ESC-TR-2005-064

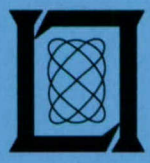
Project Report
IA-2

Evaluating and Strengthening Enterprise Network Security Using Attack Graphs

R.P. Lippmann
K.W. Ingols
C. Scott
K. Piwowarski
K.J. Kratkiewicz
M. Artz
R.K. Cunningham

5 October 2005

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS



Prepared for the Department of the Air Force under Contract FA8721-05-C-0002.

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Department of the Air Force, HQCPSG/NIS, under Contract FA8721-05-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Gary Tutungian
Administrative Contracting Officer
Plans and Programs Directorate
Contracted Support Management

Massachusetts Institute of Technology
Lincoln Laboratory

Evaluating and Strengthening Enterprise Network Security Using Attack Graphs

R.P. Lippmann
K.W. Ingols
C. Scott
K. Piwowarski
K.J. Kratkiewicz
M. Artz
R.K. Cunningham
Group 62

Project Report IA-2

5 October 2005

Lexington Massachusetts

ABSTRACT

Assessing the security of large enterprise networks is complex and labor intensive. Current security analysis tools typically examine only individual firewalls, routers, or hosts separately and do not comprehensively analyze overall network security. We present a new approach that uses configuration information on firewalls and vulnerability information on all network devices to build attack graphs that show how far inside and outside attackers can progress through a network by successively compromising exposed and vulnerable hosts. In addition, attack graphs are automatically analyzed to produce a small set of prioritized recommendations to enhance network security. Field trials on networks with up to 3,400 hosts demonstrate the ability to accurately identify a small number of critical stepping-stone hosts that need to be patched to protect against external attackers. Simulation studies on complex networks with more than 40,000 hosts demonstrate good scaling. This analysis can be used for many purposes, including identifying critical stepping-stone hosts to patch or protect with a firewall, comparing the security of alternative network designs, determining the security risk caused by proposed changes in firewall rules or new vulnerabilities, and identifying the most critical hosts to patch when a new vulnerability is announced. Unique aspects of this work are new attack graph generation algorithms that scale to enterprise networks with thousands of hosts, efficient approaches to determine what other hosts and ports in large networks are reachable from each individual host, automatic data importation from network vulnerability scanners and firewalls, and automatic attack graph analyses to generate recommendations.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONTINUED)

LIST OF TABLES

1. INTRODUCTION

It is difficult to secure large enterprise networks. Such networks are vulnerable to physical attacks on network components, social engineering attacks where users are coerced into revealing sensitive information, and cyber attacks where malicious attackers achieve higher levels of access privilege than should be allowed over network connections. This paper focuses on cyber attacks. These include attacks launched by malicious outsiders on the Internet and malicious insiders directly connected to internal networks. Both types of attackers can take advantage of vulnerabilities in network protocols and in systems such as servers, desktops, routers, gateways, and firewalls. Analysis of cyber attacks is difficult because compromising one system often provides a stepping-stone that can be used to launch further attacks. An attacker can often jump from system to system, eventually achieving a goal such as compromising an essential mail or file server.

Attackers may obtain different privilege levels on compromised systems. One important goal is to obtain the privilege of a system administrator (e.g., root on UNIX systems or administrator on Windows systems). Attackers also seek the lower privilege of a user or server, gather information (e.g., a user account name, a list of files, the contents of a file, or access to a database or other server), or disable a system or server by performing a Denial of Service (DoS) attack.

An essential type of analysis for security assessment is to determine the level of compromise possible for important hosts in a network from a given attacker starting location. This is a complex task. The answer depends on the network topology, on the security policy in the network as expressed by the placement of firewalls, routers, switches, and the rule sets used in these devices, and on vulnerabilities in systems and protocols. This type of analysis is often performed by a "red team" of computer security professionals who actively probe a network and often run exploits that compromise systems. Red team exercises can be very effective in locating weaknesses in a network, but they are labor intensive and time-consuming, and can disrupt system operations. This paper presents an alternative approach that automates basic red-team analyses, does not interfere with network operations, and requires no additional network traffic other than that used to perform normal host vulnerability scans.

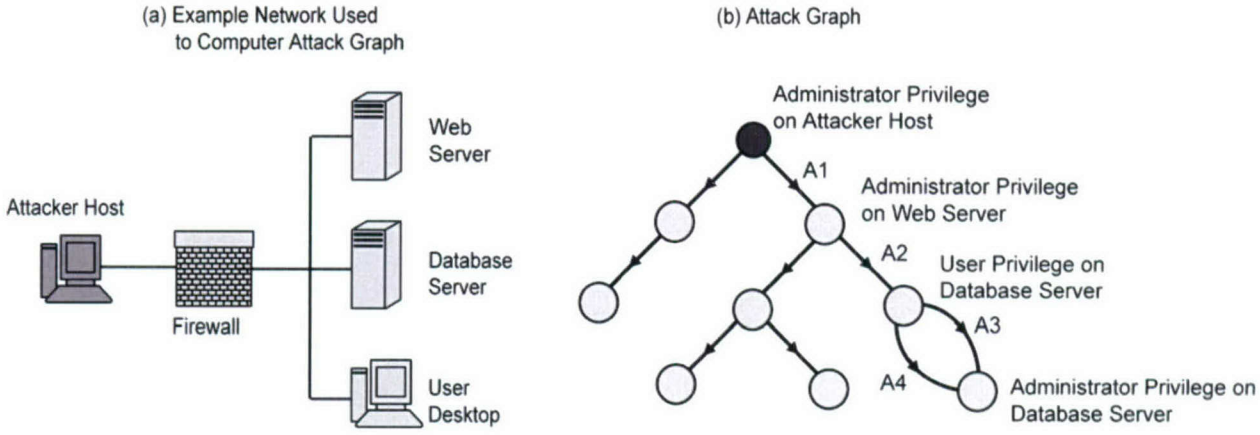


Figure 1. Example of a simple network (a) and a partial attack graph for this network (b). Each node in the attack graph represents a state of the network and attacker; arcs represent specific exploits or attacks; and the attacker starts at the black node representing the attacker host.

Our approach uses attack graphs as an underlying analysis tool. The left-hand side of Figure 1 shows a simple network with a firewall, two internal servers, and an internal user desktop. The right side of this figure shows a partial attack graph for an attacker located outside the firewall. In this graph, each node represents the state of the network and attacker privileges obtained so far. Each edge represents a specific attacker action, exploit, or attack component used to directly or indirectly gain additional privileges. This attack graph shows that an outside attacker can first use a remotely exploitable vulnerability A1 that is exposed by the firewall and allows the attacker to achieve administrator-level privileges on the web server, a type of vulnerability we call *remote-to-admin*. The database server can then be compromised first at user level by *remote-to-user* exploit A2 from the web server, and this privilege can be raised to administrator level by *local-to-admin* exploits A3 or A4. Other sequences of exploits are also possible, as indicated by the unlabeled part of this graph.

