

Topological Analysis of Multi-phase Attacks Using Expert Systems*

HAMID REZA SHAHRIARI, YASSER GANJISAFFAR, RASOOL JALILI
AND JAFAR HABIBI
*Department of Computer Engineering
Sharif University of Technology
Tehran, Iran*

With the increasing number and complexity of network attacks, the demand for automatic vulnerability analysis tools has increased. The prerequisite of making these tools is to have a formal and precise model of network configurations and vulnerabilities. Utilizing this model, network administrators can analyze the effects of vulnerabilities on the network and complex attack scenarios can be detected before happening. In this paper, we present a general logic-based framework for modeling network configurations and topologies. Then, a number of important and wide-spread network vulnerabilities are modeled as general inference rules based on the framework definitions. We implemented the approach using an expert system to analyze network configurations and detect how an attacker may exploit chain of vulnerabilities to reach his goal. Our approach explores all attacking paths and generates the closure of access rights that the attacker can gain by exploiting the vulnerabilities. The time complexity of calculating the closure is polynomial. Having the closure, we can test if a user has a special right over a resource in just $O(1)$ time complexity. Moreover, the firewall filtering rules can be modeled and analyzed to determine the initial accesses in the network. Our framework is more flexible than previous ones, as it can model some major parts of Denial of Service (DoS) attacks and infer about network topology. Finally, a case study is also presented to explore the model applicability and show its efficiency and flexibility.

Keywords: computer security, vulnerability model, vulnerability analysis, topological analysis, attack, expert system

1. INTRODUCTION

The increasing growth of computer networks has increased the demand for automatic network vulnerability analysis tools. A large network consists of a great number of hardware and software platforms and connects many hosts to each other. In such networks, there would be subtle attack scenarios which can not be detected even by security experts. Complex, multi-phased attacks such as Mitnick attack [1] show how complex attack scenarios can remain hidden from the eyes of system designers and network administrators.

In order to prevent occurrence of intrusions and attacks, network administrators need to know attack scenarios that may happen in their networks. Many host scanning tools have been offered during recent years for scanning individual hosts to detect their vulnerabilities; while in many attack scenarios, attackers exploit vulnerabilities of more

Received May 1, 2006; revised August 1, 2006; accepted January 23, 2007.

Communicated by Tzong-Chen Wu.

* This research was partially supported by Iran Telecommunication Research Center (ITRC) by contract No. 500/419.

than one host to reach their goals. Therefore, automatic multi-host vulnerability analysis is required, which was done in the past manually by Red teams [2].

Some approaches have been proposed in the literature to analyze network vulnerabilities considering the relations between individual hosts and network configurations [3-12]. These approaches mainly use model checking and graph-based techniques to generate and analyze an attack graph and typically need an exponential time for execution. In [6, 13] polynomial time approaches have been suggested for the same problem. But in their approach neither interruption attacks nor network topology has been considered.

Expert systems are computer programs that are used to solve problems in domains that ordinarily require human expertise. This goal is usually achieved by combining a logical inference engine with a knowledge base. The information in the knowledge base contains a set of known facts and a set of production rules that allow if-then inferences on the facts and other acquired information [14]. Expert systems have been applied in a wide range of industrial and commercial applications. Perhaps the most prominent application of expert systems in network security has been in the field of intrusion detection [15-19].

In this paper, we propose a general framework for modeling network topologies, configurations, and vulnerabilities; and then we will show how an expert system can be implemented based on the framework for automating the process of multi-host vulnerability analysis. Our approach explores all attacking paths and generates the closure of access rights that an attacker can gain. Having the closure, we can detect in $O(1)$ time complexity whether a user has an especial right over a resource or not. The approach is efficient as well as flexible enough to address most of the wide-spread vulnerabilities and attack methods. Additionally the DoS vulnerabilities which may interrupt a service or access can be represented and analyzed in the framework, which is a new contribution over previous works. Moreover, in the proposed framework, the topology of the network can be represented and analyzed to obtain the initial accesses of principals in the network. In our approach, the firewall filtering rules are modeled and analyzed in this regard.

The remaining of this paper is organized as follows. Section 2 introduces the formal framework for modeling network configurations and vulnerabilities and the inference rules to analyze them. In section 3, we show how the framework can be used to find multi-host attack scenarios on a sample network. In section 4, the process of building a rule-based expert system based on this framework is discussed and the experimental results for evaluating its efficiency are presented. In section 5, related work is discussed and compared with our approach. Section 6 addresses the advantages and limitations of the approach. Finally, the last section underlines some concluding regards and future research lines.

2. VULNERABILITY ANALYSIS FRAMEWORK

In this section, we propose a formal framework for network vulnerability analysis. We first show how we can describe the topology of a typical network and then we represent the vulnerabilities based on their preconditions and post-conditions in an expert system.

2.1 Network Model

In our framework, we represent the network in an abstract specification. The network is partitioned into *Zones*, which are connected to each other through *Gateways*. Gateways are packet routing and filtering devices that can be either *firewalls* or *routers*. A router does not filter the packets passing it while a firewall can filter packets using rules embedded in it. A *Zone* is a set of hosts that can communicate with each other directly and packets that are sent between any two of these hosts do not need to pass through any gateway. In typical networks, zones correspond to a corporation's subnets. In our model, we consider one large Internet zone, that contains hosts that are not in the corporation's network.

Formally speaking, we define a network using the following definition:

Definition 1 Network: A network is defined as a tuple $NET = (HS, ZN, GW, CN)$, where:

- $HS = \{h_1, \dots, h_m\}$ is the set of hosts in the network.
- $ZN = \{z_1, \dots, z_n\}$ is the set of zones in the network, where z_i is a set of hosts and $z_i \cap z_j = \emptyset$ for all $i \neq j$.
- $GW = \{g_1, \dots, g_k\}$ is the set of gateways.
- $CN = \{(z_i, g_j) \mid z_i \in ZN, g_j \in GW\}$ is the connection relationship which $(z_i, g_j) \in CN$ shows that the gateway g_j is connected to zone z_i .

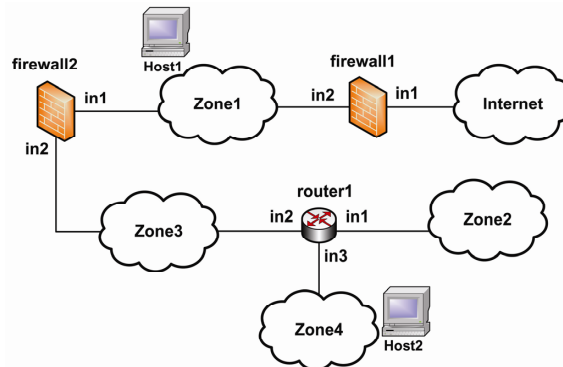


Fig. 1. A sample network.

Fig. 1 shows a sample network that is divided into five zones (four local zones and an internet zone) by firewalls and routers. In this figure, firewalls control the passing traffic between their interfaces. *Host1* and *Host2* reside in *Zone1* and *Zone4* respectively.

There are some propositions that are used for modeling network topology, as follows:

- $InZone(z, h)$: host h is located in zone z .
- $ZoneAddressInInterface(gw, in, z)$: The address space of zone z is in the address space of the in interface of gateway gw . Therefore, the routing table of gateway gw causes packets that have host h as their destination to be forwarded to its in interface.

