An Intelligent Technique for Generating Minimal Attack Graph

Nirnay Ghosh, S. K. Ghosh
School of Information Technology
Indian Institute of Technology, Kharagpur-721302, India
nirnay.ghosh@gmail.com, skg@iitkgp.ac.in

August 6, 2009

Abstract

Attack graph is a tool to analyze multi-stage, multi-host attack scenarios in a network. It is a complete graph where each attack scenario is depicted by an *attack path* which is essentially a series of *exploits*. Each *exploit* in the series satisfies the pre-conditions for subsequent *exploits* and makes a casual relationship among them. One of the intrinsic problem with the generation of such a full *attack graph* is its scalability. In this work, an approach based on *planner* has been proposed for time-efficient scalable representation of the *attack graphs*. A *planner* is a special purpose search algorithm from *artificial intelligence* domain, used for finding out solutions within a large state space without suffering *state space explosion*. A case study has also been presented and the proposed methodology is found to be efficient than some of the earlier reported works.

1 Introduction

In today's enterprise, with increasing dependency on IT infrastructure most of the activities rely on services that are provided by the organizational networks. Therefore, primary objective of a network administrator is to maintain a stable and secure network infrastructure. Present day security technologies include some efficient network scanners such as Nessus [1], Retina [2], Nmap [3], CyberCop [4] and so on. These scanning tools are useful as far as detecting vulnerabilities local to a system but do not identify all conditions for a complete attack to take place, or how different vulnerabilities existing in different systems are correlated to produce attacks potentially more harmful than individual attacks. One such tool that gives description about the correlated attacks in a network is the *attack graph*. It shows the network administrators all possible sequences of attacker actions that eventually lead to the desired level of privilege on the target. In some literatures, *attack graph* is also termed as the *exploit dependency graph* [Noel et al.2003]. Therefore, a complete *attack graph* quickly becomes unmanageably large as the network complexity grows past a few machines. Analysis show that such an *attack graph* has exponential complexity. To improve the complexity of graph generation, some of the approaches [Ammann, Wijesekera, and Kaushik2002] [Noel et al.2003] rely on explicit assumption of *monotonicity*. This means once an attacker has gained certain level of privileges on a particular host, he does not have to regain the same at some later stages of the attack. This removes the concept of back-tracking from the *attack graphs* and the complexity is improved from exponential to polynomial one. However, the *attack graph* which is generated based on *monotonic* assumptions are still not scalable and also contains a number of redundant paths. This creates problem in terms of visual representation. But it is desirable to present the network administrator with *attack graphs* that are understandable so that appropriate network hardening measures may be adopted. The proposed approach deals with the generation of *minimal attack graph* where all the the attack paths terminate to a particular *goal* node [Pamula et al.2006]. In the proposed approach, *SGPlan*, a variant of *Planner* has been used to generate *minimal* attack paths, which are eventually collapsed to form a *minimal attack graph* in polynomial time.

One of the earliest work in the field of *attack graph* was done by Moskowitz et. al. in [Moskowithz and Kang1997]. The authors have used a graph to represent *insecurity flow* to identify the possible loop-holes in a network.

[1] http://www.nesssus.org
[2] http://www.eeye.com/html/products/Retina
[3] http://www.insecure.org/nmap/index.html
[4] http://www.nai.com

Swiler et. al. have formally defined *attack graph* [Phillips and Swiler1998] as a tool which can identify the set of attack paths based on probability of success. Ritchey and Ammann [Ritchey and Ammann2000] have used *SMV model-checkers* to determine if a final goal state is reachable from an attacker starting with limited privileges. Swiler et. al. [Swiler et al.2001] eliminates redundant nodes and finds a set of near-optimal shortest path from a given *attack graph*. An automated technique for generating and analyzing exhaustive and succinct *attack graphs* using symbolic model checking algorithm is presented in [Sheynar et al.2002]. Ammann et. al. [Ammann, Wijesekera, and Kaushik2002] proposed an algorithm for more compact and scalable representation of *attack graphs*. This approach relies on an explicit assumption of monotonicity which reduces the complexity of generation from exponential to polynomial. In [Ammann et al.2005], the authors have presented an intuitive, polynomially efficient, and scalable vulnerability analysis approach, from a penetration tester's perspective, that generates suboptimal attack paths rather than the complete graph. Various literature survey and previously reported works have depicted the various difficulties related to the generation and representation of *attack graphs* namely, scalability and exponential time complexity, presence of redundant nodes and edges, *model checkers* used for generation of *minimal attack graph* suffers from *state space explosion* problem.

