

# An Attack Graph Based Risk Management Approach of an Enterprise LAN

Somak Bhattacharya, S. K. Ghosh

School of Information Technology  
Indian Institute of Technology, Kharagpur  
Kharagpur 721302, India  
*somakb@sit.iitkgp.ernet.in; skg@iitkgp.ac.in*

**Abstract:** In today's large complex enterprise network, security is a challenging task for most of the administrators. The typical means by which an attacker breaks into a network is through a series of exploits, where each exploit in the series satisfies the precondition for subsequent exploits and makes a causal relationship among them. Such a series of exploits constitutes an attack path and the set of all possible attack paths form an attack graph. Even the well administered networks are susceptible to such attacks as present day vulnerability scanners are only able to identify the vulnerabilities in isolation but there is a need for logical formalism and correlation among these vulnerabilities within a host or across multiple hosts to identify overall risk of the network. In this paper we propose a novel approach by map this problem in artificial intelligence domain and find out an attack path consisting of logically connected exploits, which essentially shows the minimum number of exploits required to gain access over a critical network resource. The solution is further extended to form an attack graph and find out the set of vulnerabilities which are the root cause for overall security threat towards enterprise network. The inherent time and scalability problem of attack graph generation is also taken care of in this approach. Once the set of vulnerabilities has been identified for rectification, the network administrator can then prioritize the vulnerability rectification procedure to make the network secure.

**Keywords:** attack graph, exploit, vulnerability, planner, artificial intelligence.

## 1. Introduction

As networks of hosts continue to grow in size and complexity, evaluating their vulnerability to attack become increasingly more important to automate. This makes the generation of attack graph is a classical problem for network security analysis. Construction of attack graph was first done by red team but this manual effort was tedious, error-prone, and impractical for a moderate size network. Philips and Swiler [1] developed a tool for generating attack graph. The tool constructs the attack graph by forward exploration starting from initial state using attacker's profile, configuration information of networked host and a database containing template of action. Attack templates represent generic (known or hypothesized) attacks including pre and post conditions, such as operating system version, which must hold for the attack to be possible and the privilege obtained after the attack is over. The configuration file gives detailed information about the specific system to be analyzed

including the topology of the network and configuration of particular network entity such as workstation, printer, or router. The attacker's profile contains information about the assumed attacker's capabilities, such as the possession of an automated toolkit or a sniffer as well as skill level. Ritchey and Ammann [2] propose a model checking approach which generates a counter-example against a selected security property as an attack path. The work by Sheyner et al. [3] was the first formal treatment of attack graph where they have used BDD based model checker NuSMV to compute multi-stage, multihost attack paths in a form of scenario graph. The mature logic used in their model is less error-prone and exhaustive than custom designed algorithm, but the model suffers from serious time and scalability problem. The time it took to construct an attack graph for a network with 3 hosts, 4 actions, and 6 vulnerabilities is about 5 seconds and for a network with 5 hosts, 8 actions and several vulnerabilities is about 2 hours along with 5948 nodes and 68364 edges. As a whole the complexity of this model is exponential to the number of exploits in the model. Amman et al. [4] describe an algorithm that can be used to create attack graphs through forward and backward chaining analysis to identify the necessary and sufficient exploits to reach the goal in polynomial time. The basic difference of the proposed algorithm from its previous approaches is the assumption of monotonicity. The assumption of monotonicity, states that the precondition of a given exploit is never invalidated by the successful application of another exploit. In other words, the attacker never needs to give up certain privileges in order to advance further attack. To be more general, once an attribute over the network become true from false, it will never reverse back. The assumption reduces the complexity of the problem analysis from exponential to polynomial and address the scalability issue of attack graph to some extent. Lipmann et al. [5] give a good review on the past attack graph papers and tries to find out the trade-off among these approaches. They have identified that previous works on attack graph have suffered from several substantial problems like scaling to large network, obtaining attack details, computing reachability and generating recommendations from attack graph. Jajodia et al. [6] describe the "Topological Vulnerability Analysis" (TVA) which implements an

