately controlled.

B. EXPERIMENT RESULT

As host number plays an important role in evaluating effectiveness of attack paths discovery algorithms, Table 1 tells the relationship between increasing host number and mean value, standard deviation of attack paths discovery time consumption after fixing vulnerability number to 5. Generally, it is easy to find from Table 1 that the proposed compact graph planning based attack paths discovery algorithm shows great advantage in discovering hidden attack paths than graph plan and forward search based algorithms. The mean value of graph planning based attack paths discovery algorithm is about 3s from the beginning and ends with 5.7s. Forward search based attack paths discovery algorithms showed relative lower time consumption from 2.2s to 4s. However, our proposed compact graph planning based attack paths discovery algorithm shows average time consumption from 1.9s to 2.3s, which is less than graph planning, forward search based attack paths discovery algorithms. The compact graph planning based attack paths algorithm benefits from

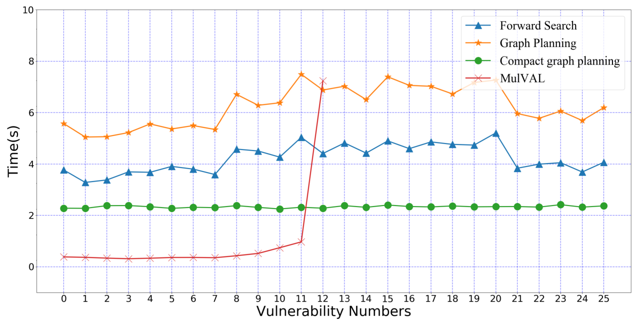


FIGURE 9. **Time consumption of attack paths discovery algorithms over increasing vulnerability number.**

eliminating useless states effectively at the very beginning while the other algorithms could not eliminate those redundant states, resulting in increasing time consumption. The time consumption of MulVAL is the least because increased host shows no effect on enlarging branches of attack graph, so that the time consumption is limited. Even though the time consumption of proposed algorithm is not as good as MulVAL when host number increases, it shows great advantage when vulnerability number increases which will be described in detail later.

In order to describe the relationship between time consumption and host number intuitively, the box relation diagram of time consumption and host number is showed in Figure 8. It is easy to find that both graph planning and forward search based attack paths discovery algorithms show linear increasing time consumption along with host number

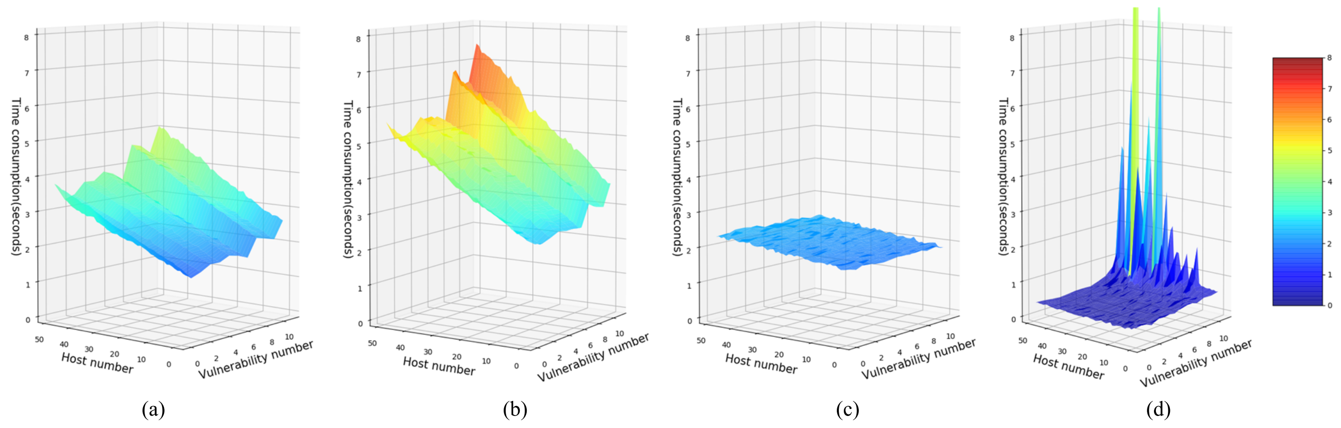


FIGURE 10. Time consumption of attack path discovery algorithms over increasing host and vulnerability number. (*The performance of forward search, graph planning, compact graph planning and attack graph based attack paths discovery algorithms are shown in subfigure (a)(b)(c)(d) sequentially.*).