A single attack graph includes all the hosts that can be compromised in a network and the level of privilege obtained by the attacker on these hosts. This type of graph is more comprehensive than those used in previous studies (e.g., [1, 2]) that only determine how an attacker can reach a single goal from a starting point. Instead of building separate graphs for each potential attacker target, only one attack graph needs to be constructed. Paths to hosts in the attack graph represent the sequences of attacks that achieve the level of compromise shown. To simplify the graphical presentation, multiple attacks that achieve the same level of privilege (e.g., A3 and A4 in Figure 1) may connect the same two nodes. Each edge that connects two attack graph nodes represents one attacker action, but the physical path used may be complex and pass through multiple routers, communications channels, and firewalls. An edge indicates that the destination node has a vulnerability that the attacker can successfully exploit, either by contacting the machine remotely if the vulnerability is remotely exploitable or by previously obtaining user privileges on the machine if the vulnerability is only locally exploitable.

We have recently developed approaches that made it feasible to create attack graphs for large networks. This required (1) the availability and widespread use of automated vulnerability scanning tools such as Nessus to collect data, (2) new types of attack graphs and construction algorithms, (3) efficient approaches to determine reachability between hosts, and (4) automated tools to analyze large enterprise-network attack graphs to make recommendations based on these graphs. The remainder of this paper provides an overview of the most recent version of a system named NetSPA (Network Security Planning Architecture) that generates attack graphs and uses them to make recommendations concerning network security. The paper describes requirements for any tool that generates attack graphs, the types of data that must be imported, how reachability analyses can be performed efficiently, how large graphs for enterprise networks can be built efficiently, and the types of graph analyses that can be performed to provide useful recommendations. It also provides examples of how this type of analysis can improve security on different networks, presents the results of simulation studies, and discusses other work in this area.

2. SYSTEM REQUIREMENTS

Our initial attack graph research focused on algorithm development. We soon discovered, however, that system administrators would not allow us to test or even use tools unless they satisfied some practical requirements. First, because attack graphs provide a blueprint for malicious users, they need to be generated on site in a physically protected area and preferably on a computer not connected to any network. Second, because networks are large and complex and system administrators have limited time, only small amounts of information should be entered by hand. Vulnerability information on hosts must be imported from scanners such as Nessus [3] that are already in use. Firewall and router information as well as details concerning recent vulnerabilities and attacks must also be imported automatically from configuration files. Finally, because analyses often have to be performed on site using existing computers, computations should be performed on relatively inexpensive but capable commodity hardware. It should be possible to analyze networks with thousands of hosts in minutes on such hardware.

Figure 2 shows a block diagram of an off-line system that can perform multiple analyses on a network and is designed to meet these requirements. Preprocessing software on the left automatically imports information required to generate attack graphs. The structured relational database shown in the middle is used to support multiple runs and analyses. The database supports the analyses shown on the right of the graph: reachability analysis, attack graph generation, and attack graph analysis.

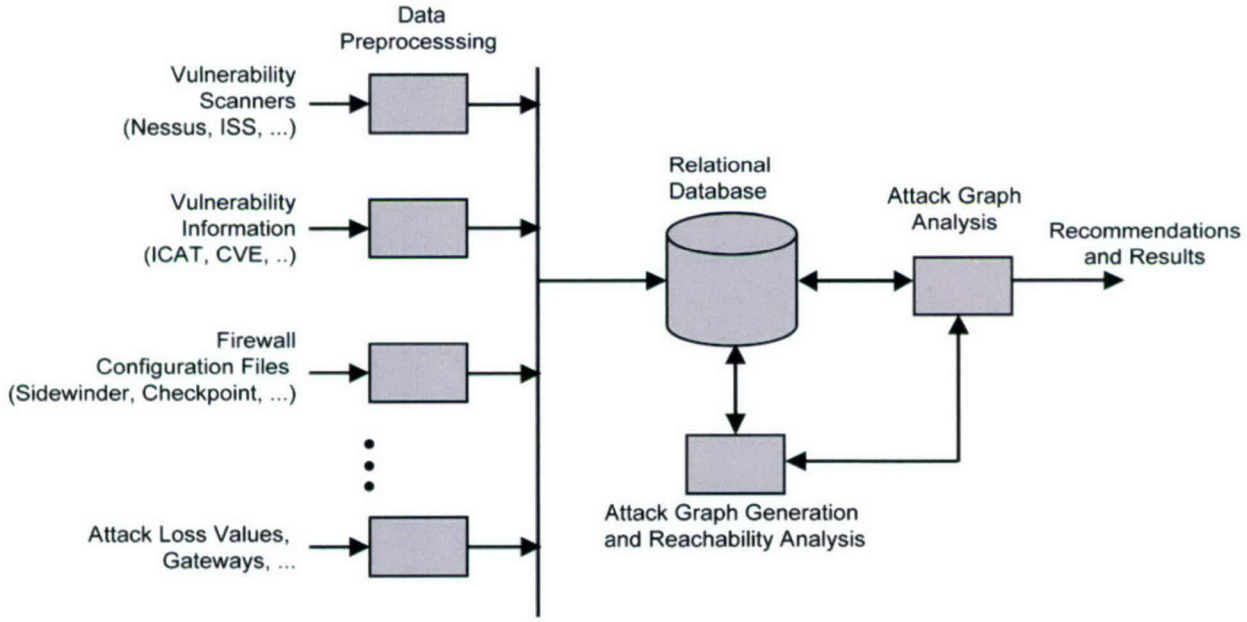


Figure 2. Block diagram of the NetSPA tool.

Reachability analysis determines how hosts communicate. It takes firewall rules and the network topology into account to determine whether any two hosts can communicate using a given TCP or UDP destination port. Attack graph construction uses reachability information and data on prerequisites and effects of per-host attacks to construct attack graphs given specific attacker starting locations. Each attack graph shows all hosts that can be compromised from one or more attacker starting locations. A final attack graph analysis examines attack graphs to make recommendations and provide details on network security.

Automated analyses are required because attack graphs are complex to understand and interpret for even moderately sized networks. Although initial NetSPA prototypes performed these three analyses sequentially, the current system integrates reachability and attack graph construction and performs often expensive reachability computations only on demand when required to construct attack graphs. The following four sections of this paper first describe details concerning information required to construct attack graphs and then the three types of analyses.

3. INFORMATION REQUIRED TO BUILD ATTACK GRAPHS

A major effort was made when designing NetSPA to automatically import as much of the data required to build attack graphs as possible and to minimize hand data entry. Table 1 shows information used to construct attack graphs, how this information is obtained, and whether it is required or optional. The first two columns list the types of data and the sources of the data. The third column indicates whether the data is extracted automatically from the source; the fourth indicates if the data is required or optional; and the last column specifies actions that must be taken to obtain the desired data. For example, in the first row, one vulnerability scan is required for each subnet to determine the presence of hosts and vulnerabilities on hosts in a subnet, and each scan is imported. Here and throughout the report, a subnet is defined as a collection of hosts and network devices with interfaces that share the same IPv4 address space and are connected by devices that do not perform filtering. A network is a collection of subnets and network devices that connect these subnets. In the second row of Table 1, the complete NIST ICAT database [4], the CVE vulnerability dictionary [5], and other information is imported. Only network topology information must be entered by hand in order for NetSPA to perform an analysis.