- *BetweenZones*(gw, z, z'): Gateway gw is located between zones z and z' and any packet that wants to travel from one of these zones to the other must pass through gateway gw .

The last proposition can be generated automatically using the following inference rules:

$$\frac{\text{ZoneAddressInInterface}(gw, in, z) \quad \text{ZoneAddressInInterface}(gw, in', z')}{\text{BetweenZones}(gw, z, z')} \quad (1)$$

$$\frac{\text{BetweenZones}(gw, z_1, z_2) \quad \text{BetweenZones}(gw', z_2, z_3)}{\text{BetweenZones}(gw, z_1, z_3), \text{BetweenZones}(gw', z_1, z_3)} \quad (2)$$

In network attacks, the attackers usually try to find the open ports and exploit any possible vulnerability. Let PR be the set of all ports, then the relation $CanConnect \subseteq HS \times HS \times PR$ is used to show the connectivity relation among hosts. $CanConnect(h, h', p)$ means that the network topology and configuration let host h connect to port p of host h' .

There are some other general facts to describe the configuration of the network:

- *Account*(a, h): The user account a is defined on host h .
- *AccountOwner*(a, h, A): The user A is the legal owner of the user account a on host h .
- *Priv*(u, a, h): User u has privileges of user account a on host h .
- *Service*(s, p, a, h): Host h offers service s on its port p and this service runs by privileges of the user account a .
- *CanReadConfidentialData*($a, h, path$): Users who have privileges of account a on host h can read the contents of file or directory which is referenced by $path$ and contains confidential data.
- *CanModifyData*($a, h, path$): Users who have privileges of account a on host h can modify the contents of file or directory which is located at $path$.
- *NetworkAccess*(u, h, p): User u can use the network to access to port p of host h .

The last proposition can be deduced from previously defined propositions using the following inference rule:

$$\frac{\text{CanConnect}(h, h', p) \quad \text{Priv}(u, a, h)}{\text{NetworkAccess}(u, h', p)} \quad (3)$$

This rule states that if host h can connect to port p of host h' , then any user who has a user account on host h can use this connection and get access to that port.

Another inference rule that must be added to the model is the *Root Privileges* rule. When a user has access to the *root* account on a host, he can also access to other accounts on that host. So the following rule can also be added to our inference rules:

$$\text{Priv}(u, \text{root}, h) \wedge \text{Account}(a, h) \Rightarrow \text{Priv}(u, a, h). \quad (4)$$

2.1.1 Modeling firewall rules

Another aspect of the network that must be modeled is the behavior of firewalls on packets that pass through them. In order to model a firewall behavior, the rules defined for determining its behavior should be modeled. The following propositions are used for this purpose:

- $\text{FirewallRule}(fw, SH, DH, DP, a, pr)$: This fact defines a filtering rule in firewall fw for packets with source host in SH and destination host and port in DH and DP . Suppose $sh \in SH$, $dp \in DP$, and $dh \in DH$, then this rule causes action a to be done on accesses from host sh to port dp of host dh , providing that there is no other rule with higher priority matched with the packet. The action a can be either permit or deny and the priority of this rule is PR . The reason for using priorities in this proposition is that a request may match in more than one rule. In this case, a conflict resolution technique must be used for selecting only one rule. The priority of rules can be used for this purpose and the rule with the lowest value for priority is selected.
- $\text{PacketCanPass}(fw, h, h', p)$: The configuration of firewall fw permits packets with source host h , destination host h' , and destination port p to pass through this firewall.

The second proposition need not be declared manually and it can be derived automatically by analyzing firewall rules as below:

$$\frac{\begin{array}{l} \text{FirewallRule}(fw, H', H, P, \text{permit}, pr) \bullet \\ ((h' \in H') \wedge (h \in H) \wedge (p \in P)) \\ \exists \text{FirewallRule}(fw, H'_2, H_2, P_2, \text{deny}, pr') \bullet \\ ((pr' < pr) \wedge (h' \in H'_2) \wedge (h \in H_2) \wedge (p \in P_2)) \end{array}}{\text{PacketCanPass}(fw, h', h, p)}. \quad (5)$$

2.1.2 Modeling routers

In order to make the model simpler, we model routers as firewalls with the following rule defined on them:

$$\text{FirewallRule}(\text{router}, \text{AnyHost}, \text{AnyHost}, \text{AnyPort}, \text{permit}, 0)$$

This rule states that routers can be modeled as firewalls that permit all the packets to pass through them. AnyHost and AnyPort can match with any set of host and ports respectively.

2.1.3 Analyzing network configuration

Now suppose that we want to analyze a network configuration that is modeled using the propositions introduced above and intend to determine if host h can connect to port p of host h' (i.e., whether $\text{CanConnect}(h, h', p)$ is true or false). There are two cases: h and

h' are located in the same zone or they are located in different zones. If these two hosts be located in the same zone, then no filtering device is located between them and therefore they can access each other without any restriction. In this case, if one of the hosts offer a service on one of its ports then the other one can connect to this port and use the service. The following inference rule shows this:

$$\frac{\begin{array}{l} \text{Service}(s, p, a, h') \\ \text{InZone}(Z, h) \\ \text{InZone}(Z, h') \end{array}}{\text{CanConnect}(h, h', p)}. \quad (6)$$

In the other case, the hosts are located in different zones. Thus, the path between them must be analyzed to determine if packets that are sent from the first host can pass through *all of the firewalls and routers* in the path and reach to the second host:

$$\frac{\begin{array}{l} \text{Service}(s, p, a, h) \\ \text{InZone}(z, h) \\ \text{InZone}(z', h') \\ z \neq z' \\ \forall \text{BetweenZones}(fw, z, z') \bullet \text{PacketCanPass}(fw, h', h, p) \end{array}}{\text{CanConnect}(h', h, p)}. \quad (7)$$

For example, suppose that we want to determine whether *Host1* can connect to port *Port1* of *Host2* on the network of Fig. 1. After modeling the topology and configurations of this network, we will have the following facts in the list of initial known facts:

- $\text{ZoneAddressInInterface}(\text{firewall2}, \text{in1}, \text{Zone1})$
- $\text{ZoneAddressInInterface}(\text{firewall2}, \text{in2}, \text{Zone3})$
- $\text{ZoneAddressInInterface}(\text{router1}, \text{in2}, \text{Zone3})$
- $\text{ZoneAddressInInterface}(\text{router1}, \text{in3}, \text{Zone4})$

The inference engine can use rules 1 and 2, and deduce the following new facts:

- $\text{BetweenZones}(\text{firewall2}, \text{Zone1}, \text{Zone3})$
- $\text{BetweenZones}(\text{router1}, \text{Zone3}, \text{Zone4})$

These two facts state that *firewall2* is located between zones *Zone1* and *Zone3*, and *router1* is located between zones *Zone3* and *Zone4*. Since host *Host1* is in zone *Zone1* and host *Host2* is in zone *Zone4*, it can be deduced that a packet from *Host1* to *Host2* must pass through *firewall2* and *router1*. The inference engine can use rule (5) for analyzing firewall rules and determining if $\text{PacketCanPass}(\text{firewall1}, \text{Host1}, \text{Host2}, \text{Port1})$ and $\text{PacketCanPass}(\text{router1}, \text{Host1}, \text{Host2}, \text{Port1})$ are true and then it can use rule (7) to determine if $\text{CanConnect}(\text{Host1}, \text{Host2}, \text{Port1})$ is true.