In this work, an intelligent approach is proposed for generation of *minimal attack graph* using *SGPlan*, a variant of *Planner* from *artificial intelligence* domain. Therefore, the aim of the work is *time efficient generation of a minimal attack graph using a model-checker that removes visualization problems and avoids state-space explosion.* The organization of the rest of the papers is as follows. Section 2 gives a brief overview of *planner*. Section 3 describes the proposed methodology along with a case study. The conclusion is drawn in section 4.

2 Planner

Planner [Blum and Furst1997] is a special purpose search algorithm in *artificial intelligence* domain for finding out solution within a large state space. In this work, a variant of *Planner*, called *SGPlan*, is used for finding the attack paths. Initial state, *goal* state and the state transition operators are provided as input to the *Planner*. The input specifications are written in *PDDL* [Fox and Long2003](Planning Domain Definition Language) in two files viz. *domain.pddl* and *fact.pddl*. The *domain.pddl* contains un-instantiated predicates and state transition operators. These un-instantiated predicates are initialized by real world entities using a number of *objects* and *STRIPS* operators [Fox and Long2003] to represent initial state and goal state in the *fact.pddl*. Appropriate changes in the *fact.pddl* allows the *Planner* to discard the previous plan and search for the new plan.

The *Planner* begins its execution from the initial state with a graph based representation called *plangraph*. The *plangraph* is generated starting from the initial state and successive application of state transition operators. The generation of *plangraph* consumes the major amount of time in the entire attack path identification process. With n number of objects, m number of STRIPS-like operators each having maximum k number of constant formal parameters, the generation time for a t–level *plangraph* will be polynomial as maximum generated nodes in any action level will be $O(mn^k)$ [Blum and Furst1997]. The motivation behind selecting *planner* as a technique for generating *attack paths* are as follows:

- It prunes unnecessary actions from the system and finds the *shortest path.*
- It allows addition of actions to the plan where ever and whenever they are required.
- It uses richer input language, *PDDL*, to express complex state space domains relatively easier than custom-built analysis engines.
- It does not suffer from *state space explosion* problem.

3 Generation of Minimal Attack Graph Using Planner

In this section, an approach to generate *minimal attack graph* using *Planner* has been proposed. The objective for preferring generation of *minimal attack graph* to *complete attack graphs* are as follows:

- *Minimal attack graph* consists of only those attack paths which terminate to a particular *goal* node. Therefore, it does not contain redundant nodes or edges, and enables a network administrator to have a *better visualization and apprehension of different attack scenarios for a network.*

- It is based on explicit assumption of *monotonicity*, which removes the concept of backtracking from *attack graphs* and reduces the generation time from exponential to polynomial.
- As *planner* generates acyclic paths, collapsing them will always result in a *minimal attack graph.*

The overall mechanism is shown in figure 1. It starts with the assumption that the initial network configurations and the vulnerability analysis has been done apriori and are input to the *Planner* engine i.e. the *domain.pddl* and the *fact.pddl* files written in PDDL [Fox and Long2003]. With the initial network configurations, connectivity relationships, vulnerability analysis, a *minimal* attack path is generated. To generate other *minimal* attack paths, the *fact.pddl* file is modified. If all the attack paths are generated, they are collapsed to form the *minimal attack graph.* The proposed methodology has been explained with the help of a case study in the following section.