while compact graph planning and MulVAL share steady time consumption no matter how host number changes. Also, it turns out that graph planning based algorithm spends much time discovering hidden attack paths than the other three algorithms because there are many irrelevant states during solution extracting process. For example, irrelevant state *connectivity 192.168.1.0/24 192.168.4.3* appears as precondition towards goal state *compromised 192.168.4.3*, but it has no help for finding attack paths because state *connectivity 192.168.1.0/24 192.168.4.3* will never be satisfied by initial conditions. Not only irrelevant state, but irrelevant action would cause invalid time consumption as well, for example, action *Joomla_SQL_Remote_Code_Execution_2* makes no sense in finding hidden attack paths because irrelevant state *connectivity 192.168.1.0/24 192.168.4.3* appears as precondition for the action. Theoretically, supposing that there are m objects, h goals, n actions and each action contains k preconditions. There are totally hnm^k actions which should be traversed backward by graph planning based algorithm within each layer. The longer attack path it is, the more actions and states it would iterate, making it time wasting process. And it is the same to forward search based algorithm. However, compact graph planning based algorithm could prune irrelevant actions and states according to dependency relationship between goals and actions, so that there will only be several relevant actions and states occurred in compact planning graph structure, making solution extraction process much more effective.

Fixing host number to 57, Figure 9 tells how time consumption changed along with vulnerability number axis from the view of respective approach in a general way. In Figure 9, the green lines denote the time consumption of compact graph planning based attack paths algorithm along with vulnerability number axis, while blue, orange and red lines represent time consumed by forward search, graph planning and attack graph based attack paths algorithms. It is easy to tell that compact graph planning based algorithm outperforms the other three algorithms in discovering attack paths with increasing size of vulnerability number. Forward search based and graph planning based algorithms share similar development of time consumption and increasing vulnerability number shows little influence on performance of attack graph and the proposed compact graph planning based algorithm at the very beginning. But when vulnerability number reaches 12, the time consumption of attack graph based algorithm grows exponentially because of state space combinational explosion problem, limiting its application in large network scenario. However, compact graph planning based algorithm shows great robustness than MulVAL over increasing size of vulnerability number. This is because increasing vulnerabilities create much irrelevant actions that would not lead to goal states, and compact graph planning based algorithm do not need to traverse those irrelevant actions while other algorithms do not own the ability to prune those actions, so that increasing vulnerability number shows much influence on time performance.

Figure 10 tells the time consumption of all attack path discovery algorithms on both vulnerability number and host number. Performance of forward search, graph planning, compact graph planning and attack graph based algorithm are shown in subfigure (a)(b)(c)(d) separately, from which we could see that host number shows much influence on time performance than vulnerability number for both forward search and graph planning based algorithms, especially for graph planning based algorithm. Attack graph based attack paths discovery algorithm shows good performance when vulnerability number is less than 12, after which, time consumption grows exponentially, making it hard to apply in complicated network scenario. However, our proposed compact graph planning based attack paths discovery algorithm shows little reaction to increasing size of both action number and vulnerability number because all of those irrelevant states and actions are pruned before attack paths discovery process. After pruning irrelevant states and actions, only few states and actions will be left for attack paths discovery with compact graph planning.

TABLE 2. Precision of each attack path discovery algorithm for successful penetration testing under experiment scenario.

Algorithms	Success	Times	Precision
Forward Search	False	1	0
MulVAL	True	32	3.13%
Graph planning	True	9	11.11%
Compact graph planning	True	9	11.11%

Fixing host number to 57 and vulnerability number to 50, we compare the precision of attack paths discovery by counting tried penetration times for final successful penetration testing through metasploit [32] under experiment scenario. The result is shown in table 2, from which we could see that attack graph, graph planing and compact graph planning based attack paths discovery algorithms could achieve successful penetration testing eventually, while forward search based algorithm could not because it finds only one wrong attack path during discovering process. The other three algorithms achieve successful penetration by discovering all possible attack paths where each step matches preconditions of specific vulnerability. However, as state space of attack graph model is far large than graph plan and compact graph planning based attack paths discovery algorithms, there are much branches in attack graph, causing large tried penetration times, so that graph plan and compact graph planning based algorithms show great advantage in discovering attack paths. Even though graph plan based algorithm shares same precision with compact graph planning based attack paths discovery algorithm, it consumes much time than compact graph planning based algorithm shown in figure 10, proving the effectiveness of compact graph planing based attack paths discovery algorithm.