In summary, for checking whether one host can connect to a specific port of another host, the inference engine must first find the firewalls that reside on the path between

these two hosts and then it must analyze the rules that are defined in these firewalls to check whether these rules let the packets from the source host go to the destination host.

2.2 Modeling Vulnerabilities

Hosts in a network may be vulnerable such that an attacker is able to compromise them and intrude the systems. Such vulnerabilities can be in different levels including the operating system, application programs or in the layers of network protocol stack. Attackers increase their privileges via exploiting the vulnerabilities to reach their goals in the network, which is typically accessing to a special resource. Accordingly, exploiting the vulnerabilities can be modeled as *change in privileges*. The change can be an increase in attacker privileges or a decrease in the other users' privileges. The attacker may acquire permissions on resources that may lead to violation of security aspects, namely confidentiality, integrity or availability. In the cases where the attacker prevents the users from accessing the services, the availability aspect of security is violated. Therefore, the privileges defined in the system can be changed through vulnerabilities. These issues are used to model and analyze the network vulnerabilities.

Vulnerabilities are modeled using their preconditions and post-conditions. To exploit a vulnerability, its preconditions must be satisfied. Preconditions are any condition based on the network configuration and users' privileges and include (but not limited to) network configuration, current privileges of the attacker, and version of services and applications running on the hosts. Post-conditions usually increase the attacker privileges, but other changes in the system configurations (*e.g.* change of the firewall configuration) are also possible. For example, a DoS attack can stop a service or change in the hosts' connections.

In the reminder of this section, we propose a method for modeling network vulnerabilities using their pre-conditions and post-conditions. The following template is used for this purpose:

<p style="text-align: center;"><i>Vulnerability name</i></p> <p>Preconditions:</p> <p style="text-align: center;">$p_1 \wedge p_2 \wedge \dots \wedge p_n$</p> <p>Postconditions:</p> <p style="text-align: center;">$p'_1 \wedge p'_2 \wedge \dots \wedge p'_m$</p>

In order to be able to model vulnerabilities effectively, the following classification for vulnerabilities is used: *Service vulnerability*, *Local program vulnerability*, *Denial of service*, *Weak password*, *Confidentiality loss*, *Integrity loss*.

In this work, we do not want to analyze network vulnerabilities which are in individual hosts and only their existence is stated with simple propositions. For example, *ServiceVuln(s, h)* is used to show that there is a vulnerability (*e.g.* buffer overflow vulnerability) in service *s* which is offered by host *h*. As mentioned before, the existence of these kinds of vulnerabilities on individual hosts can be detected with host scanning tools like Nessus [20] and System Scanner [21]. In the remaining of this section, we will show how the aforementioned classes of vulnerabilities can be modeled with this method.

2.2.1 Service usurp vulnerability

Suppose that service s is running with privileges of account a on port p of host h . If attacker a can access this port and s has a vulnerability that can be exploited by him (e.g. buffer overflow vulnerability) then he can run his code on that host with privileges of account a :

<p>Vulnerability <i>ServiceUsurp</i> Preconditions: <i>Service(s, p, a, h)</i> <i>NetworkAccess(A, h, p)</i> <i>ServiceVuln(s, h)</i> Postconditions: <i>Priv(A, a, h)</i></p>

2.2.2 Local program vulnerability

Sometimes a program or a service accepts some inputs from its user and without controlling their contents for being malicious uses them. The following example shows this issue:

```
Program P (fname:string)
  cmd = append ("Open", fname);
  execute (cmd);
```

Here, we have a program that expects its users to give it some file name. It trusts its users and doesn't check the input parameter submitted by a user. Now, if an attacker calls this program with $P(\text{"data.txt; rm -rf *"})$ then the program will delete all of the files in the current directory. In this vulnerability, if the program or service that has this vulnerability runs with the privileges of account a on host h , then the attacker can exploit this vulnerability and get access to the a account on this host.

If the programs that run on hosts accessible to attackers have vulnerabilities such as unexpected input, attackers can use these vulnerabilities and increase their privileges. In other words, in this class of vulnerabilities, there is a program on host h which executes with the privileges of account a' and has a vulnerability that can be exploited by attackers. If an attacker has access to account a on this host, then he can exploit this vulnerability and get access to privileges of account a' . If *LocalPrgVuln(h, a)* be used for showing that there is a local program vulnerability in a program which is running with privileges of account a on host h then this vulnerability can be modeled as follows:

<p>Vulnerability <i>LocalProgramVuln</i> Preconditions: <i>Priv(A, a, h)</i> <i>LocalPrgVuln(h, a')</i> Postconditions: <i>Priv(A, a', h)</i></p>

2.2.3 Weak password vulnerability

If an attacker has access to an account on a host and there is another account on this host with a weak password, then this attacker can guess the password of this account and get access to this new account. In fact, when a user has access to one account on a host, he can find all of the other accounts on this host. In Unix based operating systems, this can be done by reading the contents of the `/etc/passwd` file. After finding all of the accounts on a host, the attacker can guess the password of those accounts on this host which have weak passwords.

Another form of this vulnerability happens when a host offers services like Telnet and SSH that users can use to have a remote login to a host. An attacker which knows there is a user account a on this host (e.g. using social engineering methods) can use a dictionary attack to find the password of this account.

Models of these classes of vulnerabilities are as follows:

<p>Vulnerability <i>LocalWeakPassword</i> Preconditions: $Priv(A, a, h)$ $Account(a', h)$ $WeakPassword(a', h)$ Postconditions: $Priv(A, a', h)$</p>	<p>Vulnerability <i>RemoteWeakPassword</i> Preconditions: $Knows(A, a, h)$ $NetworkAccess(A, h, a)$ $PasswordAuth(p)$ $WeakPassword(a, h)$ Postconditions: $Priv(A, a, h)$</p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.2.4 Denial of service vulnerability

In this kind of vulnerability, a service that is running on a host is vulnerable to some patterns of data. Attackers that have access to this host can exploit this vulnerability and prevent the service from responding to requests:

<p>Vulnerability <i>DoS</i> Preconditions: $Service(s, p, u, h)$ $NetworkAccess(A, h, p)$ $DoSVuln(s, h)$ Postconditions: $\neg Service(s, p, u, h)$</p>

2.2.5 Confidentiality loss

In this class of vulnerabilities, an attacker A that was not allowed to read some confidential data has got privileges of a new user account a that is authorized to read these data. Therefore although the attacker is not the legal owner of account a , he can use the privileges of this account and read these confidential data:

<p>Vulnerability <i>ConfidentialityLoss</i></p> <p>Preconditions:</p> <p>$Priv(A, a, h)$</p> <p>$CanReadConfidentialData(a, h, path)$</p> <p>$\neg AccountOwner(A, a, h)$</p> <p>Postconditions:</p> <p>$ConfidentialityLoss(h, path)$</p>

2.2.6 Integrity loss

In this class of vulnerabilities, an attacker A that was not allowed to modify some data has got privileges of a new user account a that is authorized to modify these data. Therefore although the attacker is not the legal owner of account a , he can use the privileges of this account and modify these confidential data:

<p>Vulnerability <i>IntegrityLoss</i></p> <p>Preconditions:</p> <p>$Priv(A, a, h)$</p> <p>$CanModifyData(a, h, path)$</p> <p>$\neg AccountOwner(A, a, h)$</p> <p>Postconditions:</p> <p>$IntegrityLoss(h, path)$</p>

3. CASE STUDY

In this section, we use the framework introduced in the previous section to find the attack paths of the sample network shown in Fig. 2. The figure shows a local area network protected by the firewall FW . The attacker resides outside of firewall and owns the host $Devil$.