TABLE 1
Data Sources Used to Create Attack Graphs

Information	Source	Automatic	Required	Actions
Vulnerability and Host Presence	Network Vulnerability Scanner	Yes	Yes	Perform a vulnerability scan for each subnet
Vulnerability Prerequisites and Effects	ICAT Database, Scanner Data, CVE Dictionary, and optional Security Expert Analyses	Yes	Yes	Import ICAT database, scanner data, CVE dictionary, and optional configuration file with expert analyses
Firewall Rules	Configuration Files	Yes	Yes	Import rules for each firewall
Gateway Information	System Administrator	No	Yes	Provide network topology (gateway and subnet interconnections)
Administrative Host	System Administrator	No	No	Indicate administrative hosts
Asset Values	System Administrator	No	No	Specify asset values for all hosts

3.1 A WORST-CASE ASSUMPTION FOR VULNERABILITIES

The first two rows of Table 1 describe how information about vulnerabilities is obtained. Different types of security analyses require different levels of detail concerning vulnerabilities. After examining available vulnerability databases and finding inconsistencies, missing information, and no standard terminology or taxonomy, we decided that vulnerabilities should be characterized using only limited

information and few low-level details. Because accurate low-level requirements for exploiting vulnerabilities could not be easily obtained, we decided to follow a worst-case approach. This approach is similar to Swiler's [2] use of default values for missing information, but we assume worst-case default values. We assume that if there is a vulnerability, it will be exploited as long as the attacker can reach the vulnerable machine over the protocol or service required to exploit the vulnerability. Details concerning the exploit code used, where it is stored, how it is hidden, the attacker motivation or skills, other steps required to compromise a host, or the difficulty of the exploit are irrelevant for this worst-case analysis. These details may be useful for forensic analysis or to predict exploitation frequency, but they are difficult to obtain, often highly dynamic and variable over time, and not required to analyze the worst-case security of a network. For this purpose it is sufficient to know what vulnerabilities exist, how they can be accessed over the network, and the access or privilege level they provide.

A worst-case approach is supported by vulnerability and exploit analyses reported in [6]. Their analysis found that the vast majority (roughly 70%) of vulnerabilities announced in the first half of 2004 are easy to exploit because exploit code is either available or unnecessary. When exploit code was developed after the announcement of a vulnerability, the average time to publish the code was only six days. This means that malicious attackers with limited skills could almost always exploit new vulnerabilities within a few days of their announcement. The same study found that in many instances exploit code was rapidly incorporated into automatic self-propagating worms and bot-networks (remotely controlled systems used to launch new attacks), and it was not unusual for automatic agents to include exploits for multiple vulnerabilities. Automated worms were surprisingly effective, and it was found that worm traffic originated from IP spaces controlled by more than 40% of Fortune 100 companies. This suggests that automated worms have compromised a large fraction of presumably well-secured networks. The worst-case assumption is also supported by the 2004 E-Crime Watch Survey [7] in which roughly 70% of the responding organizations reported one or more e-crimes or intrusions and by the Computer Crime and Security Survey [8] in which more than half of the respondents reported unauthorized use of computer systems in the past year. These reports of intrusions are likely to be low because many companies don't want to admit that their networks have been compromised. The intrusions that are reported suggest that malicious attackers and automated agents are actively exploiting exposed vulnerabilities and that exploitation can rapidly follow announcements of a new vulnerability.

3.2 FINDING VULNERABILITIES

The first row of Table 1 indicates that the existence of vulnerabilities and hosts is currently determined using network vulnerability scanners such as the Nessus network scanner [3]. Network scans are becoming common practice and they provide a listing of all networked devices and lists of vulnerabilities found on each device. Open-source scanners such as Nessus are kept up to date by a dedicated user community and are often updated to detect new vulnerabilities within a few days of or simultaneous with vulnerability announcements (e.g., [9]). We performed a validation of the Nessus scanner results on a Windows 2000 and Solaris host using a host-based scanner and found that it accurately detects vulnerabilities that can be exploited remotely, except that it is overly pessimistic. In some cases, it reports vulnerabilities in servers such as IIS when a vulnerability may occur, but is not

ruled out by the server version number. These "false alarms" can occur when system administrators follow the common practice of allowing scanners only to execute non-damaging vulnerability scans that rely primarily on server software version numbers extracted by the scanner.

If the most recent patch dates of network devices are known, the information can be used to eliminate some vulnerability scanner "false alarms." This involves first determining the last date each host was fully patched. When the patch release date for a vulnerability is before the last host patch date, then it is often reasonable to assume that that vulnerability has been patched and that the vulnerability is a "false alarm." System administrators frequently know these dates because they patch systems manually or use automated patching tools. At sites where hosts are centrally managed and patched frequently, there are few vulnerabilities, and it is practical for system administrators to confirm the existence of all or a few critical vulnerabilities by hand. If patch management tools and host-based vulnerability scanners become standardized and enjoy widespread use, it should become possible to obtain more accurate information about host vulnerabilities from these tools.

3.3 ASSIGNING PORTS AND PROTOCOLS TO VULNERABILITIES

Some vulnerability scanners such as Nessus provide extra information, including TCP/UDP ports used to connect to a system when exploiting vulnerabilities, open ports where servers are running, and CVE identifiers for discovered vulnerabilities. This information makes our attack graph generation more accurate. Information on TCP/UDP ports used to exploit vulnerabilities is the most important. It is required during reachability analysis and attack graph generation to determine if filtering devices such as firewalls or routers prevent an attacker from exploiting a vulnerability by blocking necessary ports or protocols. CVE numbers make it possible to cross-reference information in vulnerability databases. They make it easier to determine prerequisites and effects of attacks that exploit vulnerabilities. Finally, information concerning open ports is used to identify hosts running services that use clear-text passwords that can be sniffed and also to identify hosts running special services used by administrator hosts for remote access and control.

3.4 ATTACK PREREQUISITES AND EFFECTS

As shown in the second row of Table 1, prerequisites and effects of attacks that exploit vulnerabilities found in the network need to be known. Prerequisites specify necessary conditions to launch attacks and effects specify capabilities an attacker obtains by exploiting vulnerabilities. Initially we thought that vulnerability details would be easy to obtain, given the many online vulnerability databases and the common use of CVE vulnerability numbers. However, an analysis of vulnerability databases and listings (e.g., [3–5, 10, 11]), demonstrated that detailed information on vulnerabilities and CVE cross-reference numbers are often missing. Information across vulnerability websites is also sometimes inconsistent, especially lists of affected software and version numbers. This suggested that attack models would only be accurate if we rely on limited basic vulnerability information that can be easily obtained and verified.

Vulnerability scanners identify the presence of vulnerabilities, but do not provide the two types of additional information required to construct attack graphs that are shown in Table 2. Table 2a on the left defines the *locality* of an attack. This differentiates between *remote* attacks launched from a different host and *local* attacks launched from and against the same host. Remote attacks allow an attacker to progress to a new host while local attacks allow an attacker to achieve a higher level of privilege on the same host. Table 2b on the right defines the *effect* of an attack. This specifies the level of privilege gained by the attacker. For simplicity and because more detailed information is difficult to obtain, we use the following four levels shown in Table 2b: (1) User, (2) Administrator, (3) Denial of Service (DoS), and (4) Other. The *user* level models a typical user. The *administrator* level models the administrator on Windows or root user on UNIX who has access to the entire system. The *DoS* level models attacks that can disable a service but cannot gain access. It is assumed that an attacker who achieves administrator privilege can also launch a DoS attack. Finally, the *other* category includes attacks that provide information about a host, make it possible to read files, or provide only limited access to a printer or database on a system. We write our evaluation of a vulnerability as *locality*-to-*effect*. For example, a vulnerability that is remotely exploitable and yields administrator access when exploited is a *remote-to-administrator*, or more simply *remote-to-admin* or *r2a*, attack.