integrated, topological approach to network vulnerability analysis using the underlying algorithm of [4]. The approach also uses a symbolic analysis technique to identify sets of actions (e.g., patching vulnerabilities, remove network services, configuration changes on hosts) that if followed, prevent the attacker from reaching goal state. Noel et al. [7] use the exploit-dependency representation of TVA [6] and represent it into symbolic equation. Analysis of this equation recommends the least cost change to be done in order to guarantee the safety of critical network resources. Noel et al. [8] describe various approaches to collapse parts of exploit-dependency attack graphs generated by the TVA [6] system to make visual understanding easier and interactive. This approach is first of its kind which identifies that “perhaps the greatest challenge in making network attack graphs practical ... is managing their visual complexity in user interaction”. Hence scalability becomes an important issue related to attack graph. This aggregation allows attack graph complexity to be compressed to an arbitrary degree, with higher levels of aggregation corresponding to higher levels of abstraction. Three basic approaches of grouping exploit dependency graph are presented in a recursive manner. At the bottom of the hierarchy, multiple exploits and their conditions between the same two hosts are aggregated into exploits sets and condition sets, respectively. Condition sets are aggregated into either machine abstractions or exploit sets. Sets of machines and the exploits among them are aggregated into machine-exploit sets or protection domains. Set of hosts or machines that are fully connected in protection domains (e.g., on a single LAN with no filtering devices) are aggregated into protection domain sets. Ritchey et al. [9] describe how network connectivity is modeled in the TVA [6] system by including the details of different layers of TCP/IP stack rather than model them as only physical link or service-port combination. This way it enables to represent the real world network security devices like firewalls, filtering routers and switches. Li et al. [10] describe a process to model system vulnerabilities and possible exploitations in homogeneous *high performance computing* (HPC) cluster environments using exploitation graphs or e-graphs. The e-graphs are further analyzed to determine attacker’s work factor analysis, cost/benefit analysis of security, detection of attacks and identification of critical vulnerabilities. Attacker’s work factor analysis identifies number of different paths in an attack graph to reach the goal. The cost/benefit analysis uses the values on each edge of the attack graph to identify minimum cost required for an attacker to reach the goal. The approach proposes to derive such values using experts view like security professionals, business management team members, or the hacker community. The approach has also used minimal vertex cut algorithm to identify critical set of vulnerabilities without which goal is not feasible. Zhang et al. [11] propose a forward-search, breadth-first and depth-limited algorithm to produce attack route and the corresponding attack graph. This approach however has all the inherent problems of breadth first search like high branching factor while applies

on an enterprise local area network. Ammann et al. [12] describe a host based algorithm for attack graph generation from a penetration tester’s perspective. The algorithm computes a suboptimal attack path among each and every pair of hosts in the network to find out maximum privilege can be gained on each host by an attacker. The approach computes the attack graph in two steps as initialization and calculation of maximal access among hosts. The initial access level has been found out using the network trust relationship table. The maximal level accesses are computed between hosts as directly and indirectly. In direct computation, if sufficient connectivity on the appropriate ports exists between two hosts, an exploitable vulnerability exists on the destination host, and the source host has all of the prerequisites for the exploit, an edge can be added from source to destination or the existing edge can be updated with maximal privilege. In indirect computation, the algorithm computes indirect access edges (i.e., transitive closure) of a directed graph that runs in  $O(n^3)$  time where  $n$  is the total number of hosts. The proposed approach further extends to leverage existing attack graph when new hosts are added to network, changing the existing trust relationship, changing the existing user credentials or giving different permissions. However this approach does not present complete information about possible damage instead provides worst case possible damage. Noel et al. [13] represent the attack graph as a form of adjacency matrix where each element represents an attack and computes higher power of the matrix for correlating, predicting, and hypothesizing about network attacks from intrusion detection system alerts. Wang et al. [15] propose interactive analysis of attack graph using relational queries. They have devised a relational model for representing network configuration and domain knowledge and generate attack graph using relational queries. Ingols et al. [16] have created a multiple-prerequisite graph that scales nearly linear as the size of a typical network increases. The entire attack graph consists of three types of nodes as state nodes, prerequisite nodes, and vulnerability instance nodes. State nodes represent an attacker’s level of access on a particular host. Outbound edges from state nodes point the prerequisites to be provided to an attacker. Prerequisite nodes represent either a reachability group or a credential. Outbound edges from prerequisite nodes point to the vulnerability instances require for successful exploitation. Vulnerability instance nodes represent a particular vulnerability on a specific port. Outbound edges from vulnerability instance nodes point to the single state that the attacker can reach by exploiting the vulnerability. The proposed attack graph uses a breadth first algorithm to create the attack graph. However the graph contains a single instance of any type of node in the entire attack graph with numerous back edges to address scalability. Ou et al. [17] uses a monotonic logic-based approach called MulVAL to produce counter-example for a given security policy over an enterprise network. The runtime complexity of the algorithm is between  $O(n^2)$  and  $O(n^3)$ . Lye et al. [18] presents a game-theoretic method for analyzing the security

of computer networks. They view the interactions between an attacker and the administrator as a two-player stochastic game and construct a model for the game to find out best response strategies.

Most of this previous approaches use an exhaustive attack graph generation methodology. Such exhaustive methodologies are exponential in nature, hence computationally expensive too for proper risk management. To overcome this problem the proposed approach uses an interactive search based methodology for attack graph generation and managing the risk.

The rest of the paper is organized as follows. Section 2 discuss about the attack graph. Section 3 describes our proposed model along with two case studies. Finally we conclude in section 4.

## 2. Attack graph

With the increasing size and complexity of today's enterprise network, attack graph becomes an essential tool for security administrators to defend their network resources. However systematic analysis of attack graph is much more important than generating it only. But till date there is no promising approach for automated visual interpretation of complex attack graph to identify recommended security rectifications. The sample attack graph for a subnet of 14 machines, with less than 10 vulnerabilities per host in [8] demonstrates how scalability becomes an important issue for attack graph over an enterprise local area network. Perhaps the greatest challenge in making attack graph analysis practical for real world network is to manage it's complexity in user interaction. Visual aggregation is an obvious way to address such problem followed by [8]. But again this kind of approach though visually reduces the sizes of the attack graph but functional complexity and size of the graph remain same. This approach would not help administrators to take proper decisions regarding which set of vulnerabilities should be rectified to make the network secure. Apart of the scalability problem, generation of attack graph in minimum time is also a growing concern for most of the administrators over a large network. To address these issues, our model has observed the problem from a security administrator's perspective while answering the question "is there any path in the network which leads to compromise the critical resources in minimal number of steps?" This essentially leads to the requirement of identifying an attack path over the specified network domain in minimum time as the attack graph consists of many such attack paths. The model also implicitly assumes that the entire network topology, vulnerability data, firewall and router rules, trust relationship etc are available, as the tool is used by a security administrator. The output attack path is represented as a form of exploit-dependent attack path where each node represents exploit along with the source host and target host and edges represent pre-condition and effects of the exploits. Once an attack path has been identified and proper remediation has been taken off, the model can go in search for another path. When no more paths exist in the network,