VI. CONCLUSION

In this paper, we proposed a compact graph planning based attack paths discovery algorithm to discover hidden attack paths. Firstly, we formalized attack paths discovery problem into PDDL and calculate the closure of goal states based on functional dependency theory. Secondly, we construct compact planning graph to describe attack paths discovery process based on achieved goal states closure. Thirdly, solution is extracted through depth-first backward search algorithm. Finally, the experimental results demonstrated that our proposed compact graph planning based attack paths discovery algorithm could discover hidden attack paths effectively when compared with existing known attack paths discovery algorithms. And we hope that our research work could contribute to future studies.

REFERENCES

[1] (2019). *National Internet Emergency Response Center (CNCERT-CC)*. [Online]. Available: http://www.cert.org.cn/publish/main/46/index.html

[2] N. Polatidis, M. Pavlidis, and H. Mouratidis, "Cyber-attack path discovery in a dynamic supply chain maritime risk management system," *Comput. Standards Interfaces*, vol. 56, pp. 74–82, Feb. 2018.

[3] MooseDojo. (2019). *APT2—An Automated Penetration Testing Toolkit*. [Online]. Available: https://github.com/MooseDojo/apt2

[4] NullArray. (2019). *AutoSploit–Automated Mass Exploiter*. [Online]. Available: https://github.com/NullArray/AutoSploit

[5] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A logic-based network security analyzer," in *Proc. USENIX Secur. Symp.*, 2005, p. 8.

[6] G. Frances and H. Geffner, "Modeling and computation in planning: Better heuristics from more expressive languages," in *Proc. 25th Int. Conf. Automated Planning Scheduling*, 2015, pp. 70–78.

[7] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *Proc. Int. Symp. Formal Methods Compon. Objects*. Berlin, Germany: Springer, 2003, pp. 344–371.

[8] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," in *Proc. DISCEX*, Jun. 2001, pp. 307–321.

[9] B. Zhang, K. Lu, X. Pan, and Z. Wu, "Reverse search based network attack graph generation," in *Proc. Int. Conf. Comput. Intell. Softw. Eng. (CiSE)*, Dec. 2009, pp. 1–4.

[10] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," in *Network Security Metrics*. Cham, Switzerland: Springer, 2017, pp. 53–73.

[11] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 336–345.

[12] X. Sun, J. Dai, P. Liu, and A. Singhal, and J. Yen, "Using Bayesian networks for probabilistic identification of zero-day attack paths," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 10, pp. 2506–2521, Oct. 2018.

[13] S. Pan, T. Morris, and U. Adhikari, "Developing a hybrid intrusion detection system using data mining for power systems," *IEEE Trans. Smart Grid*, vol. 6, no. 6, pp. 3104–3113, Nov. 2015.

[14] Y. Ding, W. Li, D. Zhong, H. Huang, Y. Zhao, and Z. Xu, "System states transition safety analysis method based on FSM and NuSMV," in *Proc. 2nd Int. Conf. Manage. Eng., Softw. Eng. Service Sci.*, 2018, pp. 107–112.

[15] Z. Chao, W. Huiqiang, G. Fangfang, Z. Mo, and Z. Yushu, "A heuristic method of attack graph analysis for network security hardening," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery (CyberC)*, Oct. 2014, pp. 43–47.

[16] S. Wang, G. Tang, G. Kou, and Y. Chao, "An attack graph generation method based on heuristic searching strategy," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 1180–1185.

[17] S. Shunhong *et al.*, "Method of network attack graph generation based on greedy policy," *Comput. Eng.*, vol. 37, no. 2, pp. 126–135, 2011.