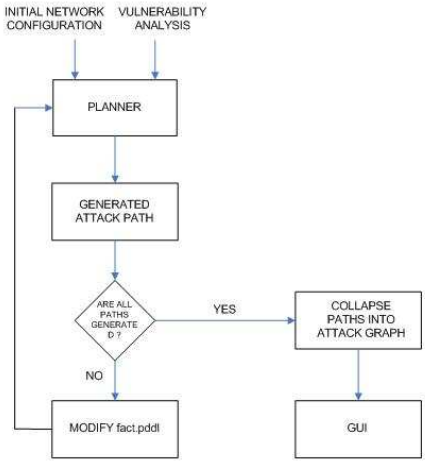


Figure 1: Flow Chart showing *Planner* actions

3.1 Case Study

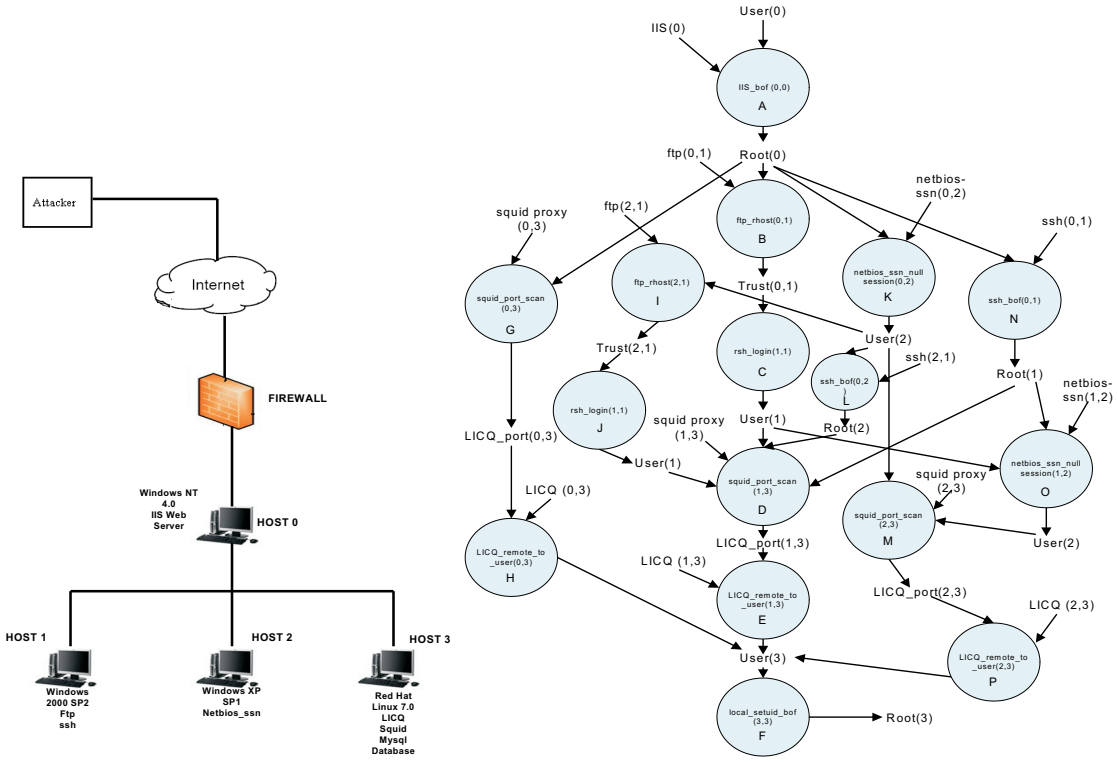
A network similar to [Sheynar2004] has been considered (refer figure 2(a)) as the test network. The network consists of four hosts viz. $Host0(H0)$, $Host1(H1)$, $Host2(H2)$, and $Host3(H3)$. $H3$ is taken as the target machine or *goal* and the MySQL [5] database running on that machine is the critical resource. The system characteristics of the hosts in the network are composed in the table 1. These data are available in Nessus, NVD [6], Bugtraq [7]. Each *generic*

Table 1: System Characteristics

Host	**Services**	**Ports**	**Vulnerabilities**	**CVE – IDs**	**OperatingSystem**
H0	IIS Web Service	80	IIS buffer overflow	CVE-2002-0364	Windows NT 4.0
H1	ftp ssh rsh	21 22 514	ftp rhost overwrite ssh buffer overflow rsh login	CVE-2008-1396 CVE-1999-1455 CVE-1999-0180	Windows 2000 SP1
H2	Netbios-ssn rsh	139 514	Netbios-ssn nullsession rsh login	CVE-2003-0661 CVE-1999-0180	Windows XP SP2
H3	LICQ Squid Proxy Mysql DB	5190 80 3306	LICQ-remote-to-user squid-port-scan local-setuid-bof	CVE-2001-0439 CVE-2001-1030 CVE-2006-3368	Red Hat Linux 7.0

vulnerability present in table 1 has a set of preconditions and effects [Sheynar2004] [Sheynar and Wing2004]. The preconditions and effects of one of the *generic* vulnerabilities viz. *IIS buffer overflow* is given below:

[5] http://www.mysql.com
[6] http://nvd.nist.gov/
[7] http://www.securityfocus.com/archive/



(a) Test network (b) Attack graph

Figure 2: Test network and Attack graph

- **Preconditions** (1) IIS Web Service running on target (2) IIS buffer overflow vulnerability exists (3) Attacker's privilege on target >= user (4) Transport layer connectivity exists between attacker and target
- **Effects** (1) IIS Web Service is disabled on target (2) Attacker gains root level privilege on target

The firewall in the test network (refer figure 2(a)) allows external hosts to connect to IIS web service running on port 80 on $H0$. But connection to all other ports are blocked. The internal hosts are allowed to connect on any port within the network. The connectivity limiting firewall policy are presented in table 2. In table 2, *All* indicates