that instance of the network along with its configuration and hosts information, vulnerabilities, services and firewall rules are said to be secure. Hence the model tries to incorporate a dynamic risk mitigation procedure while the attack graph generation is undergone.

## 3. Proposed Approach

The model consists of two parts

- Identifying attack paths
- Generation of attack graph

### 3.1 Identifying attack paths

The first part of the model is implemented using *planner* [14] from artificial intelligence domain. *Planner* is a special purpose search algorithm for finding out a solution from initial state to goal state within a large state space. There are several advantages of using *planner* over general purpose *theorem prover* or search techniques like:

- Actions are given as logical description of pre-condition and effects. This enable *planner* to make direct connection between states and actions and prune the unnecessary actions from the system which may not help to reach the goal.
- The *planner* is free to add actions to the plan wherever they are required, rather than in an incremental way starting from the initial state. This helps in our domain by adding new exploits whenever they are published.
- The approach taken by the *planner* is most parts of the world are independent of most other parts like an attacker's goal may be to get administrator privilege on web server and evade firewall. This goal can be better solved by divide and conquer method and combine the sub-plans to generate a complete plan.

Apart of these, there are several advantages of using *planner* over custom built analysis engine like richer input language of *planner* makes it easy to express complex security conditions. There is also no logic to reinvent the wheel by putting effort for searching a new state space search techniques instead of using an existing one. *Planners* are undoubtedly fast and accurate over large state space problems like space vehicle control and in the field of robotics. Different research groups around the world are working on *planner* to make it fast and more accurate, which can be incorporated in the model to upgrade its efficiency. SGPlan 5.2.2 [19], [20] has been chosen for implementation of attack path identification component in the model. SGPlan is a classical *planner* which partitions a large planning problem into sub problems, each with its own sub goal, as each partitioned sub problem involves a substantially smaller search space than that of the original problem. The *planner* further detects reasonable orders among sub goals, an intermediate goal agenda analysis to hierarchically decompose each sub problem, a search-space-reduction algorithm to eliminate irrelevant actions in sub

problems, and a strategy to call the best *planner* like LPG-td, Metric-FF etc to solve each bottom-level sub problem. Before using SGPlan we have used several *planners* like Blackbox, LPG-td, but there are certain reasons for choosing SGPlan over other *planner* like its support for numeric predicates or fluents, derivative predicates and durative actions to represent temporal fact. The *domain* and *fact* file describes the input to the *planner* in PDDL [19], [20] (Planning Domain Definition Language) which is the de-facto standard for input specification language for *planner*. The *domain* file contains the function free ground literals or un-instantiated predicates in PDDL to represent the states of the entire network and state transition operators or exploits. One notable difference between ours and the approach used by [4] is that we have used negative predicates as a precondition and effect of exploits, removing implicit assumption of monotonicity to make the approach more realistic. The model has also followed the approach used by Ritchey et al. [9] to represent network connectivity in three distinct layers of TCP/IP stack namely application, transport and link layer rather than presenting them as a single physical connectivity only. The *fact* file represents the initial state and goal state of the system in terms of instantiated predicates like (*and* (= *has\_priv Attacker Host2*) 3)) implies attacker need root privilege on *Host2*. Appropriate changes in the *fact* file enable a *planner* to invalidate the current plan and search for a new one.

### 3.2 Generation of attack graph

The planner can efficiently find out an attack path over the specified network domain. However generating exhaustive attack graphs over a state space gives an exponential blow-up [12] so unlike model checker [3], *planner* does not find out all possible paths to the goal from initial state. Alternatively *planner* finds out an alternate path by systematic invalidation of an existing plan. The *Generate\_Attack\_Graph* algorithm of figure 1 has demonstrated how this systematic invalidation of a plan can be done based on certain criteria's. In each iteration, the algorithm finds out a single attack path and the minimum cost vulnerability along the attack path has been chosen and rectified by the administrator through necessary changes in the *fact* (*fact.pddl*) file. The algorithm then again continues searching for another attack path. This loop in the algorithm will terminate when there are no more paths in the system. The basic component of the algorithm in figure 1 is *planner*. The planner uses a graph based methodology where a *plangraph* is formed starting from the initial state and successive application of state transition operators i.e. exploits. A *plangraph* represents alternate layers of *propositions* (network states) and *actions* (exploits). A backward search on such *plangraph* identifies the actual plan. The generation of *plangraph* consumes the major amount of the time in the entire plan identification process. For example, with  $n$  number of objects,  $m$  number of operators each have maximum  $k$  number of constant formal parameters,  $l$  is the length of the longest add list of any of the operator,  $p$  is the number of propositions at the initial stage then the size of the  $t$ -level *plangraph* and its generation time will be *polynomial* as maximum generated nodes in any proposition level will be  $O(p+mln^k)$  and

maximum generated nodes in any action level will be  $O(mn^k)$  [21].