TABLE 2
Vulnerabilities Are Characterized Using Two Categories That Specify Locality (a) and Four Categories That Specify the Effect of Exploitation (b)

(a) Locality
1. Remote (Launch from remote host)
2. Local (Launch from local host)

(b) Effect
1. User (User privilege)
2. Administrator (Administrator or root privilege)
3. Denial of Service (DoS)
4. Other (read/write file, limited access to database, ...)

Two approaches are used to determine the locality and effect of attacks. The first and most accurate approach is to have a security expert determine this information by hand. We found that roughly 90–100 vulnerabilities can be analyzed per day when extracting only this information for vulnerabilities found using the Nessus vulnerability scanner on a typical network. This high rate of analysis assumes the use of a custom tool that extracts critical text from Nessus plugin descriptions, the ICAT database, the CVE dictionary, and the Bugtraq vulnerability database. Manual analysis is practical when there are only 10's or 100's of new non-analyzed vulnerabilities on a network. It can also be performed incrementally as new vulnerabilities are announced. To support this hand analysis we store the resulting locality and effect analyses in a configuration file, import this information into NetSPA, and rely on it exclusively when available. One expert can thus provide vulnerability information for NetSPA analyses performed at many sites. This hand analysis needs to be revisited as new exploits for a given vulnerability are developed or knowledge about the vulnerability develops. For example, many buffer overflow vulnerabilities that are

originally labeled DoS eventually become remote-to-admin vulnerabilities as more advanced exploits are developed.

For general security analyses on large networks where thousands of vulnerabilities may occur, we developed an automated pattern classifier that determines locality and effect for all vulnerabilities that can be found by Nessus. A pattern classifier was used to provide accurate decisions with missing and inconsistent information and to integrate information across multiple data sources including free-text descriptions. A block diagram of this classifier is shown in Figure 3. As shown on the left, it relies on information from (1) the Nessus vulnerability scanner, (2) the ICAT database, and (3) the CVE vulnerability dictionary. We use information extracted from all these sources as input features for two pattern classifiers contained in the right block of Figure 3. One determines locality and the other determines effect. The Nessus scanner shown at the top left of the figure provides a single text description for all vulnerabilities included in each Nessus plugin. It also categorizes vulnerabilities into different types. We use the following three Nessus types that are relevant to determining the attack effect: (1) Gain a shell remotely, (2) Gain root remotely, (3) Denial of Service. We also use the Nessus rating of severity that ranges from "low" to "critical" in five levels. The ICAT database shown at the bottom left provides a binary "attack range" category that specifies locality as local or remote that we use. It also provides six binary valued "loss types" that can be set independently and that we use. These are (1) Obtain superuser privilege, (2) Obtain user privilege, (3) Obtain other privilege, (4) Loss of availability, (5) Loss of confidentiality, and (6) Loss of integrity. Finally, we use text descriptions of vulnerabilities provided by the CVE dictionary (e.g., "Stack-based buffer overflow in AppleFileServer for Mac OS X 10.3.3 and earlier allows remote attackers to execute arbitrary code via ...").

The CVE dictionary and ICAT database can only be used if the Nessus vulnerability scanner provides a CVE number for the detected vulnerability. This is indicated by the dotted lines in Figure 3. If a CVE number is not provided, then the classifier uses only the text description and categorical values provided by Nessus. If a CVE number is provided, then the classifier also uses categorical database fields from the ICAT database when they are available as well as text descriptions from the CVE dictionary. The upper two and bottom two arrows into the right-hand box of Figure 3 represent binary input features that indicate whether a corresponding categorical feature was selected. There is one input per category, and the input is one if the category is selected and zero otherwise. The middle binary inputs from the box labeled "Text Analysis" indicate whether common generic phrases used in human-readable text that often indicate categorical values occur in text descriptions. Phrases are collected into groups for each category. For example, the administrator privilege category can be indicated by phrases including "execute arbitrary code" and "gain system privileges." The remote locality category can be indicated by phrases including "remote host" and "remotely." There are four binary inputs for each of the arrows labeled "effect" and two for each of the arrows labeled "locality" that represent the categorical values shown in Table 2. These binary inputs are one if any one of the phrases in a corresponding phrase group are found and zero if they are not. All types of binary inputs are also zero if categorical inputs are missing or if no phrases are found by the text analysis. Multiple features that represent locality and effect are provided to the classifier to compensate for missing and inconsistent information across the three data sources.

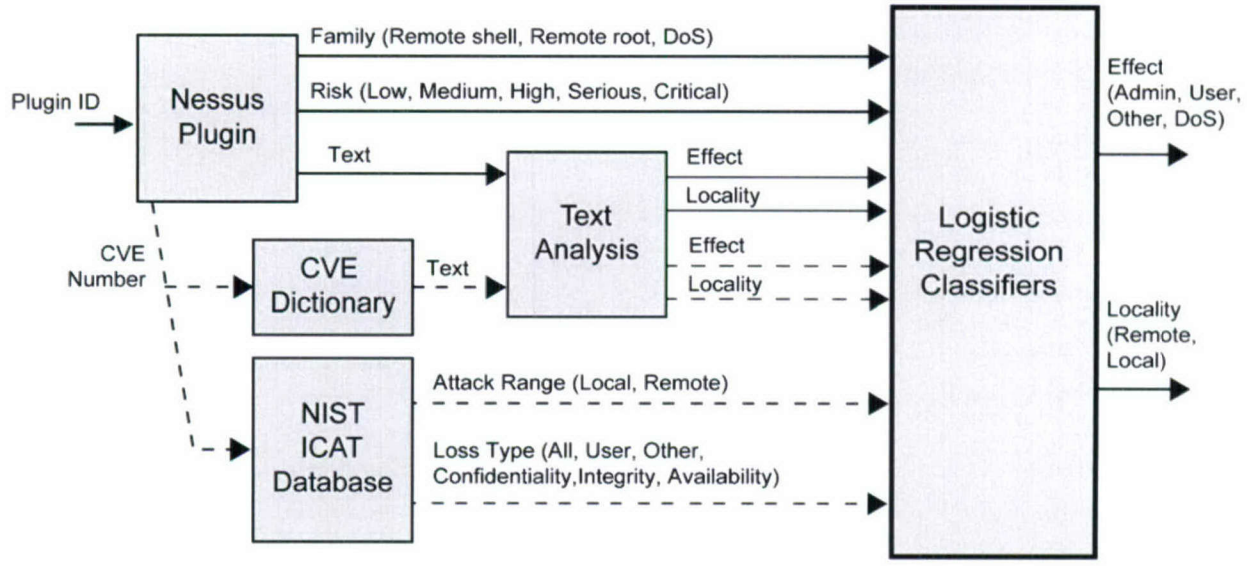


Figure 3. Block diagram of classifiers that determine the effect and locality of vulnerabilities found using the Nessus vulnerability scanner. Dotted lines indicate information only available if the CVE number is provided by the Nessus plugin.