Table 2: Connectivity-Limiting Firewall Policies

Host	**Attacker**	**H0**	**H1**	**H2**	**H3**
Attacker	All	Yes	None	None	None
H0	None	All	All	All	All
H1	None	Yes	All	All	All
H2	None	Yes	All	All	All
H3	None	Yes	All	All	All

that a source host may connect to any port on a destination host and *None* signifies that the source machine is prevented from accessing any port on the destination machine. Depending upon connectivity limiting firewall policies, each *generic* exploit has some *instantiated* exploits [Ammann, Wijesekera, and Kaushik2002]. Some of the *instantiated* exploits are as follows:-

- *IIS_bof(0,0)*- IIS buffer overflow exploited from $Host0$ on $Host0$.
- *ftp_rhosts (0,1)*- rsh trust from $Host0$ to $Host1$.
- *squid_port_scan (1,3)*- squid port scan done from $Host1$ on $Host3$.

3.2 Identification of Attack Path Using GraphPlan

GraphPlan, a variant of *Planner*, is a search algorithm which finds out solution within a large state space. Initial network configuration, attacker's objective, and exploits are considered as inputs to the *GraphPlan*. In this work, *SGPlan 5.2.2* [8] [Chen, Hsu, and B.Wah2006], a variant of *GraphPlan*, is used as an attack path identification component. *SGPlan* has been preferred to other variants of *GraphPlan* viz. *LPG-td* [9], *Metric-FF* [10], as it supports numeric predicates or fluents, derivative predicates, and durative predicates.

3.2.1 *domain* and *fact* files

As mentioned in section 2, *Planner* requires two files viz. *domain.pddl* and *fact.pddl* to realize network configuration and the vulnerabilities existing in it. An instance of *domain.pddl* is given in table 3 (read left column, then right column). The *domain.pddl* (refer table 3) encodes the following:

Table 3: domain.pddl

(define(domain attackgraph)	(IIS_apps_connectivity ?S ?T)
(:requirements :strips :fluents :equality)	(ftp_apps_connectivity ?S ?T)
(:predicates (IIS_web_service ?H)	(ssh_port_connectivity ?S ?T)
(ftp ?H)	(squid_port_connectivity ?S ?T)
(ssh ?H)	(LICQ_apps_connectivity?S?T)
(rsh ?H)	(rsh_apps_connectivity ?S?T)
(netbios_ssn ?H)	(netbios_apps_connectivity?S?T))
(LICQ_chat_service ?H)	(:functions (has_priv ?A ?H)
(squid_proxy ?H)	(root_priv)
(IIS_bof ?H)	(user_priv)
(ftp_rhost_overwrite ?H)	(none_priv))
(rsh_login ?H)	(:functions
(ssh_bof ?H)	(port_scan ?A ?H)
(netbios_ssn_nullsession ?H)	(none_priv))
(LICQ_remote_to_user ?H)	(port_scan_done)
(local_setuid_bof ?H)	(:action IIS-buffovflw
(IIS_port_connectivity ?S ?T)	:parameters
(ftp_port_connectivity ?S ?T)	(?A
(ssh_port_connectivity ?S ?T)	?S
(squid_port_connectivity ?S ?T)	?T)
(LICQ_port_connectivity ?S ?T)	:precondition
(rsh_port_connectivity ?S ?T)	(and (>=(has_priv ?A?S)(user_priv))(IIS_web_service?T)
(netbios_port_connectivity ?S?T)	(IIS_port_connectivity ?S ?T) (IIS_bof?T(<(has_priv ?A?T)
(IIS_apps_connectivity ?S ?T	(root_priv))))
(ftp_apps_connectivity ?S ?T)	:effect
(ssh_apps_connectivity ?S?T)	(and (not(IIS_bof ?T)) (assign (has_priv ?A?T)(root_priv))
(squid_apps_connectivity?S?T)	(not (IIS_web_service ?T)))
(LICQ_apps_connectivity?S?T)	)
(rsh_apps_connectivity?S ?T)	
(netbios_apps_connectivity?S?T))	

- **Domain name** - given as *attackgraph*
- **Requirements** - specifies the type of operators required viz. *strips, fluents, equality* etc.
- **Predicates** - depicting the services running on hosts, for e.g. *ftp ?H* etc., and the type of vulnerabilities present, for e.g. *ftp_rhost_overwrite ?H* etc.
- **Functions** - e.g. *has_priv* to assign privilege levels.
- **Actions** - describe *state transition operators* in terms of *action rule* specification that has four components: *intruder precondition, intruder effect, network preconditions, and network effects*, for e.g. *IIS_buffovflw*