[18] D. E. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*. Amsterdam, The Netherlands: Elsevier, 2014.

[19] J. Yuen, "Automated cyber red teaming," Defence Sci. Techonol., Org. Edingburg, Cyber Electron. Warfare Division, Edingburg, Scotland, Tech. Rep. 6, 2015.

[20] J. L. Obes, C. Sarraute, and G. Richarte. (2013). "Attack planning in the real world." [Online]. Available: https://arxiv.org/abs/1306.4044

[21] C. Sarraute. (2013). "Automated attack planning." [Online]. Available: https://arxiv.org/abs/1307.7808

[22] M. Boddy, J. Gohde, T. Haigh, and S. Harp, "Course of action generation for cyber security using classical planning," in *Proc. ICAPS*, 2005, pp. 12–21.

[23] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z. S. Byrne, and J. M. Weidert, "Personalized vulnerability analysis through automated planning," in *Proc. Working Notes IJCAI Workshop Intell. Secur. (SecArt)*, 2011, p. 50.

[24] C. Combi *et al.*, "Mining approximate temporal functional dependencies with pure temporal grouping in clinical databases," *Comput. Biol. Med.*, vol. 62, pp. 306–324, Jul. 2015.

[25] K. Deuser and P. Naumov, "Armstrong's axioms and navigation strategies," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 6343–6350.

[26] W. W. Armstrong, "Dependency structures of database relationship," *Inf. Process.*, vol. 54, pp. 580–583, Feb. 1974.

[27] B. Stephane. "Functional dependencies," in National University of Singapore. [Online]. Available: https://www.comp.nus.edu.sg/~lingtw/dependencies.pdf

[28] N. R. Sturtevant and A. Felner, "A brief history and recent achievements in bidirectional search," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 8000–8006.

[29] D. Wang and B. Williams, "tburton: A divide and conquer temporal planner," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 3409–3417.

[30] S. Johansen and B. Nielsen, "Analysis of the forward search using some new results for martingales and empirical processes," *Bernoulli*, vol. 22, no. 2, pp. 1131–1183, 2016.

[31] W. M. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio, "Heuristic planning for PDDL+ domains," in *Proc. Workshops 13th AAAI Conf. Artif. Intell.*, 2016, pp. 615–623.

[32] F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta, "Effective penetration testing with Metasploit framework and methodologies," in *Proc. IEEE 15th Int. Symp. Comput. Intell. Inform. (CINTI)*, Nov. 2014, pp. 237–242.

ZANG YICHAO received the B.S. and M.S. degrees in computer science from the Information and Engineering University, Zhengzhou, China, in 2014 and 2017, respectively, where he is currently pursuing the Ph.D. degree in computer software and theory with the State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include data mining, machine learning, cybersecurity, and automated planning.

ZHOU TIANYANG received the B.S. and M.S. degrees from the Zhengzhou Science and Technology Institute, Zhengzhou, China, in 2002 and 2011, respectively, where he is currently pursuing the Ph.D. degree in computer software and theory.

Since 2018, he has been an Associate Professor with the State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China. He is the author of three books and over 20 articles. His research interests include the software vulnerability analysis, virtualization-based security technology and application, penetration test, fundamental study of network modeling and simulation, and cyber security assessment.

Mr. Zhou became a member of the China Computer Federal, in 2011. His awards and honors include the Honor Member of the CCF YOCSEF (Zhengzhou) Academic Committee, the CCF YOCSEF Excellent Academic Forum Secretary, and the Henan Province Science and Technology Progress Award.

GE XIAOYUE received the B.S. degree in integrated circuit design and integration system from the Harbin University of Science and Technology, Harbin, China, in 2017. She is currently pursuing the M.S. degree in computer software and theory with the State Key Laboratory of Mathematical Engineering and Advanced Computing. Her research interests include cybersecurity, game theory, machine learning, automated planning, and policy selection.

WANG QINGXIAN received the B.S. degree from the Department of Applied Mathematics, Zhengzhou Information Science and Technology Institute (ISTI), in 1982, and the M.S. degree from the Department of Computer Science and Technology, Peking University. He is currently a Professor with ISTI. His current research interests include network security, mobile security, intrusion detection, malicious code analysis, and vulnerability discovery.