Different types of classifiers were evaluated using the LNKnet pattern classification software package [12]. It was found that a logistic regression classifier performed as well as more complex decision trees and support vector machine classifiers. A logistic regression classifier was developed using 215 hand-analyzed vulnerabilities found by Nessus. Ten-fold cross-validation testing on this data demonstrated that the overall accuracy is nearly 100% correct for locality and better than 90% correct for effect. The small number of effect errors was caused primarily by missing values in the ICAT category fields and ambiguous text descriptions. Although there are still a few errors for effect, this classifier makes it possible to perform security analyses for large networks using the most recent version of Nessus that searches for the most up-to-date vulnerabilities. An execution of NetSPA's data preprocessing stage requires importing the most recent NIST ICAT database and the CVE dictionary and re-running the automatic vulnerability classifier. To reduce the effect of errors, the hand analysis described above overrides the classifier output. In addition, NetSPA makes recommendations that usually involve a small number of critical vulnerabilities. These few vulnerabilities can be hand verified after recommendations are provided to detect classification errors and improve the overall recommendations. Any corrections in the effect and locality of a vulnerability can then be added to the configuration file to prevent the same error from occurring in future analyses.

Our hand analysis suggested some useful extensions to our approach of alert characterization that could be added in the future. One is to add more attack effects and extend the *other* category to differentiate between confidentiality loss and integrity loss. Confidentiality loss includes data exfiltration attacks, such as reading a password file or the user's email box. Integrity loss includes data corruption attacks, such as modifying cost fields on a vendor's database. Another extension is to extend the concept

of locality to include client-side attacks in which a user accesses a malicious server with a vulnerable client (e.g., a vulnerable web browser), allowing the server to attack the client software. Client-side attacks would require information on host client software that is usually not available from network vulnerability scans but may be provided by automated patching systems, network management systems, or host-based vulnerability scanners. Information on the *other* category might be provided by specialized vulnerability scanners designed to assess database and web servers or by similar patch or network management tools.

It would also be useful to provide more fine-grained analysis of effects. In some cases a separate analysis for loss of confidentiality, integrity, and availability might be desired. We currently can perform such an analysis to determine how an attacker can affect availability of services by including vulnerabilities in which the effect is *DoS* or *administrator*. An analysis to determine if an attacker can affect integrity and confidentiality can include vulnerabilities in which the effect is *user* or *administrator*. It is difficult, however, to analyze the effect on confidentiality, integrity, and availability when an attacker exploits a vulnerability and the effect is *other*. More detailed analyses of the *other* category would make this possible because the *other* category currently implies loss of confidentiality and/or integrity for specific programs or data. Because this detailed analysis is difficult to perform with the available data, it was not included in the current version of NetSPA.

3.5 FIREWALL CONFIGURATION FILES

The third row of Table 1 shows that configuration files for firewalls that perform traffic filtering are required to determine reachability between hosts. For firewalls, configuration files contain rule sets or access control lists (ACLs) for traffic filtering and network address translation (NAT). Rule sets must be considered because a firewall can prevent outside access to a vulnerability if the protocol or the TCP/UDP port required to exploit it is blocked. Although similar types of filtering can be performed by routers, we currently do not read in router configuration files. This is planned for a future version of NetSPA. The current system imports configuration files from Sidewinder and Checkpoint firewalls. Router and firewall rules are necessary to determine the reachability between hosts. Some past systems (e.g., [1]) assumed that reachability can be determined using information from vulnerability scanners. Such data is incomplete because firewalls frequently allow connections only from specific source IP addresses or ports. A vulnerability scanner will not find ACLs that create holes in firewalls by permitting traffic only from specific IP addresses. It will only find the potentially limited connections that are allowed from the IP address of the host running the vulnerability scanner. Even if the vulnerability scanner were to exercise the filtering rules properly, it is still prohibitive to scan the entire network from each network segment as the network size grows. This is also a weakness of port and IP scanners such as NMAP [13]. Only an analysis of firewall and router rules can determine complete reachability through these and other devices that perform traffic filtering.

3.6 GATEWAYS AND ADMINISTRATION HOSTS

Information in the last three rows of Table 1 is provided by a system administrator. The names of gateway hosts, filtering and routing devices, and the subnets they interconnect make it possible to determine the overall network topology and combine data from vulnerability scans of separate subnets. This data indicates how to interconnect separate subnets. The names of administration hosts and a list of network services used for remote administration make it possible to add vulnerabilities associated with system administration. In particular, if any administrator host is compromised, it can be assumed that all remotely-administered hosts that have an open port for the service used for remote administration (e.g., SSH or VNC) are also compromised.

3.7 HOST ASSET VALUES

Providing host asset values listed in the last row of Table 1 makes it easier to generate recommendations following construction of attack graphs. Each host is assigned an asset value representing the importance of the host and loss to the network if that host is compromised. After generating an attack graph, host asset values are used to determine a value called the Network Compromised Percentage (NCP) as shown in Equation 1.

$$\text{Network Compromised Percentage} = 100 \times \frac{\sum_{\substack{\text{Compromised} \\ \text{Hosts}}} \text{Host Asset Value}}{\sum_{\text{All Hosts}} \text{Host Asset Value}} \tag{1}$$

The NCP is the percentage of the total assets across all hosts that have been captured by the attacker. It is the sum of host asset values across all compromised hosts determined by adding the asset values for all hosts on the attack graph divided by the sum of the host asset values across all hosts in the network. This fraction is then multiplied by 100 to convert it to a percentage. A value of zero indicates that no hosts have been compromised, and a value of 100 indicates that all hosts have been compromised and all assets have been captured. This metric is used to compare alternative approaches to improving network security. Any measure that reduces the NCP is assumed to improve overall network security.

Our current default analysis counts a host as compromised in Equation 1 and considers it as captured by the attacker when the effect of any exploited vulnerability used to compromise the host is *user*, *administrator*, or *DoS*. It is also possible to prevent vulnerabilities with an effect of *other* or *DoS* from being counted as compromised in Equation 1. Such an analysis considers a host to be captured only if it is compromised at *user* or *administrator* privilege levels.

For simplicity, the asset value should represent the worst-case loss incurred by either loss of confidentiality, integrity, or availability. Useful recommendations can be made if the asset value is categorical and takes on one of three to five values as suggested in [14]. For example, in a network where

confidentiality and availability are most important, user desktops can have the lowest and default asset value (e.g., 100), switches that support traffic to multiple desktops may have higher values (e.g., 1000) and file servers required by all workstations may have the highest value (e.g., 5000). In most cases, values selected for categories are roughly proportional to the sum of the assets of the systems that are affected by a host compromise. For example, compromising a switch used by ten hosts with asset values of 100 each causes a loss of 1,000 due to loss of availability. The switch should thus have an asset value of 1,000. Compromising a file server for 50 hosts with asset values of 100 each causes a total loss of 5,000 due to all three loss types. The file server should thus have an asset value of 5,000. The current NetSPA tool has no easy method of inferring asset values of shared network infrastructure devices because information on dependencies is difficult to obtain. System administrators thus need to provide asset values for all hosts. We have found that system administrators can easily understand the concept of a host asset value. They are able to assign these values to hosts in large networks because only a few servers and other special hosts need to be assigned higher asset values and all other hosts can be assigned the lowest default asset value.