[8] http://manip.crhc.uiuc.edu/programs/SGPlan/sgplan5.html
[9] http://www.zeus.ing.unibs.it/lpg/
[10] http://www.members.deri.at/ joergh/metric-ff.html

- **Parameters**- contains the constant formal parameters (for e.g. *A, S, T*) used to realize the *actions*

The *fact.pddl* encodes various network *objects* that includes the hosts, the attacker, the firewall etc. An instance of the *fact.pddl* is given in table 4 (refer left column then right column). The *fact.pddl* (refer table 4) encodes the

Table 4: fact.pddl

(define (problem Attack)	(IIS_bof Host0)
(:domain attackgraph)	(ssh_bof Host1)
(:objects	(ftp_rhost_overwrite Host1)
Host0	(rsh_login Host1)
Host1	(netbios_ssn_nullsession Host2)
Host2	(LICQ_remote_to_user Host3)
Host3	(local_setuid_bof Host3)
Attacker	(IIS_port_connectivity Attacker Host0)
)	(ssh_port_connectivity Host0 Host1)
(:init	(ssh_apps_connectivity Host0 Host1)
(= (has_priv Attacker Attacker) 3)	(ssh_port_connectivity Host2 Host1)
(= (has_priv Attacker Host0) 1)	(ssh_apps_connectivity Host2 Host1)
(= (has_priv Attacker Host1) 1)	(ssh_port_connectivity Host3 Host1)
(= (has_priv Attacker Host2) 1)	(ssh_apps_connectivity Host3 Host1)
(= (has_priv Attacker Host3) 1)	(ftp_port_connectivity Host0 Host1)
(= (root_priv) 3)	(ftp_apps_connectivity Host0 Host1)
(= (user_priv) 2)	(ftp_port_connectivity Host2 Host1)
(= (none_priv) 1)	(ftp_apps_connectivity Host2 Host1)
(= (port_scan Attacker Host3) 0)	(ftp_port_connectivity Host3 Host1)
(= (port_scan_not_done) 0)	(ftp_apps_connectivity Host3 Host1)
(= (port_scan_done) 1)	(netbios_port_connectivity Host0 Host2)
(IIS_web_service Host0)	(netbios_apps_connectivity Host0 Host2)
(ssh Host1)	(netbios_port_connectivity Host1 Host2)
(ftp Host1)	(netbios_port_connectivity Host1 Host2)
(rsh Host1)	(netbios_port_connectivity Host1 Host2)
(netbios_ssn Host2)	(squid_port_connectivity Host0 Host3)
(squid_proxy Host3)	(squid_port_connectivity Host1 Host3)
(LICQ_chat_service Host3)	(squid_port_connectivity Host2 Host3)
	(LICQ_port_connectivity Host0 Host3)
	(LICQ_port_connectivity Host1 Host3)
	(LICQ_port_connectivity Host2 Host3))
	(:goal (and(= (has_priv Attacker Host3) 3))))

following attributes:

- **Problem name** - given as *Attack*.
- **Domain name** - same as the one specified in *domain* file i.e., *attackgraph*.
- **Objects** - includes different network objects viz. hosts, firewall etc.
- **Numerical predicates** - with respect to the functions defined in *domain.pddl* for e.g. *(=(root_priv) 3)* which means *root* privilege has been assigned a value of 3.
- **Initial network configuration** - includes services running on hosts ((*IIS_web_service Host0*), transport layer connectivities (*ssh_port_connectivity Host2 Host1*), and application layer connectivities (*netbios_apps_connectivity Host1 Host2*)
- ***Goal* condition** - given as *(:goal (and (= (has_priv Attacker Host3) 3)))*.

SGPlan uses *domain.pddl* and *fact.pddl* to generate single *shortest* attack path. Systematic invalidation of the identified path enables *SGPlan* to identify alternate *shortest* attack path. It depends on the administrator's discretion about which network configurations should be changed to invalidate the paths. Invalidation is done in *fact.pddl* by disabling a service running in one of the hosts, or a connectivity between a pair of hosts by placing a *double-semicolon (;;)* before that predicate. From the given *domain.pddl* and *fact.pddl*, the *shortest* attack path generated by *SGPlan* is as follows.

```
; Time 0.00
; ParsingTime 0.00
; NrActions 4
; MakeSpan
; MetricValue
; PlanningTechnique Modified-FF(enforced hill-climbing search) as
the subplanner

0.001:(IIS-BUFFOVFLW ATTACKER ATTACKER HOST0) [1]
1.002:(SQUID-PORT-SCAN ATTACKER HOST0 HOST3) [1]
2.003:(LICQ-REMOTE-TO-USER ATTACKER HOST0 HOST3) [1]
3.004:(LOCAL-SETUID-BUFFOVRFLW ATTACKER HOST3) [1]
```

SGPlan generated attack path may be re-written in the following way:
$Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow squid\_port\_scan(H0, H3) \rightarrow LICQ\_remote\_to\_user(H0, H3) \rightarrow local\_setuid\_bof(H3, H3)$.