Combination of attack paths form the attack graph where for consistency the exploit dependent representation includes a distinguished *start conditions* exploit having null pre-condition and post-conditions equivalent to the initial conditions in the network model as shown in figure 4 and figure 7. Similarly *goal conditions* are handled by a distinguished *goal conditions* exploit having null post-condition and pre-conditions equivalent to the specified attack goal conditions. The exploit dependent attack path is the basic representation used in our model where each exploit's pre-condition has been satisfied by its ancestor exploits along the attack path. In an enterprise network, security administrator's are more interested to know the best way to harden their network rather than knowing about the sequence of exploits. So at the end, the algorithm will show the set of vulnerability which should be rectified to make the network secure. Rectification of a single vulnerability can be done in many forms like deployment of vendor specific patch, stop the vulnerable service, replace the service with its updated version, change the access control list of filtering devices like firewall, router, prevent certain groups of user to access the service etc. Some of these above mentioned rectification procedure needs the organization's security policy, which is a concise list of what is allowed and what is not allowed, to be reassessed. Though there is no well-defined method to quantitatively express vulnerability rectification cost, it is assumed that installing a patch provides less overhead than stopping a service or change firewall's access control list. So the cost of rectification can be expressed in terms of usability requirement i.e. whether stopping or removing a vulnerable service conforms to organization's security policy or not. For example to prevent *ftp writable home directory* vulnerability on remote host requires either to stop the service or change its configuration such that remote users can not access the service whereas rectify a *buffer overflow vulnerability* in a *ssh* daemon, requires only to deploy a vendor specific patch for rectification. As a result different vulnerability rectification incurs different cost towards organization's security budget which in turn leads to an administrator's decision support system to find out least cost vulnerability to invalidate an attack path. In average case this approach might require  $O(n)$  time complexity where  $n$  is the number of exploits in an attack path with an implicit assumption of making decision regarding each vulnerability rectification cost is constant.

Unlike other approaches [3], our approach does not present a large, complex attack graph with redundant paths before administrator, which is not possible to visually interpret and recommend for necessary security rectifications. Our approach also seems realistic compare to network hardening [6], [7] approach as they are using a recursive algebraic substitution of dependency terms in a backward direction to represent a goal state in terms of initial conditions, which they have admitted can potentially grow exponentially with the number of initial conditions, is not suited for a large enterprise network. There approach

presents an exponential time bound algorithm in support of their view which can execute in backward direction starting from goal state to identify minimal cost maxterms for the consideration which seems to be closer of choosing least cost rectification vulnerability followed in our approach. The attack graph generated by our model is highly dependent on administrator's decision support system and different attack graphs can be generated depending upon different decision taken up by the security administrator. This leads to a what-if analysis approach in the specified network domain. There are many attack paths in the attack graph which have a common set of exploits among them, but may not be identified during individual rectification procedure as the model follows an unsupervised learning approach to construct attack graph.

### Algorithm Generate\_Attack\_Graph

**Input:** domain and fact file containing network description.

**Output:** Attack Graph and rectified vulnerability set or an error message.

**Steps:**

1. `attack_path=PLANNER(domain.pddl, fact.pddl)`
2. **If** (`attack_path==NULL`)
  1. Print "No path exists to the critical resource".
  2. **Exit**.
3. **EndIf**
4. **While** (`attack_path ≠ NULL`) **do**
  1. Find the set of vulnerability corresponding to each exploit along the attack paths.
  2. Choose the vulnerability with Minimum network hardening cost and rectify it.
  3. Store the rectified vulnerability in the rectified vulnerability set.
  4. Do the necessary changes in `fact.pddl`.
  5. `attack_path= PLANNER (domain.pddl, fact.pddl)`.
5. **EndWhile**
6. Merge Individual Attack Paths to generate attack graph.
7. Print "Rectified vulnerability set ".

8. **Stop**.

**Figure 1.** Attack graph generation algorithm

### 3.3 Case study

In the following subsection we demonstrate with the help of two case studies how the model can efficiently carry out an attack graph generation and rectification procedure compared to previous approaches [2], [3], [4], [6], [7], [8] over real world network.

#### 3.3.1 Case study 1

For case study 1, we have collected the data from a previous research paper [3]. The example network is shown in figure 2. There are two hosts on the internal network, *Host1* and *Host2*, and the firewall separating the internal network from external network. The attacker's host is *Host0* on the external network. The host information on the internal network is shown as Table 1.

Host	Services	Vulnerabilities	OS
<b>Host1</b>	FTP, SSH	sshd buffer overflow, ftp .rhost overwrite	Linux
<b>Host2</b>	FTP, XTERM, DATABASE,	ftp.rhost overwrite, local xterm buffer overflow	Linux

*Table 1.* Host description

The firewall allows the inbound *ftp* and the *ssh* packets to communicate with the *Host1* and *Host2*, but interdicts other packets. In the internal network, connection relation won't be controlled by firewall, so it can be assumed that the internal host can make connection with any remote server. The connection relation among each other is described in the following Table 2.