The standard in [14] suggests using three categories or values for host asset values (low, moderate, and high adverse effect). We have only limited experience with the sensitivity of final recommendations to asset values and the numbers of categories used. In many network analyses, a group of stepping-stone hosts is discovered that allows compromise of many other machines by an attacker starting from outside a perimeter firewall. The most important resulting recommendation is to patch or block the vulnerabilities that allow this group of hosts to be compromised. This recommendation will remain the most important, even for widely different host asset values. In other networks with inside attackers, asset values will directly influence recommendations and the highest priority recommendations will be to eliminate vulnerabilities in the hosts with the highest asset values. It is not computationally expensive to explore the effect of varying host asset values on recommendations. Since the attack graph is not affected by asset values and only the order of recommendations is affected, many different asset values could potentially be explored to analyze how the order of recommendations varies with asset values.

3.8 AUTOMATIC IMPORT UTILITIES

Each data source described above has a corresponding import utility customized for the specific formats and semantics of that source. These utilities extract necessary fields from the source, map the information into the schema of the NetSPA database, and generate SQL scripts to populate the database. Import utilities for vulnerability scanners input scanner results and populate the network, vulnerability, and attacker action tables in the database. The vulnerability classifier shown in Figure 3 and described in Section 3.4 is implemented in the scanner import utility. The classifier determines the locality and effect for each vulnerability found by a vulnerability scanner unless this information is already available in a human-generated configuration file also read in by the import utility.

Other studies used software names and versions to determine whether servers are vulnerable instead of vulnerability scanner outputs (e.g., [15, 16]). We did not follow this approach because an analysis of version numbers reported by Nessus on one of our class C subnets suggested that using software versions

to find vulnerabilities is error prone and unreliable. This analysis revealed three major problems with using software names and versions. First, version numbers are often not reported by Nessus (they were available for only roughly 30% of all open ports). Second, software version numbers often remain the same even across many patches and vulnerabilities and thus do not differentiate between patched and unpatched software. Third, databases such as ICAT do not provide complete listings of all vulnerable software versions.

Import utilities for firewall configuration files read configuration files and populate firewall policy tables in the database. The firewall import utilities map firewall-specific constructs into a generic structure in the database. For some firewalls and rules this mapping is very straightforward; in other cases the mapping is complex and multiple rules may be generated for a single rule in the data source.

Import utilities for configuration files read a comma-delimited file and update the database. The required gateway information file allows the import utilities to connect the subnets together properly and to identify hosts with multiple interfaces. This is especially important for firewalls.

Optional configuration files provide information on hand-analyzed attacks, asset values, and administration hosts. Information provided in configuration files may add information to the database, update existing information, or delete information. Host names and IP addresses in configuration files are disambiguated by providing a subnet identifier, which associates a host name or IP address with a specific subnet. Disambiguation of IP addresses is necessary when a network contains enclaves using overlapping IP address space.

3.9 THE NETSPA DATABASE

The NetSPA relational database contains all the information required to generate and analyze attack graphs. The database is populated by import utilities as described in the preceding section. It is then read at the beginning of each experiment to load data into memory and does not need to be accessed when generating and analyzing attack graphs. The database includes four types of tables: (1) vulnerability, (2) attacker actions, (3) network description, and 4) firewall policy. Vulnerability tables contain information concerning vulnerabilities and remote access services discovered by the vulnerability scanner. A vulnerability is linked directly to the TCP/UDP port where it was detected. This linkage from ports to vulnerabilities enables efficient attack graph generation because once reachability is found to a port, a list of vulnerabilities is immediately available that can be used to extend the attack graph. Vulnerability tables also contain server software names and versions when they are provided by the vulnerability scanner. These are stored only to provide further detailed information in the NetSPA system's final reports.

Unlike other approaches that use complex formal languages to describe exploits and actions that attackers can take (e.g., [1, 17]), we describe attacks using the simple locality and effect values shown in Table 2. Attacker action database tables contain values of locality and effect for all vulnerabilities found. Attacker actions are constrained by these values, reachability, and the location of vulnerable ports on hosts. An attacker on one host can use a vulnerability to compromise a different victim host by exploiting a vulnerability only if the vulnerability locality is remote, if the victim has that vulnerability on a port, and

if the victim's port is reachable from the attacker. The access gained on the victim host is specified by the effect of the vulnerability. An attacker with *user* privilege on a host can elevate this privilege by exploiting a vulnerability only if the locality of the vulnerability is local, the host has the vulnerability, and the effect of the vulnerability is to provide *administrator* privilege.

Network description tables contain the network topology, including all the hosts in the network, interfaces on the hosts, ports open on the interfaces, network addresses the interfaces respond to, and DNS names associated with the addresses. Network description tables also specify hosts in subnets and how subnets are interconnected to form a network. A network is the unit of work for NetSPA. All processing is performed within the context of a network.

Firewall policy tables contain filtering and network address translation (NAT) rules. Rule tables contain a full description of each rule, including relationships to the host enforcing the rule, the source and destination host interfaces, the source and destination network objects, filtering and address translation rules, and any time restrictions governing when the rule is in effect. In these tables, a network object is one of several types, including an IP address, an address range, a Domain Name Server (DNS) host name, a DNS domain name, or a group of other network objects. We currently support two options for handling time restrictions on rules. The default behavior is a worst-case assumption. All "open" or "allow" rules activated only during specified time intervals are changed to be permanently open and all "block" rules activated only at specified time intervals are eliminated. This models an attacker who waits for "open" rules to activate and for "block" rules to deactivate. It is also possible to specify a start time when NetSPA begins and to use the firewall rules that apply at this start time.

4. REACHABILITY ANALYSIS

Critical to generating attack graphs is an analysis that determines which hosts are reachable from an already compromised host and which TCP/UDP ports can be used to connect to and compromise new vulnerable hosts. A basic reachability analysis creates a matrix, with rows as outbound interfaces on all hosts and columns as all active inbound ports on all hosts. The analysis evaluates each entry in this matrix with respect to the network's topology, routing, and filtering policies to determine if a logical connection is possible. Our current analysis supports end-point or personal firewalls, boundary firewalls, gateways, and subnets and assumes stateful firewalls. The most common usages of source and destination NAT rules are also supported. We plan to improve the handling of destination NAT (DNAT) rules in the next version of NetSPA to support more complex DNAT situations that can occur when traversing multiple firewalls.

The reachability matrix representation has been used for the simple hand-generated networks analyzed in [18]. Although this is complete and correct in principle, it rapidly becomes a performance bottleneck as the size of the network increases. If H is the number of hosts in a network and P is the maximum number of inbound TCP and UDP ports or services per host, then the size of the matrix is bounded by H^2P. In a network with 1000 hosts with 20 TCP ports maximum, this can require 20 million complex reachability analyses. Each analysis involves an examination of firewall or router rules, and if there are R total rules across all filtering devices, the computation required is bounded by H^2PR operations. Complex cases which offer multiple physical paths between two hosts and degenerate cases which loop traffic back through filtering devices multiple times may perform even worse than H^2PR; in both cases, rules may be considered multiple times for a given reachability calculation.

4.1 REACHABILITY DOMAINS

Over the past year, we have developed two major improvements to the basic reachability matrix approach that dramatically improve performance by allowing us to avoid computing some of the matrix. The first improvement involves grouping hosts with similar reachability matrix entries. We do this by separating the traditional reachability matrix into multiple submatrices for traffic within an individual subnet and traffic that crosses between subnets. Figure 4a shows a reachability matrix for a small hypothetical network with two subnets separated by a firewall. Each row in the matrix indicates a source interface, and each column indicates a destination port. Each cell of the matrix potentially holds a Boolean value, indicating whether or not reachability exists between the source interface and the destination port. In this case, we list interfaces and ports in the first subnet before interfaces and ports in the second. This allows us to visualize the matrix more easily. The upper left and lower right submatrices represent intra-subnet reachability, and the remaining two submatrices are inter-subnet reachability.