If an alternate attack path is to be generated, the *fact.pddl* needs to be modified. If the transport layer connectivity between $Host0$ and $Host3$ on *Squid_proxy* and *LICQ* services are disabled, *SGPlan* will generate an alternative *shortest* attack path. The modified *fact.pddl* is given in table 5.

Table 5: Modified fact.pddl

(define (problem Attack)	(IIS_bof Host0)
(:domain attackgraph)	(ssh_bof Host1)
(:objects	(ftp_rhost_overwrite Host1)
Host0	(rsh_login Host1)
Host1	(netbios_ssn_nullsession Host2)
Host2	(LICQ_remote_to_user Host3)
Host3	(local_setuid_bof Host3)
Attacker	(IIS_port_connectivity Attacker Host0)
)	(ssh_port_connectivity Host0 Host1)
(:init	(ssh_apps_connectivity Host0 Host1)
(= (has_priv Attacker Attacker) 3)	(ssh_port_connectivity Host2 Host1)
(= (has_priv Attacker Host0) 1)	(ssh_apps_connectivity Host2 Host1)
(= (has_priv Attacker Host1) 1)	(ssh_port_connectivity Host3 Host1)
(= (has_priv Attacker Host2) 1)	(ssh_apps_connectivity Host3 Host1)
(= (has_priv Attacker Host3) 1)	(ftp_port_connectivity Host0 Host1)
(= (root_priv) 3)	(ftp_apps_connectivity Host0 Host1)
(= (user_priv) 2)	(ftp_port_connectivity Host2 Host1)
(= (none_priv) 1)	(ftp_apps_connectivity Host2 Host1)
(= (port_scan Attacker Host3) 0)	(ftp_port_connectivity Host3 Host1)
(= (port_scan_not_done) 0)	(ftp_apps_connectivity Host3 Host1)
(= (port_scan_done) 1)	(netbios_port_connectivity Host0 Host2)
(IIS_web_service Host0)	(netbios_apps_connectivity Host0 Host2)
(ssh Host1)	(netbios_port_connectivity Host1 Host2)
(ftp Host1)	(netbios_port_connectivity Host1 Host2)
(rsh Host1)	(netbios_port_connectivity Host1 Host2)
(netbios_ssn Host2)	;;(squid_port_connectivity Host0 Host3)
(squid_proxy Host3)	(squid_port_connectivity Host1 Host3)
(LICQ_chat_service Host3)	(squid_port_connectivity Host2 Host3)
	;;(LICQ_port_connectivity Host0 Host3)
	(LICQ_port_connectivity Host1 Host3)
	(LICQ_port_connectivity Host2 Host3))
	(:goal (and(= (has_priv Attacker Host3) 3))))

The *shortest* attack path generated by *planner* using modified *fact.pddl* is given as:

```
; Time 0.00
; ParsingTime 0.00
; NrActions 5
; MakeSpan
```

```
;MetricValue
; PlanningTechnique Modified-FF(enforced hill-climbing search) as
the subplanner

0.001:(IIS-BUFFOVFLW ATTACKER ATTACKER HOST0) [1]
1.002:(SSH-BUFFOVFLW ATTACKER HOST0 HOST1) [1]
2.003:(SQUID-PORT-SCAN ATTACKER HOST1 HOST3) [1]
3.004:(LICQ-REMOTE-TO-USER ATTACKER HOST1 HOST3) [1]
4.005:(LOCAL-SETUID-BUFFOVRFLW ATTACKER HOST3) [1]
```

SGPlan generated attack path may be represented in the following way:
$Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow ssh\_bof(H0, H1) \rightarrow squid\_port\_scan(H1, H3) \rightarrow LICQ\_remote\_to\_user(H1, H3) \rightarrow local\_setuid\_bof(H3, H3)$.