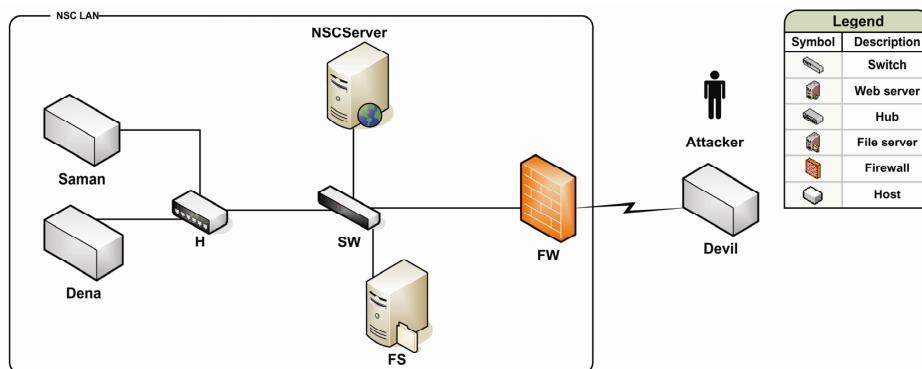


Fig. 2. A Sample LAN network for case study.

The rules defined at the firewall *FW* just allow external users to access web and email services, and internal users to access any service outside. The Host *NSCServer* offers *HTTP* service on port *http* and *MAIL* service on port *smtp*. The *HTTP* and *MAIL* services run with *apache* and *root* accounts privileges, respectively. The initial configuration of the network is represented by the following facts:

InZone(NSC, Dena), InZone(NSC, Saman)
InZone(NSC, NSCServer), InZone(NSC, FS)
Service(HTTP, http, apache, NSCServer)
Service(MAIL, smtp, root, NSCServer)
Service(SMB, smb, root, FS)
Service(SSH, ssh, root, FS)
Service(RPC, rpc, root, Saman)
InZone(Internet, Devil)
FirewallRule(fw, AnyHost, NSCServer, http, permit, 0)
FirewallRule(fw, AnyHost, NSCServer, smtp, permit, 1)
FirewallRule(fw, AnyHost, AnyHost, AnyPort, deny, 2)
ZoneAddressInInterface(fw, in1, NSC)
ZoneAddressInInterface(fw, in2, Internet)

After using the Nessus scanning tool, we found that *NSCServer* has buffer overflow vulnerability in its *HTTP* service. Host *Saman* and *FS* have also the same vulnerability in their *RPC* and *SSH* services, respectively. We show these vulnerabilities using the following facts:

ServiceVuln(HTTP, NSCServer)
ServiceVuln(RPC, Saman)
ServiceVuln(SSH, FS)

Suppose that the attacker has *root* account on *Devil* host which is located in the external network and his goal is to get access to *manager* account on host *FS*. Therefore, we have the fact *Priv(Attacker, root, Devil)* and we want to check the value of proposition *Priv(Attacker, manager, FS)*.

Proof: We applying the inference rule (1) on the initial results:

$$\frac{\text{ZoneAddressInInterface}(fw, in2, Internet) \quad \text{ZoneAddressInInterface}(fw, in1, NSC)}{\text{BetweenZones}(fw, Internet, NSC)}$$

By applying inference rule (5), we can deduce that host *Devil* can send packets to port *http* of host *NSCServer*:

Devil \in *AnyHost*
NSCServer \in *AnyHost*
Firewallrule(fw, AnyHost, AnyHost, http, permit, 0)

$$\frac{\exists \text{FirewallRule}(fw, H', H, P, deny, pr') \bullet ((pr' < 0) \wedge (Devil \in H') \wedge (NSCServer \in H) \wedge (http \in P))}{\text{PacketCanPass}(fw, Devil, NSCServer, http)}$$

Now, we apply the inference rule (7):

$$\frac{\begin{array}{l} \text{Service}(HTTP, http, apache, NSCServer) \\ \text{InZone}(NSC, NSCServer) \\ \text{InZone}(Internet, Devil) \\ NSC \neq Internet \\ \forall \text{BetweenZones}(fw, Internet, NSC) \bullet \\ \text{PacketCanPass}(fw, Devil, NSCServer, http) \end{array}}{\text{CanConnect}(Devil, NSCServer, http)}$$

Inference rule (3) shows that *Attacker* can access *http* port of *NSCServer*:

$$\frac{\begin{array}{l} \text{CanConnect}(Devil, NSCServer, http) \\ \text{Priv}(\text{Attacker}, root, Devil) \end{array}}{\text{NetworkAccess}(\text{Attacker}, NSCServer, http)}$$

At this point all of the pre-conditions of *Service Usurp* vulnerability are satisfied, so its post-conditions can also be added to the facts list:

<p>Vulnerability <i>ServiceUsurp</i></p> <p>Preconditions:</p> <p style="padding-left: 40px;"><i>Service(HTTP, http, apache, NSCServer)</i></p> <p style="padding-left: 40px;"><i>NetworkAccess(Attacker, NSCServer, http)</i></p> <p style="padding-left: 40px;"><i>ServiceVuln(HTTP, NSCServer)</i></p> <p>Postconditions:</p> <p style="padding-left: 40px;"><i>Priv(Attacker, apache, NSCServer)</i></p>

Inference rule (6) shows that *NSCServer* can connect to port *SSH* of *FS*:

$$\frac{\begin{array}{l} \text{InZone}(NSC, NSCServer) \\ \text{InZone}(NSC, FS) \\ \text{Service}(SSH, ssh, root, FS) \end{array}}{\text{CanConnect}(NSCServer, FS, ssh)}$$

Inference rule (3) shows that attacker can access *SSH* port of host *FS*:

$$\frac{\begin{array}{l} \text{CanConnect}(NSCServer, FS, ssh) \\ \text{Priv}(\text{Attacker}, apache, NSCServer) \end{array}}{\text{NetworkAccess}(\text{Attacker}, FS, ssh)}$$

Now, exploiting the Service Usurp vulnerability of the *SSH* service of *FS* host yields that:

<p>Vulnerability <i>ServiceUsurp</i> Preconditions: <i>Service(SSH, ssh, root, FS)</i> <i>NetworkAccess(Attacker, FS, ssh)</i> <i>ServiceVuln(SSH, FS)</i> Postconditions: <i>Priv(Attacker, root, FS)</i></p>

From inference rule (4), we can deduce that *Attacker* can access the *manager* account on host *FS*, because he has access to the *root* account on this host. □

In summary, we found that attacker can use the following attack scenario to reach his goal:

- (1) Exploit the *HTTP* service vulnerability of host *NSCServer* and get access to the *apache* account on this host.
- (2) Exploit the *apache* account of *NSCServer* to exploit the *SSH* service vulnerability of host *FS* and get the *root* account on this host.