Relation	Host0	Host1	Host2
<b>Host0</b>	Local host	FTP, SSH	FTP
<b>Host1</b>	Any	Local host	FTP
<b>Host2</b>	any	FTP	Local host

*Table 2.* Connection description

There are four possible atomic attacks which can be numerically denoted as follows: *sshd buffer overflow (att0)*, *ftp\_rhosts (att1)*, *remote login (att2)* and *local buffer overflow (att3)*. Each of these attacks can be described as follows:

1. *sshd buffer overflow*: This attack immediately gives a *root* shell on the victim machine to the remote user. It has detectable and stealthy variants.

2. *ftp\_rhosts*: Using an *ftp* vulnerability, the intruder creates an *.rhosts* file in the *ftp* home directory, creating a remote login trust relationship between his machine and the target machine. This attack is stealthy.

3. *remote login*: Using an existing remote login trust relationship between two machines, the intruder logs into from one machine to another and get a user shell without supplying a password. This attack is detectable.

4. *local buffer overflow*: Exploiting a *buffer overflow* vulnerability on a *setuid root* file gives attacker *root* access on a local machine. This attack is stealthy.

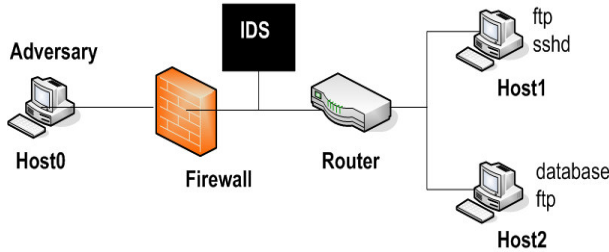


Figure 2. Network diagram for case study 1

The intruder launches his attack starting from a single computer *Host0* which lies outside the firewall. The attacker’s eventual goal is to disrupt the functioning of the database on *Host2* for which he needs *root* privilege on *Host2*. While generating solution for this problem, *model checker* took 5 seconds to find out all the paths and outputs a large complex attack graph with lots of repetition of same sets of actions in different sequence. The only difference between the two attack paths shown in figure 3(in bold line) are their order of occurrence of *sshd buffer overflow* (*att0*) and *ftp\_rhosts* (*att1*) from *Host0* to *Host1*. However none of these two exploit depends on each other by their pre or post condition under given network specification. Hence in reality they can be executed in any order. The attack path *att0(0,1)->att1(1,2)->att2(1,2)->att3(2,2)* also appears twice in the graph presented in figure 3. Each node in figure 3 and figure 4 represents an exploit along with 0, 1, or 2 to denote *Host0*, *Host1*, and *Host2* of figure 2 respectively. Actually the problem has only three paths to the goal as shown in figure 4. The model’s objective is to find these actual paths one by one in a timely efficient manner and also identify the necessary reification to be done. The *domain* file of our model contains the *predicates* to represent network and host configuration, connectivity, related vulnerability, user privilege, user level trusts and possible set of exploits. For example (*ftp ?H*) will be true if *Host H* is running *ftp* service and (*user ?A ?H*) will be true if attacker *A* has *user* privilege on *Host H*. It also uses *numeric literals* or *fluent* to represent the privilege level of user. The model has assumed a total ordering of user privilege in the form of *none<user<root* and represent them in *fact.pddl* file as follows:

```
(= (root_priv) 3)
(= (user_priv) 2)
(= (none_priv) 1)
```

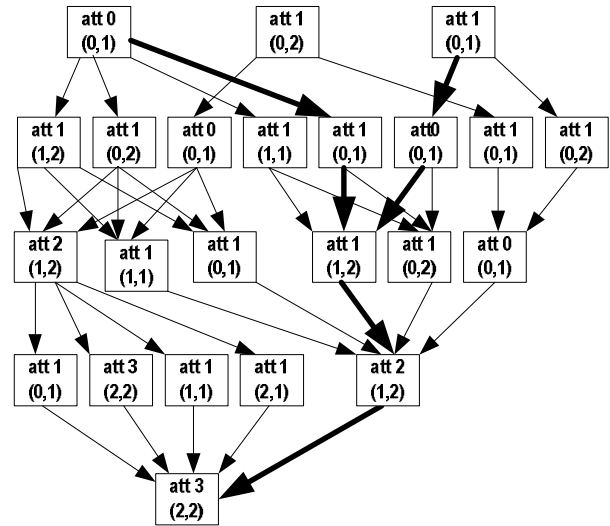


Figure 3. Attack graph generated by model checker [3]

The *fact* file also uses *derivative predicates* to represent scenarios like attacker has a transport layer connectivity to any particular host and that host running *ftp* service implies attacker can access the *ftp* service on that host as follows:

```
(:derived (ftp_port_connectivity ?X ?Y) (and
(trans_port_connectivity ?X ?Y)(ftp?Y)))
```

The model also incorporates negative predicate like (*not (ssh ?h)*) as pre and post-condition of exploits to remove the implicit assumption of monotonicity and helps to model non monotonic attacks like *denial-of-service* (DOS). The *fact.pddl* file contains the initial and goal state of the attacker as *root\_priv(attacker, Host0)* which implies attacker has *root* privilege on *Host0*, (*and(= (has\_priv Attacker Host2) 3)*) implies that attacker’s goal is to get *root* privilege on *Host2*. Putting all these data as input, the *Generate\_Attack\_Graph* algorithm has been executed, which generates an attack path within *0.01 seconds*.