Modifying the *fact.pddl* in similar way *six* different attack paths are generated by *SGPlan*. These attack paths are as follows:

1. $Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow netbios\_ssn\_nullsession(H0, H2) \rightarrow squid\_port\_scan(H2, H3) \rightarrow LICQ\_remote\_to\_user(H2, H3) \rightarrow local\_setuid\_bof(H3, H3)$
2. $Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow ftp\_rhost\_overwrite(H0, H1) \rightarrow rsh\_login(H0, H1) \rightarrow squid\_port\_scan(H1, H3) \rightarrow LICQ\_remote\_to\_user(H1, H3) \rightarrow local\_setuid\_bof(H3, H3)$
3. $Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow ssh\_bof(H0, H1) \rightarrow netbios\_ssn\_nullsession(H1, H2) \rightarrow LICQ\_remote\_to\_user(H2, H3) \rightarrow local\_setuid\_bof(H3, H3)$
4. $Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow ftp\_rhost\_overwrite(H0, H1) \rightarrow rsh\_login(H0, H1) \rightarrow netbios\_ssn\_nullsession(H1, H2) \rightarrow squid\_port\_scan(H2, H3) \rightarrow LICQ\_remote\_to\_user(H2, H3) \rightarrow local\_setuid\_bof(H3, H3)$
5. $Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow netbios\_ssn\_nullsession(H0, H2) \rightarrow ssh\_bof(H2, H1) \rightarrow squid\_port\_scan(H1, H3) \rightarrow LICQ\_remote\_to\_user(H1, H3) \rightarrow local\_setuid\_bof(H3, H3)$
6. $Attacker \rightarrow IIS\_bof(Att, H0) \rightarrow netbios\_ssn\_nullsession(H0, H2) \rightarrow ftp\_rhost\_overwrite(H2, H1) \rightarrow rsh\_login(H2, H1) \rightarrow squid\_port\_scan(H1, H3) \rightarrow LICQ\_remote\_to\_user(H1, H3) \rightarrow local\_setuid\_bof(H3, H3)$

These attack paths are input to the customized *attack graph building* algorithm that builds the *attack graph* by collapsing these paths. The algorithm will be discussed the following section.

3.3 Attack Graph Building Algorithm

The *minimal attack paths* obtained from *SGPlan* are collapsed to form the *minimal attack graph*. The *attack graph building* algorithm takes as input a set of attack paths generated by *SGPlan*, a set of *nodes* that constitute the paths, and a *two-dimensional matrix*. The *attack graph building* algorithm is presented below.

Input: A set of attack paths P, a set of nodes N, a 2-D matrix $arr[p][p]$
Output: An attack graph
Initialize $arr[p][p] = \{0\}$;
Enumerate each *node* in N ;
foreach *Path in P* **do**
  foreach *valid directed path from node i to node j* **do**
    Set $arr[i][j] = 1$;
  end
end
foreach *i = 1 to p* **do**
  foreach *j = 1 to p* **do**
    if $arr[i][j] = 1$ **then**
      Draw a directed edge from i to j;
    end
  end
end

Algorithm 1: Attack Graph Building Algorithm

Using algorithm 1, the *attack graph* is shown in figure 2(b). The circles represents the *nodes* in the *attack graph* that contain the *exploits* which the attacker has utilized in different stages of the attack. The texts in the *attack graph* represent the *conditions* obtained by utilizing *exploits* or viceversa.

3.3.1 Complexity Analysis

In section 2, it has been stated that the time complexity for generating a t–level *plangraph* at any action–level is $O(mn^k)$ where the notations have their usual meanings. In the *domain.pddl*, it may be noted that we have used only three formal parameters viz. A, S, and T to represent *an attacker, a source, and a destination* respectively. These three parameters are sufficient to realize any *action*. So k in this case is a constant. Again, the number of *STRIP* operators that have been used for generating the attack paths is bounded by the number of *generic* vulnerabilities existing in the network. Therefore, the time complexity to generate attack paths in any action–level is $O(mn^3)$, where n is the number of objects used in the *fact.pddl* i.e., mainly the number of *hosts* in the network and m is the number of *generic* vulnerabilities present in the hosts of the network.

The algorithm for generating the *attack graph* (refer algorithm 1) is dependent upon the number of *nodes* that constitutes the set of generated attack paths. Again, each node in the *attack graph* represents an *instantiated exploit*. Therefore, the running time of *attack graph building* algorithm is always bounded by $O(e^2)$, where e is the total number of *instantiated exploits*. Hence, the worst-case complexity of generating attack paths and collapsing them into *attack graph* is given as $O(mn^3 + e^2)$.