Hosts that are fully interconnected without any intervening filtering are placed in unfiltered reachability domains. For example, our system forms unfiltered reachability domains for all hosts in a subnet when there is no filtering between these hosts. Hosts in two subnets connected by a router that

performs no filtering are also placed in the same unfiltered reachability domain. This method allows us to treat submatrices of the reachability matrix as individual rows without losing information, as shown in Figure 4b. This method effectively reduces each unfiltered submatrix of dimensions $M \times N$ to a row of length M. Unfiltered reachability domains are created before reachability is computed. Each subnet is first identified and forms a separate unfiltered reachability domain. These unfiltered reachability domains are then analyzed, and those that are interconnected with gateways that perform no filtering are coalesced into larger unfiltered reachability domains by combining all hosts in the coalesced domains.

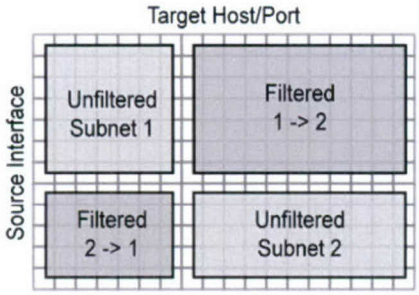


Figure 4a. Example reachability matrix with overlays. Unfiltered boxes indicate reachability that does not cross a firewall (intra-subnet). Filtered boxes represent reachability that is computed across the firewall. A traditional reachability system would need to compute and store reachability information for each cell in the matrix.

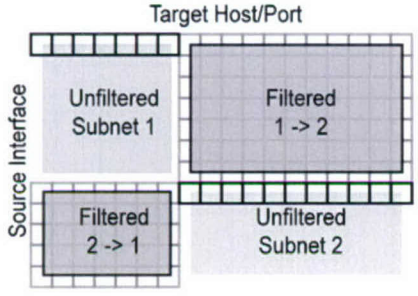


Figure 4b. Example reachability matrix with overlays. Unfiltered reachability is the same for all interfaces within each unfiltered reachability group. We, therefore, compute reachability within the group for one source interface and do not need to compute it for the other interfaces.

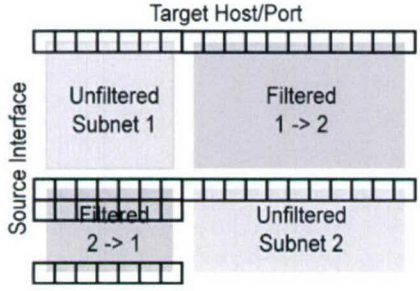


Figure 4c. Example reachability matrix with overlays. Filtered reachability is the same for all interfaces within each filtered reachability group. We, therefore, compute reachability within each group for one source interface and do not need to compute it for the other interfaces.

Hosts that are in the same subnet and are treated identically by all filtering rules are coalesced into filtered reachability domains. These hosts have identical reachability to destinations outside of their unfiltered reachability domains. This method also allows us to treat additional submatrices of the

reachability matrix as individual rows, as shown in Figure 4c. For each set of interfaces in a filtered reachability domain, we need to calculate reachability for only one interface of the set (one row of the submatrix), and all other members of the set share that reachability information. In this example, we show a single filtered reachability domain in subnet one, and three filtered reachability domains in subnet two. Two of those three domains are singletons and apply to only one host. Filtered reachability domains are constructed on demand, as described below.

In most networks, the two types of reachability domains reduce the computation and storage requirements substantially. It is possible to construct a synthetic degenerate network that has firewall rules specific to each host and that thwarts these simplification efforts, but we do not expect such networks to exist in practice. Assigning interfaces to reachability domains requires a comparatively small amount of computation to analyze firewall and router filtering rules and subnets. Our experience is that in many networks this approach reduces computation and memory requirements required to compute reachability by two orders of magnitude, making attack graph generation on large networks practical. For example, in the two-subnet matrix in Figure 4, if the subnets each contain 200 hosts and each host has one open port, the number of unique entries starts at $400 \times 400 = 160{,}000$ without reachability domains, and could drop as low as 800 with reachability domains.

4.2 ON-DEMAND REACHABILITY AND REACHABILITY DOMAINS

A second improvement to the reachability matrix approach is to compute reachability only on demand as needed to generate attack graphs. Most prior approaches (e.g., [1, 2, 18]) assume reachability between all host pairs is available prior to creating attack graphs. This may waste memory and computation because reachability is only required from hosts that are actually compromised, and in a well-protected network significantly fewer than the H hosts in a network will be compromised.

Unfiltered reachability domains are easy to construct on demand by finding hosts that are either on fully connected subnets or on subnets connected without filtering. Filtered reachability domains are more complex to construct on demand. This process begins when previously uncomputed reachability is required from a source interface X. If reachability has not been computed for any other source interface in X's unfiltered reachability domain, then it is computed and interface X becomes the first member of a new filtered reachability domain. If reachability has been computed for other interfaces in X's unfiltered reachability domain, then there are already filtered reachability domains which X could potentially join. The algorithm examines all of the IP addresses and address ranges in the network's filtering rules. If all of these IP addresses and address ranges treat connections from interface X and connections from another interface Y in X's unfiltered reachability domain identically, then X joins Y's existing filtered reachability domain and uses the domain's pre-computed reachability. Computing this relationship between X and Y is somewhat expensive, but is much simpler than recomputing the underlying reachability; we effectively consider the ruleset R once for each attempted match between Y and a filtered domain, whereas computing reachability from Y would require considering the ruleset R for every potential destination. If no matching filtered reachability domain is found, then X begins its own filtered reachability domain and calculates its own reachability.

4.3 ADDRESS SELECTION FOR REACHABILITY ANALYSIS

Network address translation (NAT) must be handled carefully. Destination NAT makes it possible to create a network such that a given source interface cannot reach a target interface using the target interface's address. Instead, an address present only in a translation rule must be used. Figure 5 illustrates the problem. In order for the attacker host to reach the database server, the attacker host must attempt to contact address 192.168.0.3. However, that address is only present in the address translation rule set of the firewall; it does not appear on any actual host on the network, including the database server. We say that the database server's interface is *occluded* from the point of view of the attacker host.