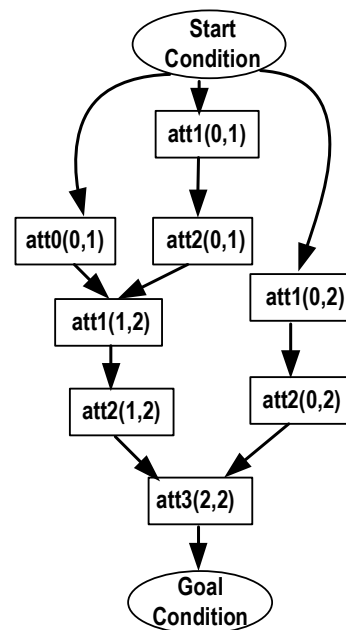


Figure 4. Attack graph generated by our model

Analyzing *attack path1* it has been decided to stop access to *ftp* service on *Host2* for outside users and the *fact* (*fact.pddl*) file has been changed by commenting the line containing (*ftp\_apps\_connectivity Host0 Host2*). Subsequent call to the algorithm gives the *attack path 2*. Again after careful examination of *attack path 2*, deployment of a vendor specific patch for the *ssh* service on *Host1* has been prioritized over the other options like stopping the *ftp* service between *Host 1* and *Host 2*. To invalidate *attack path 3* stopping the *ftp* service access on *Host1* for outside users have been chosen as least cost method. Further execution of the algorithm does not generate any new attack path rather gives the rectified set of vulnerabilities as (*ftp\_rhosts (Host2), sshd\_bof(Host1), ftp\_rhosts (Host1)*). This intuitively indicates that the entire path to the goal has been traced out and merging all this attack paths generates the required attack graph in figure 4. The individual attack paths are shown in figure 5. The total time taken to find out three paths is  $(0.01+0.01+0.01) = 0.03$  seconds compare to 5 seconds [3] taken by the *model checker* approach with added advantage of scalability and generation of network hardening recommendations. Once the attack graph of figure 4 has been generated, the administrator can then decide which set of vulnerabilities will be rectified to make the network secure depending upon their relative rectification cost.

Attack Path1	Attack Path2	Attack Path3
1.(FTP-RHOST ATTACKER HOST0 HOST2)	1.(SSH_BUFFEROVERFLOW ATTACKER HOST0 HOST1)	1.(FTP-RHOST ATTACKER HOST0 HOST1)
2.(RSH-LOGIN ATTACKER HOST0 HOST2)	2.(FTP-RHOST ATTACKER HOST1 HOST2)	2.(RSH-LOGIN ATTACKER HOST0 HOST1)
3.(SET_UID_BUFFER_OVERFLOW ATTACKER HOST2 HOST2)	3.(RSH-LOGIN ATTACKER HOST1 HOST2)	3.(FTP-RHOST ATTACKER HOST1 HOST2)
	4.(SET_UID_BUFFER_OVERFLOW ATTACKER HOST2 HOST2)	4.(RSH-LOGIN ATTACKER HOST1 HOST2)
		5.(SET_UID_BUFFER_OVERFLOW ATTACKER HOST2 HOST2)

**Figure 5.** Attack paths for different network configuration of case study 1

### 3.3.2 Case study 2

In case study 2 we have analyzed the attack graph shown by “Topological Vulnerability Analysis” (TVA) [6] approach. The network diagram is shown in figure 6. The details of the network configuration, vulnerabilities are described in [6]. The exploits can be described as follows:

- iis rds*: The attack requires connectivity to the *iis* service on the victim and successful execution of the attack gives attacker *root* privilege on the victim system.
- rcpdownload*: The attack requires connectivity to the *rsh* service on the victim and successful execution of attack helps to copy program on victim system from attacker’s system.
- portforward*: The attack requires execute access on middleman, portforwarding program on middleman, attacker’s connectivity to transport-layer (unused) port on middleman and provides transport layer connectivity to victim through middleman.
- wuftpdx*: This attack requires connectivity to the *ftp* service on the victim machine and provides *root* privilege on victim.

The *TVA* approach uses forward and backward chaining analysis based on the algorithm described by [4] along with implicit assumption of monotonicity to generate exploit-dependent attack graph. Their approach scales better than [3] and also closely resembles to our approach as it gives only necessary and sufficient attack paths to the critical resource. The figure 7 represents the attack graph generate by *TVA* where *S*, *M* and *N* denotes attacker’s host *start*, *maude* and *ned* and the exploits are denoted as follows *iis rds* (*att 0*), *rcpdownload* (*att1*), *portforward* (*att 2*), *wuftpdx* (*att 3*). Different attack paths can be used by the attacker at different network conditions which is not apparent from the attack graph representation by *TVA*. Under the described network configuration, our model is able to find out *attack path 1* as shown in figure 8 when there is already an existing transport level *ftp* connectivity between host *maude* and *ned*. According to *attack path 1*, initial execution of *iis rds* exploit enables the attacker to execute programs on *maude*. Given the access provided by the *iis rds* exploit, the *rcpdownload* program on *maude* is executed to download a *rootkit* from the attacker’s machine. A *portforward* exploit from the *rootkit* is then executed to set up access from the attack machine through *maude* to the *ftp* service on *ned*. Finally, the *wuftpdx* exploit is executed through the forwarded connection against *ned* to obtain *root* privilege on it. After analyzing *attack path 1* it has been decided to block *ftp* access from host *maude* to *ned* based on *maude*’s *ip* address as it requires less overhead than changing the configuration for *iis* web server or *wuftp* daemon. Further execution of the algorithm finds *attack path 2* as shown in figure 8. In *attack path 2* despite of closing the *ftp* connection between host *maude* and *ned*, the attacker is able to download a *portforward* exploit from host *start* and establish a *ftp* connectivity between host *maude* and *ned* to