3.4 Performance Evaluation

The proposed *Planner* based approach for finding the *shortest* attack path and then collapsing these paths to form a *minimal attack graph* has been found to be more efficient than some of the earlier reported works [Ammann, Wijesekera, and Kaushik2002] [Sheynar et al.2002]. In [Ammann, Wijesekera, and Kaushik2002], computation in the initial marking phase of the algorithm grows as n^6e, where n is the number of *hosts* and e is the number of *exploits*. In [Sheynar et al.2002], the complexity of the graph generation algorithm is NP-complete. However, in the proposed approach, the worst-case complexity for finding the *shortest* attack path and then combining these paths to generate *minimal attack graph* take place in $O(mn^3 + e^2)$. For majority of networks, having large number of hosts, this generation of *minimal attack graph* will always be upper-bounded by $O(n^3)$. This is due to the fact that in a real-world network, vulnerabilities on most of the hosts are patched and the *attack graph* of a well-protected network is usually small and sparse [Wang, Noel, and Jajodia2006]. Therefore, barring exceptional cases, the relation $e \ll n$ and $m \ll n$ will always remain valid. Hence, in terms of time-efficiency, the proposed approach gives better performance than [Ammann, Wijesekera, and Kaushik2002].

4 Conclusion

In this work, a method for finding *shortest* attack paths and then collapsing these paths to form a *minimal attack graph* has been proposed. For this purpose, an *artificial intelligence* technique, called *Planner*, has been deployed. It has been shown that the methodology is time efficient in terms of finding the attack paths and building the *attack graph* than some of the already reported works [Ammann, Wijesekera, and Kaushik2002] [Sheynar et al.2002]. The proposed approach may be extended to wireless network where generation of attack paths in timely efficient manner is of utmost importance due to its dynamic nature.

References

[Ammann et al.2005] Ammann, P.; Pamula, J.; Ritchey, R.; and Street, J. 2005. A host-based approach to network attack chaining analysis. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*.

[Ammann, Wijesekera, and Kaushik2002] Ammann, P.; Wijesekera, D.; and Kaushik, S. 2002. Scalable, graph-based network vulnerability analysis. In *Proceedings of CCS 2002: 9^{th} ACM Conference on Computer and Communications Security*, 217–224. ACM Press.

[Blum and Furst1997] Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. In *Journal of Artificial Intelligence*, 281–300.

[Chen, Hsu, and B.Wah2006] Chen, Y.; Hsu, C.; and B.Wah. 2006. Temporal planning using subgoal partitioning and resolution in sgplan. In *Journal of Artificial Intelligence Research*, 323–369.

[Fox and Long2003] Fox, M., and Long, D. 2003. Pddl 2.1: An extension to pddl for expression temporal planning domains. In *Journal of Artificial Intelligence Research*, 61–124.

[Moskowithz and Kang1997] Moskowithz, I. S., and Kang, M. H. 1997. An insecurity flow model. In *Proceedings of the* 6^{th} *New Security Paradigms Workshop*, 61–74.

[Noel et al.2003] Noel, S.; Jajodia, S.; O'Berry, B.; and Jacobs, M. 2003. Efficient minimum-cost network hardening via exploit dependency graph. In *Proceedings of* 19^{th} *Annual Computer Security Applications Conference (ACSAC 2003)*.

[Pamula et al.2006] Pamula, J.; Jajodia, S.; Ammann, P.; and Swarup, V. 2006. A weakest-adversary security metric for network configuration secuirty analysis. In *Proceedings of 2nd ACM Workshop on Quality of Protection*, 31–38. ACM Press.

[Phillips and Swiler1998] Phillips, C., and Swiler, L. P. 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the Workshop on New Security Paradigms (NSPW)*, 71–79.

[Ritchey and Ammann2000] Ritchey, R. W., and Ammann, P. 2000. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 156–165.

[Sheynar and Wing2004] Sheynar, O., and Wing, J. M. 2004. Tools for generating and analyzing attack graphs. In *Proceedings of the Workshop on Formal Methods for Components and Objects (FMCO)*, 344–371.

[Sheynar et al.2002] Sheynar, O.; Jha, S.; Wing, J. M.; Lippmann, R. P.; and Haines, J. 2002. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 273–284.

[Sheynar2004] Sheynar, O. 2004. *Scenario Graphs and Attack Graphs*. Ph.D. Dissertation, Carnegei Mellon University, USA.

[Swiler et al.2001] Swiler, L. P.; Phillips, C.; Ellis, D.; and Chakerian, S. 2001. Computer-attack graph generation tool. In *Proceedings of the 2nd DARPA Information Survivability Conference & Exposition (DISCEX II)*, volume II, 307–321. IEEE Computer Society.

[Wang, Noel, and Jajodia2006] Wang, L.; Noel, S.; and Jajodia, S. 2006. Minimum cost-network hardening using attack graphs. *Computer Communications, 29(18)* 3812–3824.