4. AUTOMATIC VULNERABILITY ANALYSIS USING AN EXPERT SYSTEM

An expert system is a computer application that performs a task that would otherwise be performed by a human expert. For example, there are expert systems that can diagnose human illnesses, make financial forecasts, and schedule routes for delivery vehicles. To design an expert system, one needs a knowledge engineer, an individual who studies how human experts make decisions and translates the rules into terms that a computer can understand. The knowledge engineer must make sure that the computer has all the knowledge needed to solve a problem. He must also ensure that the computer can use the knowledge efficiently by selecting from a handful of reasoning methods.

Every expert system consists of two major parts: the knowledge base; and the reasoning, or inference, engine. Knowledge representation formalisms are used to formalize and organize the knowledge. One widely used representation is the production rule, or simply rule. A rule consists of an *IF* part and a *THEN* part (also called a *condition* and an *action*). The *IF* part lists a set of conditions in some logical combination. If the *IF* part of the rule is satisfied; consequently, the *THEN* part can be concluded. Expert systems whose knowledge is represented in rule form are called rule-based expert systems.

The problem-solving model, or paradigm, organizes and controls the steps taken to solve the problem. One common but powerful paradigm involves chaining of *IF-THEN* rules to form a line of reasoning. If the chaining starts from a set of conditions and moves toward some conclusion, the method is called *forward-chaining*. If the conclusion is known (for example, a goal to be achieved) but the path to that conclusion is not known, then reasoning backwards is called for, and the method is *backward-chaining*. These

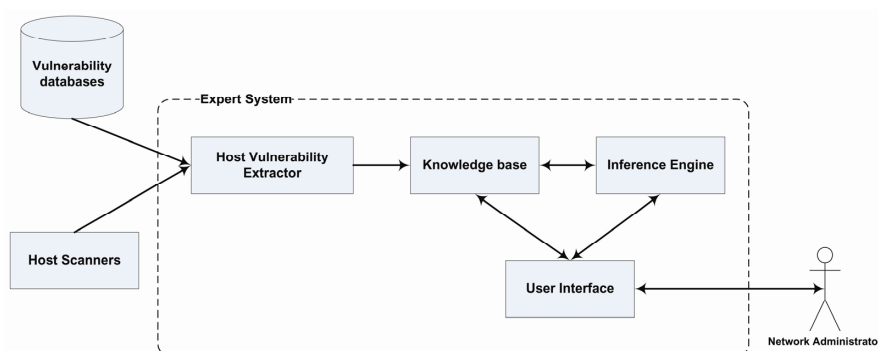


Fig. 3. Expert system architecture.

problem-solving methods are built into program modules called inference engines or inference procedures that manipulate and use knowledge in the knowledge base to form a line of reasoning.

We implemented an expert system based on the framework introduced in the previous section for automatic topological multi-host vulnerability analysis. This expert system consists of four main components: *Knowledge Base*, *Inference Engine*, *User Interface*, and *Host Vulnerability Extractor* as shown in Fig. 3. Knowledge Base is the collection of facts and inference rules that were introduced in the previous sections. Inference Engine is the processing unit that makes logical inferences on the facts and rules that are stored in the knowledge base. User Interface controls the inference engine and manages inputs and outputs.

The Host Vulnerability Extractor receives its input from two sources: vulnerability databases and host scanners. Host scanners search individual hosts for known vulnerabilities and send their results to this component. Then, this component searches the vulnerability databases to find the class of vulnerabilities that each of the detected vulnerabilities fall into. Then, it creates the corresponding facts for these vulnerabilities as introduced in section 2 and sends the result to knowledge base. For example, suppose that after using the Nessus vulnerability scanner, this tool reports that host $h1$ has a vulnerability with code CVE-2001-0554 in its Telnet service. It sends this information to Host Vulnerability Extractor component. This component contacts with the National Vulnerability Database [22] and asks for information about this vulnerability. After processing the reply it finds that this is a buffer overflow vulnerability that can be abused by attackers for gaining root privileges on this host. So it sends the *ServiceVuln(Telnet, h1)* proposition to the Knowledge base component for storing there.

The framework that was introduced in section 2 consists of three main components: (1) A set of initial facts that represents the network initial topology and configuration. (2) A set of inference rules that can be used to deduce additional information about connectivity of hosts in the network and accessibility of them. (3) A set of vulnerability models that have pre-conditions and post-conditions and can change the state of the system if their pre-conditions are satisfied. In other words, in this framework we have a number of initial facts and a set of rules and therefore it can be easily implemented as a rule based expert system.

We used the matured expert system tool CLIPS [23] for reducing the amount of time

required for developing the expert system. It is an expert system tool which is designed to facilitate the development of software to model human knowledge or expertise. By using this tool, we can specify the initial facts and inference rules. Then it will do the inference process based on the facts and rules and can find all of the facts that can be derived from initial facts by applying inference rules. Below are the details of implementing the proposed framework in CLIPS.

4.1 Defining Initial Facts

Initial facts can be simply defined in CLIPS using the `deffacts` instruction. For example, the following is the definition of some initial facts of the system introduced in section 3 as case study:

```
(deffacts initial-facts
  (InZone NSC NSCServer)
  (Service MAIL smtp root NSCServer)
  (InZone Internet Devil)
  (FirewallRule fw AnyHost NSCServer http permit 0)
  (FirewallRule fw AnyHost NSCServer smtp permit 1)
  (FirewallRule fw AnyHost AnyHost AnyPort deny 2)
  (ServiceVuln HTTP NSCServer)
  (ServiceVuln RPC Saman)
  (ServiceVuln SSH FS)
  (Priv Attacker root Devil)
  (Account manager FS)
)
```

Some of these initial fact (*e.g.* those about network topology) must be defined manually for CLIPS. The others can be automatically deduced from the outputs of other tools. For example, for defining the vulnerabilities of individual hosts we can translate the output generated by host scanning tools like Nessus and System Scanner to CLIPS facts.