achieve the goal. To invalidate *attack path 2* there are certain options before administrator as:

1. Patch the *iis rds* web server on host *maude*.
2. Prevent attacker from downloading *portforward* exploit.
3. Remove all the unused port from *maude*.
4. Patch the *wuftp* daemon on host *ned*.

Selection of any of the above option invalidates *attack path 2* and the resultant attack graph is shown in figure 7. However if option 1 or 2 has been selected by the administrator then the rectified vulnerability set will be redundant as prevention of *iis rds* exploit or download of *portforward* exploit will invalidates all the path in the graph and eradicates the necessity of closing the *ftp* port connectivity between host *maude* and *ned*. The *planner* took only *0.02 seconds* to identify each of these attack paths.

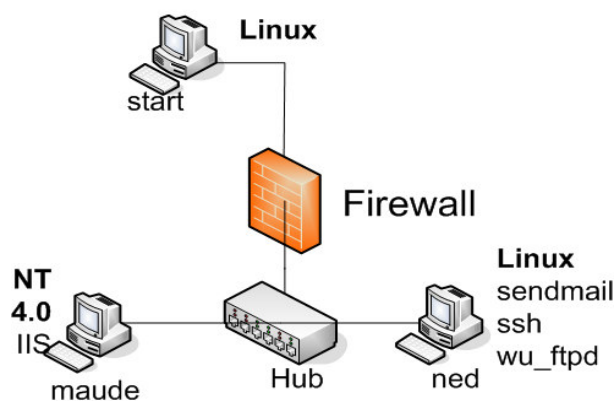


Figure 6. Network diagram for case study 2

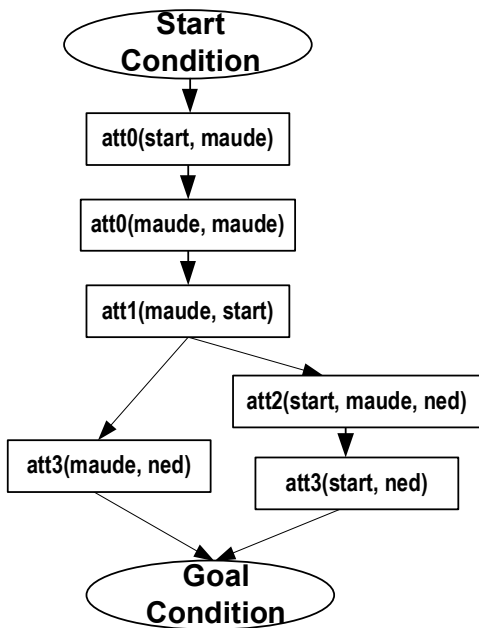


Figure 7. Attack graph for case study 2

Attack Path1 | Attack Path2

1.(IISRDS_BUFFER_OVERFLOW ADVERSARY START MAUDE)	1.(IISRDS_BUFFER_OVERFLOW ADVERSARY START MAUDE)
2.(IISRDS_BUFFER_OVERFLOW ADVERSARY MAUDE MAUDE)	2.(IISRDS_BUFFER_OVERFLOW ADVERSARY MAUDE MAUDE)
3.(RCPDOWNLOAD ADVERSARY MAUDE START)	3.(RCPDOWNLOAD ADVERSARY MAUDE START)
4.(WUFTPDX ADVERSARY MAUDE NED)	4.(PORTFORWARD ADVERSARY START MAUDE NED)
	5.(WUFTPDX ADVERSARY START NED)

Figure 8. Attack paths for different network configuration of case study 2

### 3.3.3 Discussion

The analysis of an attack graph is primarily constrained by time and scalability issues. In our proposed approach, we have demonstrated that how time and scalability problem can be managed while generating the attack graph with the help of two case studies. While in case study 1 we have shown how the scalability issue can be addressed by pruning the redundant paths from the system in near optimal time, in case study 2 we represent how low level attack details like port-forwarding can be modeled in our approach. The model also shows an alternative approach about how minimum security rectification can be found out compare to the approach presented by [7], which has an exponential time complexity with the number of increment in initial conditions.

## 4. Conclusion

In this paper we have proposed a classical *planner* based approach for attack graph generation. Unlike previous approaches, we have tried to identify the recommended vulnerability rectifications while the attack graph is under development. This approach helps us to build a scalable attack graphs in minimal time. In the proposed approach, hand coding of the domain information is required, which may be tedious for an enterprise network. However, once this domain specification has been encoded, the rest of the procedure can work in near optimal time. Our model can also be further extended to incorporate different level of attacker’s capability, such as script kiddies, hacker etc, to identify most probable critical path in the network with the help of probabilistic or contingency planning.