4.2 Defining Inference Rules

Inference rules can be defined in CLIPS using the `defrule` instruction. CLIPS supports advanced constructs of the first order logic such as \exists and \forall , which can be used in defining complex rules. The followings are representations of two of the rules introduced in section 2:

```
(defrule NetAccessRule
  (CanConnect ?h1 ?h2 ?p)
  (Priv ?u ?a ?h1)
  =>
  (printout t ?u " has network access to port" ?p "of host" ?h2 crlf)
  (assert (NetworkAccess ?u ?h2 ?p)))
```

```

(defrule FirewallRule
  (FirewallRule ?fw ?H2 ?H ?P allow ?pr)
  (test (or
    (eq ?H2 AnyHost) (subsetp (create$ ?h2) (create$ ?H2))))
  (test (or
    (eq ?H AnyHost) (subsetp (create$ ?h) (create$ ?H))))
  (test (or
    (eq ?P AnyPort) (subsetp (create$ ?p) (create$ ?P))))
  (not (exists (FirewallRule ?fw ?H3 ?H4 ?P4 deny ?pr4)
    (test (< ?pr2 ?pr))
    (test (or
      (eq ?H3 AnyHost) (subsetp (create$ ?h2) (create$ ?H3))))
    (test (or
      (eq ?H4 AnyHost) (subsetp (create$ ?h) (create$ ?H4))))
    (test (or
      (eq ?P4 AnyPort) (subsetp (create$ ?p) (create$ ?P4))))))
  =>
  (assert (PacketCanPass ?fw ?h2 ?p))
)

```

4.3 Defining Vulnerability Models

In our framework vulnerabilities are represented as inference rules. The following is representation of two of the vulnerabilities in CLIPS:

```

(defrule LocalProgramVulnRule
  (LocalPrgVuln ?h ?a2)
  (Priv ?A ?a ?h)
  =>
  (assert (Priv ?A ?a2 ?h))
  (printout t "Attacker" ?A "can exploit a local program"
    "vulnerability on host" ?h "and get the privileges"
    "of account" ?a2 "on this host.")
)

(defrule RemoteWeakPasswordVulnRule
  (Knows ?A ?a ?h)
  (NetworkAccess ?A ?h ?p)
  (RemoteConnectProtocol ?p)
  (WeakPassword ?a ?h)
  =>
  (assert (Priv ?A ?a ?h))
)

```

CLIPS uses a forward chaining algorithm and after starting execution, it will continue running until no change is made in the state of the system (*i.e.* no new fact is added

to the fact list and none of the rules is fired for execution). But, in many cases we want to halt the execution after a given condition is satisfied. For example, in the sample network introduced in section 3 our goal is to check whether the attacker can get access to the *manager* account on *FS*. In these cases we can add an extra rule to the system, so upon salification of a condition, the system stops execution. For example, the following is one such rule:

```
(defrule GoalRule
  (Priv Attacker manager FS)
  =>
  (printout t "GOAL Satisfied!" crlf)
  (halt)
)
```

4.4 Performance Evaluation

We used simulation for evaluating the performance and scalability of the implemented expert system. We generated virtual networks with various topologies and configurations and with different number of hosts and tested the performance of the implemented expert system on these networks. For generating random networks, we used the following parameters:

- *Hosts*: number of hosts in the network.
- *Services*: number of different services that are offered by individual hosts.
- *External-Accessible hosts*: number of hosts that are accessible from the external network.
- *Vulnerabilities*: number of different vulnerabilities that are injected in the generated network.

Table 1. Elapsed times of the analysis on different random generated networks.

Network Configuration				Elapsed Times	
Hosts	Services	External-Accessible Hosts	Vulnerabilites	Generation Time (milliseconds)	Analysis Time (milliseconds)
10	3	1	10	0.9	0.5
100	10	5	100	35.8	18.4
200	20	10	200	206.1	80.8
400	40	12	400	1517.9	626.5
600	55	20	600	4567.4	1947.0
800	60	25	800	8778.0	4112.4
1000	65	30	1000	14739.2	5877.7
1200	75	35	1200	24433.7	10718.8
1400	85	40	1400	38074.1	25314.5
1600	90	50	1600	51506.0	33070.0

Table 1 shows the elapsed times for generating random network configurations and also the elapsed times for doing the vulnerability analysis on these network configurations. These experiments are done on a 2.4 GHz Pentium processor with 512MB of RAM. As can be seen in this table, our expert system can automatically analyze the vulnerability of a big network with 1600 hosts in just about 31 seconds, which is a quite acceptable execution time.

Fig. 4 shows how the analysis time is affected by the number of vulnerabilities in the network. In the first trend, on average, there is one vulnerability per host and in the last trend, on average, there are 50 vulnerabilities per host. As it is shown in the figure, our expert system is scalable with regard to the number of vulnerabilities and even for a network with 1600 hosts and 80000 vulnerabilities, the analysis is done in just about 60 seconds.

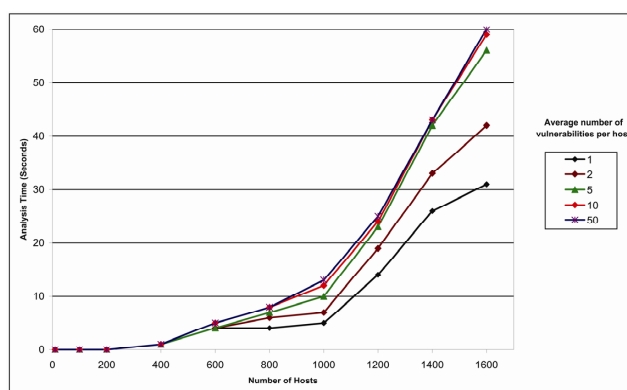


Fig. 4. Analysis time of the expert system versus number of hosts.

5. RELATED WORKS

Previous approaches to network vulnerability analysis mainly focused on model checking and graph-based techniques and most of them depend on off-the-shelf tools for scanning individual host vulnerabilities. Vulnerability scanner tools such as System Scanner [21] and Nessus [20] scan hosts to discover software and configuration vulnerabilities. However, they do not investigate how a combination of configurations on the same host or among hosts on the same network can contribute to the vulnerabilities. The NetKuang system [3] tries to assess beyond host vulnerabilities. It is an extension of a previous work on building a rule-based expert system, namely Kuang. NetKuang extends the Kuang's rule-set to include certain UNIX network security issues, which are undetectable when analyzing a single host. NetKuang uses a backward-chained goal-based search. Given initial and goal privileges, it analyzes the systems on which it is running to determine if the initial privileges are adequate to achieve the target privileges. Although it was the first work tried to detect multi-phase attacks, it was limited to some UNIX configuration vulnerabilities, mainly the critical file permissions, and did not propose any general model for vulnerabilities.

Dacier [4] proposed the concept of privilege graphs. Privilege graphs are explored

to construct an attack state graph, which represents different ways in which an intruder may reach a certain goal such as root access on a host. Swiler *et al.* presented a method in [5] for generating attack graphs by forward exploration. However, there was no approach for generating the privilege graph. In these approaches the network topology was not considered. Amman *et al.* [6] proposed a graph-based search algorithm to analyze network vulnerabilities. This approach has been adopted in [7] by Noel *et al.* and investigated more in [8]. In their approach, exploits are modeled as pre- and post-conditions and a specific tool has been used to construct the attack graph. Although the algorithm is efficient, encoding each exploit individually resulted in a large and complex model and the role of firewalls was not considered.

Noel and Jajodia [7] used model checking for vulnerability analysis of networks via the model checker SMV. They could obtain only one attack corresponding to an unsafe state and the experiment was restricted to only specific vulnerabilities. However, the model checking approach has been used in some other researches, such as [24, 25], to analyze network vulnerabilities. The main drawback of model checking is the scalability problem, which some researchers tried to overcome [7]. Ramakrishnan and Sekar [9] used a model checker to analyze a single host system with respect to combinations of unknown vulnerabilities. The key issue in their research was checking infinite space model using model abstraction. In [10], CSP was used to model and analyze TCP protocol vulnerabilities. In this approach, the model checker FDR2 was used to verify some simple security properties and find attack scenarios. CSP has been used widely in modeling and analyzing security protocols [26] and verifying intrusion detection systems [27]. But all of these approaches suffer from state space explosion problem.

The access control based approach has been explored in [11]. In this approach the Take-Grant access control model has been extended to address the vulnerabilities. Some general patterns of vulnerabilities have been modeled, and then the closure of the graph representing the protection state of the system has been generated. The main drawback of this approach is to generate the initial VTG of the model.

The main area of security that expert systems have been used in is Intrusion Detection and many rule based intrusion detection systems, such as [15-19], are proposed. In these approaches, an expert system is utilized to reason about the security state of the system, given rules that describe intrusive behavior. Often forward-chaining, production-based tools are used, since these are most appropriate when dealing with systems where new facts (audit events) are constantly entered into the system. These expert systems are often of considerable power and flexibility [28]. The main difference of Intrusion Detection System (IDS) and our work is that IDS detects the attacks when they occur which is usually done online, while we try to use the static analysis of the network configuration to detect attack methods before happening and therefore our approach does not have real-time constraints.

Perhaps the most similar approach to ours is MulVal [12]. It adopts Datalog, which is a subset of Prolog, as the modeling language for the elements in the analysis. The reasoning engine consists of a collection of Datalog rules that capture the operating system behavior and the interactions of various components in the network. MulVal uses the XSB environment which is an environment for executing Prolog programs. The main problem of MulVal approach is that it uses a backward search algorithm for determining whether goals can be achieved from initial state. In this kind of reasoning, users always

must define some goal and then the reasoner will check whether the goal is satisfiable or not. For example, the user can ask the system if it is possible to gain the root account on some host.

Unlike MulVal reasoning algorithm, the inference engine of our expert system uses a forward chaining algorithm and starting from the initial state it finds new facts that can be derived from initial facts by applying production rules. Therefore in our expert system, it is not required to specify specific goals and limit the reasoning to that goals. Since the number of facts is polynomial in the number of hosts, the inferencing process will terminate efficiently.

Another advantage of our approach to the previous works is that it regards the topology of the network and analyzes it. Specially the firewall filtering rules are modeled and analyzed to determine the accessibility of services and other resources.

The other advantage of our approach to MulVal is that we model and analyze all categories of vulnerabilities such as *Confidential Loss* and *Integrity Loss*, and more important the *Denial of Service* attacks. In our framework, the vulnerabilities which may lead to interruption of access of authorized users can be modeled easily (as seen in section 2.2.4). The DoS vulnerabilities may take part in a complex attack scenario. For example, if there is an Intrusion Detection System in a network, the attacker first tries to bypass it or disrupt it by DoS attacks. Accordingly, these kinds of attacks which is not detectable in previous works, can be analyzed and detected in our framework.

6. DISCUSSION

In this paper, we proposed a general framework for modeling typical network topologies, configurations, and vulnerabilities and then showed how an expert system can be implemented based on this framework for automating the process of multi-host vulnerability analysis. Our approach explores all attacking paths and generates the closure of access rights that an attacker can gain. Having the closure, it can be detected in $O(1)$ time complexity whether a user has an especial right over a resource or not.

Since the exploitation of most of the vulnerabilities yields to the change in access rights, they can be represented easily in our framework. Additionally, DoS vulnerabilities which may interrupt a service or an access can be represented and analyzed, which is a new contribution over previous works. However, there are DoS attacks that just downgrade the quality of service (QoS) instead of stopping the service. This class of DoS attacks needs to consider the quality of service in a quantitative approach. It can be done via fuzzy expert systems [29], which is not in the scope of this paper and lies in the future works. Another advantage of our framework is that it uses a high level of abstraction for specifying network configurations and topologies and does not rely on specific protocols and standards.

One of our major contributions of this work is the modeling and analyzing of network topologies to obtain the initial accesses of principals in the networks. In our approach, the firewall filtering rules are modeled and analyzed in this regard. Moreover, combination of multiple firewall deployments can be analyzed. However, the process of representing the initial configuration of the network such as firewall rules is done manually. The automatic detection of network configuration and topology has been considered

a research trend in recent years [30-33] and can be utilized to automate the representing of initial facts.

Generally, to construct the initial facts, we need three categories of information. These include host vulnerabilities, remote vulnerabilities and network topology. There are some tools which can be utilized to gather this information. The OVAL scanner is a tool developed based on the Open Vulnerability Assessment Language (OVAL) [34], an XML-based language which is used to formally describe the host configuration vulnerabilities and the required tests to detect them. This standard has been proposed by Mitre Corp. on which some major operating system vendors have reached consensus. As of writing this paper, the OVAL definitions for Windows, Redhat Linux, and Sun Solaris operating systems vulnerabilities are available. The Nessus vulnerability scanner is a powerful tool which utilizes the NASL language to find the vulnerabilities of network services remotely [20]. The typical TCP/IP networks topology can be captured using several tools, such as Cheops-ng [35].

Our analysis approach gains a forward chaining algorithm and starting from the initial state finds new facts that can be derived from the initial facts by applying production rules. It has two benefits: first, it is not required to specify specific goals and limit the reasoning to that goals. Second, the derivation phase is done just once and then the satisfaction of any attack goal can be checked in $O(1)$ and since the number of facts is polynomial in the number of hosts, the inferencing process will terminate efficiently.

The developed expert system which is used for automatic vulnerability analysis includes a set of inference rules. Accordingly, the system needs a database of vulnerabilities such as CVE [36] for encoding the inference rules. This task may be time consuming if done for each of the individual vulnerabilities. However, having a proper categorization of vulnerabilities can help this stage to done more efficiently. We are developing a categorization of vulnerabilities based on their effect on the security policy of the system, and will utilize it in our future works. The maintenance of inference rules needs tracking the new announced vulnerabilities and adding them to the database. Fortunately, OVAL language [34] has been developed to automate the process of vulnerability announcement. However, the effects of vulnerabilities have not been specified formally by OVAL, yet. Thus, it needs human interference to represent it as an inference rule.

7. CONCLUDING REMARKS

The growth of computer networks has increased the demand for automating the process of network vulnerability analysis. The prerequisite of building tools for this purpose is to have formal models of network configurations and vulnerabilities. In this paper, we first proposed a simple and flexible framework for network configuration and topology modeling. We then showed how network vulnerabilities can be modeled in this framework. The main advantage of this framework is the simplicity of the method which is used for modeling vulnerabilities. Although the number of network vulnerabilities is very large, similar vulnerabilities can be classified in the same class and instead of modeling vulnerabilities one by one, classes of vulnerabilities can be modeled. For example, although there are a great number of buffer overflow vulnerabilities, most of them have the same behavior and can be modeled in just one class.

We then showed how a rule-based expert system for automatic vulnerability analysis can be implemented based on this framework and discussed its architecture and performance. Our approach is flexible and can catch many types of vulnerabilities, such as some DoS vulnerabilities which may lead to interruption. The analysis method efficiently finds all security violations at the end of the analysis phase. Since the number of facts that are discovered by the expert system are polynomial in the number of hosts, the analysis is done in polynomial time and our experiments show this fact.

The future works lies in two aspects: first we need to automate the process of creating initial facts of network topology using off-the-shelf tools. Second, a more proper categorization of vulnerabilities must be used to ease the process of vulnerability modeling.