## 5. References

[1] C. Phillips, L. P. Swiler. “A graph-based system for network-vulnerability analysis”. In *Proceedings of the 1998 workshop on New security paradigms (NSPW)*, pp. 71-79, 1998.

- [2] R. Ritchey, P. Amman. "Using Model Checking to Analyze Network Vulnerabilities". In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pp. 156-165, 2000.
- [3] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. M. Wing. "Automated generation and analysis of attack graphs". In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 254-265, 2002.
- [4] P. Ammann, D. Wijesekera, S. Kaushik. "Scalable, graph-based network vulnerability analysis". In *Proceedings of the 9<sup>th</sup> ACM Conference on Computer and Communications Security (CCS)*, pp. 217-224, 2002.
- [5] R. Lippmann, K. Ingols. "An annotated review of past papers on attack graphs". *Lincoln Laboratory report*, MIT, USA, 2005.
- [6] S. Jajodia, S. Noel, B. O'Berry. "Topological analysis of network attack vulnerability", in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, and A. Lazarevic (eds.), Springer, Germany, 2003.
- [7] S. Noel, S. Jajodia, B. O'Berry, M. Jacobs. "Efficient minimum-cost network hardening via exploit dependency graphs". In *Proceedings of the 19<sup>th</sup> Annual Computer Security Applications Conference (ACSAC)*, pp. 86-95, 2003.
- [8] S. Noel, S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation". In *Proceedings of the ACM workshop on visualization and data mining for computer security*, pp. 109-118, 2004.
- [9] R. Ritchey, B. O'Berry, S. Noel. "Representing TCP/IP Connectivity for Topological Analysis of Network Security". In *Proceedings of the 18<sup>th</sup> Annual Computer Security Applications Conference (ACSAC)*, pp. 25-31, 2002.
- [10] W. Li, R. B. Vaughn. "Cluster Security Research Involving the Modeling of Network Exploitations Using Exploitation Graphs". In *Proceedings of the 6<sup>th</sup> IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pp. 26-37, 2006.
- [11] T. Zhang, M. Hu, D. Li, L. Sun. "An effective method to generate attack graph". In *Proceedings of the International Conference on Machine Learning and Cybernetics*, pp. 3926- 3931, 2005.
- [12] P. Ammann, J. Pamula, R. Ritchey, J. Street. "A host based approach to network attack chaining analysis". In *Proceedings of the 21<sup>st</sup> Annual Computer Security Applications Conference (ACSAC)*, pp.10, 2005.
- [13] S. Noel, S. Jajodia. "Understanding Complex network attack graphs through cluster adjacency matrix". In *Proceedings of the 21<sup>st</sup> Annual Computer Security Applications Conference (ACSAC)*, pp. 72-84, 2005.
- [14] S. J. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 1995.
- [15] L. Wang, C. Yao, A. Singhal, S. Jajodia. "Interactive Analysis of Attack Graphs Using Relational Queries". In *Proceedings of the 20<sup>th</sup> IFIP WG 11.3 Working Conference on Data and Applications Security*, pp. 119-132, 2006.
- [16] K. Ingols, R. Lippmann, K. Piowarski. "Practical Attack Graph Generation for Network Defense". In *Proceedings of the 22<sup>nd</sup> Annual Computer Security Applications Conference (ACSAC)*, pp. 121-130, 2006.
- [17] X. Ou, W. F. Boyer, M. A. McQueen. "A Scalable Approach to Attack Graph Generation". In *Proceedings of the 13<sup>th</sup> ACM conference on Computer and communications security (CCS)*, pp. 336 - 345, 2006.
- [18] K.-W. Lye, J. Wing. "Game strategies in network security". In *Proceedings of the Foundations of Computer Security Workshop*, 2002.
- [19] Y. Chen, C. Hsu, B. Wah. "Temporal Planning using Subgoal Partitioning and Resolution in SGPlan", *Journal of Artificial Intelligence Research*, 26, pp. 323-369, 2006.
- [20] C. W. Hsu, B. W. Wah, R. Huang, Y. X. Chen. "New Features in SGPlan for Handling Soft Constraints and Goal Preferences in PDDL3.0". In *Proceedings of the 5<sup>th</sup> International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 39-42, 2006.
- [21] A. Blum, M. Furst. "Fast Planning Through Planning Graph Analysis", *Artificial Intelligence*, 90, pp. 281-300, 1997.

## Author Biographies



Somak Bhattacharya, born at West Bengal, India, is presently pursuing MS in Information Technology from Indian Institute of Technology, Kharagpur. His main area of research includes network and system security.



S.K. Ghosh, PhD, born at West Bengal, India, is presently working as Assistant Professor in the School of Information Technology, Indian Institute of Technology, Kharagpur, India. Prior to IIT Kharagpur, he worked for Indian Space Research Organization (ISRO), Department of Space, Government of India, for about eight years in the field of Satellite Remote Sensing and GIS. His research interest includes

Information Security, Remote Sensing & GIS, and Image Processing. He is an IEEE Member.