REFERENCES

1. J. Pinkston, J. Undercoffer, A. Joshi, and T. Finin, "A target-centric ontology for intrusion detection," in *Proceedings of IJCAI Workshop on Ontologies and Distributed Systems*, 2003.
2. H. Ray, R. Vemuri, and H. Kantubhukta, "Toward an automated attack model for red teams," *IEEE Security and Privacy Magazine*, Vol. 3, 2005, pp. 18-25.
3. D. Zerkle and K. Levitt, "NetKuang – A multi-host configuration vulnerability checker," in *Proceedings of the 6th USENIX Security Symposium*, 1996, pp. 195-204.
4. M. Dacier and Y. Deswarte, "Privilege graph: an extension to the typed access matrix model," in *Proceedings of the 3rd European Symposium on Research in Computer Security*, LNCS 875, 1994, pp. 319-334.
5. L. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer attack graph generation tool," in *Proceedings of DARPA Information Survivability Conference & Exposition II*, 2001, pp. 307-312.
6. P. Amman, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 217-224.
7. S. Jajodia, S. Noel, and B. O'Berry, "Managing cyber threats: issues, approaches and challenges," *Topological Analysis of Network Attack Vulnerability*, Kluwer Academic Publisher, 2005.
8. S. Noel, E. Robertson, and S. Jajodia, "Correlating intrusion events and building attack scenarios through attack graph distances," in *Proceedings of the 20th Annual Computer Security Applications Conference*, 2004, pp. 350-359.
9. C. R. Ramakrishnan and R. Sekar, "Model-based analysis of configuration vulnerabilities," *Journal of Computer Security*, Vol. 10, 2002, pp. 189-209.
10. H. R. Shahriari and R. Jalili, "Using CSP to model and analyze transmission control protocol vulnerabilities within the broadcast network," in *Proceedings of the IEEE International Networking and Communication Conference*, 2004, pp. 42-47.
11. H. R. Shahriari, R. Sadoddin, R. Jalili, R. Zakeri, and A. Omidian, "Network vulnerability analysis through vulnerability take-grant model (VTG)," in *Proceedings of the 7th International Conference on Information and Communications Security*, LNCS 3783, 2005, pp. 256-268.

12. X. Ou, S. Govindavajhala, and A. W. Appel, "MulVal: a logic-based network security analyzer," in *Proceedings of the 14th USENIX Security Symposium*, 2005, pp. 113-128.
13. S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *Proceedings of the 19th Annual Computer Security Applications Conference*, 2003, pp. 86-95.
14. P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Proceedings of the 6th Nordic Workshop on Secure IT-Systems*, 2001, pp. 100-107.
15. T. F. Lunt and R. Jagannathan, "A prototype real-time intrusion-detection expert system," in *Proceedings of IEEE Symposium on Security and Privacy*, 1988, p. 59.
16. A. Mounji, "Languages and tools for rule-based distributed intrusion detection," Ph.D. Dissertation, Facultés Universitaires Notre-Dame de la Paix Namur, Belgium, 1997.
17. P. A. Porras and P. G. Neumann, "EMERALD: event monitoring enabling responses to anomalous live disturbances," in *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, 1997, pp. 353-365, <http://citeseer.ist.psu.edu/porras97emerald.html>.
18. U. Lindqvist and P. A. Porras, "Detecting computer and network misuse through the production-based expert system toolset (p-best)," in *Proceedings of the IEEE Symposium on Security and Privacy*, 1999, pp. 146-161.
19. U. Lindqvist and P. Porras, "eXpert-BSM: a host-based intrusion detection solution for Sun Solaris," in *Proceedings of the 17th IEEE Annual Computer Security Applications Conference*, 2001, p. 0240.
20. "Nessus vulnerability scanner," <http://www.nessus.org>.
21. "Internet security systems – system scanner," <http://www.iss.net/>.
22. "National vulnerability database," <http://nvd.nist.gov/>.
23. "CLIPS: a tool for building expert systems," <http://www.ghg.net/clips/CLIPS.html>.
24. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of IEEE Symposium on Security and Privacy*, 2002, pp. 273-284.
25. S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, 2002, pp. 49-63.
26. P. Ryan and S. Schneider, *Modeling and Analysis of Security Protocols: A CSP Approach*, Addison-Wesley, 2001.
27. G. Rohrmair and G. Lowe, "Using data-independence in the analysis of intrusion detection systems," *Theoretical Computer Science*, Vol. 340, 2003, pp. 82-101.
28. S. Axelsson, "Intrusion detection systems: a survey and taxonomy," Technical Report No. 99-15, Chalmers University of Technology, Sweden, 2000, <http://citeseer.ist.psu.edu/axelsson00intrusion.html>.
29. A. Kandel, *Fuzzy Expert Systems*, CRC Press, Inc., Boca Raton, FL, USA, 1992.
30. J. Schönwälder and H. Langendörfer, "How to keep track of your network configuration," in *Proceedings of the 7th USENIX Conference on System Administration*, 1993, pp. 189-193.
31. H. C. Lin, S. C. Lai, and P. W. Chen, "An algorithm for automatic topology discovery of IP networks," in *Proceedings of IEEE International Conference on Commu-*

- nications*, Vol. 2, 1998, pp. 1192-1196.
32. B. Lowekamp, D. O'Hallaron, and T. Gross, "Topology discovery for large ethernet networks," in *Proceedings of the Conference on Applications, Technologies, architectures, and Protocols for Computer Communications*, 2001, pp. 237-248.
 33. H. Tian, "Network topology discovery and its applications," Ph.D. Dissertation, School of Information Science, Japan Advanced Institute of Science and Technology, 2006.
 34. M. Wojcik, T. Bergeron, T. Wittbold, and R. Roberge, *Introduction to the Oval Language*, Version 5.0, <http://oval.mitre.org/oval/about/documents.html>.
 35. "Cheops-ng: the network swiss army knife," <http://cheops-ng.sourceforge.net/>.
 36. Mitre Corporation, "Common vulnerabilities and exposures database," <http://cve.mitre.org>.



Hamid Reza Shahriari is currently an assistant professor at the Department of Computer Engineering and Information Technology in Amirkabir University of Technology in Tehran, Iran. He received his Ph.D. in Computer Science from Sharif University of Technology in 2007. His research interests include information security, vulnerability analysis and formal methods in security.



Yasser Ganjisaffar is a graduate student in the Department of Computer Engineering at Sharif University of Technology. He is also a research assistant in the Web Intelligence Laboratory. His research interests include network security, expert systems and web technologies.



Rasool Jalili received his Ph.D. in Computer Science from University of Sydney, Australia in 1995. He then joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. He is now an associated professor, doing research in the areas of distributed computing and information security in his network security laboratory; nds.sharif.edu.



Jafar Habibi received his B.S. degree in Computer Engineering from the Supreme School of Computer, his M.S. degree in Industrial Engineering from Tarbiat Modares University and his Ph.D. degree in Computer Engineering from Manchester University. At present, he is an associate professor at the Computer Engineering department at Sharif University of Technology. His research interests are mainly in the area of computer engineering, simulation systems, MIS, DSS and evaluation of computer systems performance.