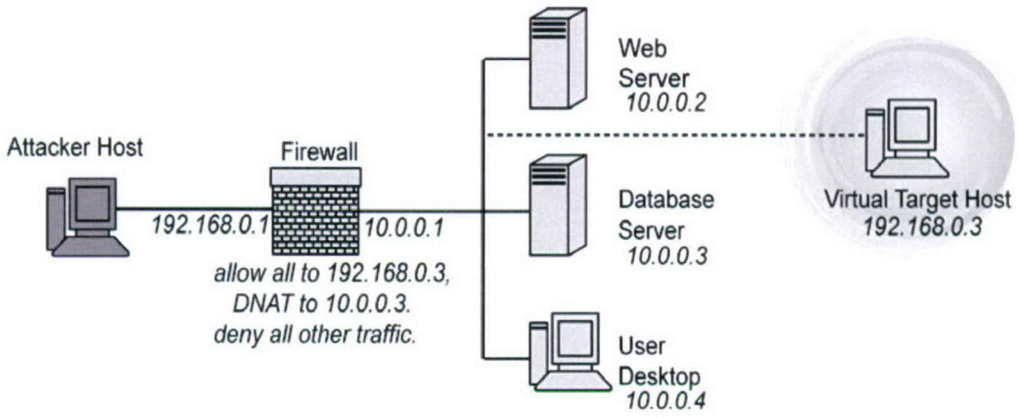


Figure 5. Example of simple network with destination network address translation (NAT). The attacker host is able to contact the database server by trying to connect to the virtual address 192.168.0.3 because this address is translated using firewall NAT rules to the actual address of the Database Server, 10.0.0.3. To make this happen when evaluating connectivity, the NAT address 192.168.0.3 is used as an address on a virtual target host.

We address this issue by fabricating a single virtual host on each subnet in the network. Importer utilities described in section 3.8 generate one virtual host for each NetSPA subnet. A virtual host has no vulnerabilities and no IP address of its own. Each of these hosts then listens on a set of addresses assigned by the address selection algorithm discussed in the next paragraph. We then discover reachability to an occluded interface as a side effect. Attempting to reach a virtual host using the destination address in the rule set will cause the reachability algorithm to exercise the translation rule and reach the occluded interface instead. Figure 5 shows the virtual host holding the necessary address 192.168.0.3. When the attacker host attempts to reach the virtual host via this address, the firewall's translation rule redirects the attacker host to the database server, finding that this server is reachable.

We initially added an exhaustive set of addresses in each virtual host as a simple approach to guarantee that every occluded interface is discovered. This set of virtual addresses included one address from each subnet and one from each address or range of addresses in each filtering, translation, and routing rule. We choose the addresses using an algorithm that selects the specific addresses referenced by the rules and a representative address for each address range referenced by the rules and for each address range used by a subnet. This algorithm first collects all addresses in filtering and NAT rules and all subnet address spaces. These include completely specified 32-bit IP addresses and address ranges. It then sorts these addresses, placing the more specific (e.g., completely specified 32-bit IP addresses) first and the more general (e.g., subnets with more bits masked off) last. The list is then traversed from the more specific to the less specific objects fabricating at most one new virtual address for each list entry. When a completely specified IP address is encountered, it is used as a new virtual address unless this virtual address already exists. When an address range is encountered, a new virtual address within this range is created that does not duplicate any previously selected address.

The addresses selected for the virtual hosts can also be used to the attacker's benefit. The attacker starting location can be an actual host on one of the scanned subnets, or it can be one of the virtual hosts. We refer to a virtual host used as an attacker starting location as a virtual attacker host, and a virtual host not used for this purpose as a virtual destination host. If a virtual host is chosen as the attacker starting location, NetSPA uses all of the virtual host's addresses as potential source addresses for the attacker. In other words, NetSPA assumes the attacker is able to spoof any source IP address to bypass as many network restrictions as possible. A worst-case analysis of network vulnerability should use a virtual host as the attacker starting location. As an option, it is also possible for the attacker starting address to be a single pre-specified IP address. This can reduce computation time when this type of analysis is desired or when IP address spoofing is prevented and incoming connections can only originate from specific IP addresses.

We discovered during testing with simulated networks that the additional computation required by placing all the addresses in virtual hosts can be excessive if there are hundreds or thousands of firewall rules. Each new address in a firewall rule set adds a new source address for the virtual attacker host and a new destination address for the virtual destination hosts. If there are many more virtual host IP addresses than actual hosts, then the number of reachability queries increases as R^2. Each reachability query requires exploration of at most R rules; hence, the computation for every new firewall rule that includes a new IP address grows as R^3. When R is large, this growth in computation can slow down attack graph generation excessively.

We recently modified the address selection algorithm to reduce the impact of using many firewall rules. Instead of using all unique IP addresses and destination NAT addresses found in firewall rules in both virtual attacker and virtual destination hosts, these addresses are split up. Virtual attacker hosts only include unique IP addresses from firewall filtering rules. Virtual destination hosts only include IP addresses from destination NAT firewall rules. This refinement reduces both the number of source addresses for virtual attacker hosts and the number of destination addresses for virtual destination hosts. In many networks the number of real hosts is far greater than the number of NAT firewall rules and extra IP addresses added in virtual destination hosts. The extra IP addresses in virtual destination hosts thus

affect scaling little. These modifications thus lead to reachability computation that scales roughly as R^2 with many firewall rules. It may also be possible to reduce computation further by not extracting IP addresses from firewall rules that deny instead of accept connections.

5. ATTACK GRAPH GENERATION

To our knowledge, no prior paper has demonstrated the ability to generate attack graphs in networks with more than 100 hosts. Many past approaches require extensive human analysis and data entry and appear to scale poorly even though practical use of attack graphs requires analyses on large enterprise networks with many thousands of hosts. This section describes how algorithms were developed that can compute in minutes attack graphs in networks with tens of thousands of hosts. We first describe the types of attack graphs that are created and the purpose of these graphs. Details of the graph-generation procedure are presented in this section. Actual and simulation scaling results on large networks are presented in the following two sections.

5.1 DIFFERENT ATTACK GRAPHS FOR DIFFERENT PURPOSES

As noted above, and shown in Figure 1, we generate attack graphs that show hosts in a network that can be compromised by an attacker starting at a specific location. For example, consider the simple flat network shown in Figure 6. It consists of one host serving as the attacker's starting location (A), a system administrator host (X), and three user desktop workstations (B, C, D). The system administrator host and all workstations have the same remote-to-admin vulnerability (vulnerability 1) that can be exploited directly by the attacker. The desktop workstations have a second remote-to-admin vulnerability (vulnerability 2) that can be exploited only from the system administrator host. For example, this could be an SSH connection only allowed from the administrator host. The three basic types of attack graphs shown in Figure 7 all accomplish the goal of determining hosts that can be compromised on this network, but they have different uses and different scaling properties. We discuss each of the three graphs and present simplified graph-generation pseudocode for each. We simplify the pseudocode by assuming that every vulnerability present in the network is a remote-to-admin vulnerability.

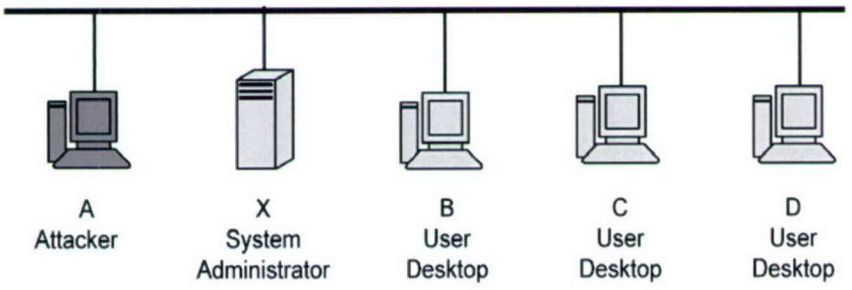


Figure 6. Example of simple network with one attacker host, an administrator host, and three user desktops. Each non-attack host (X, B, C, D) has one remote-to-administrator vulnerability that can be exploited from any other host, and the user desktops (B, C, D) have a different remote-to-administrator vulnerability that can only be exploited from the system administrator (X).

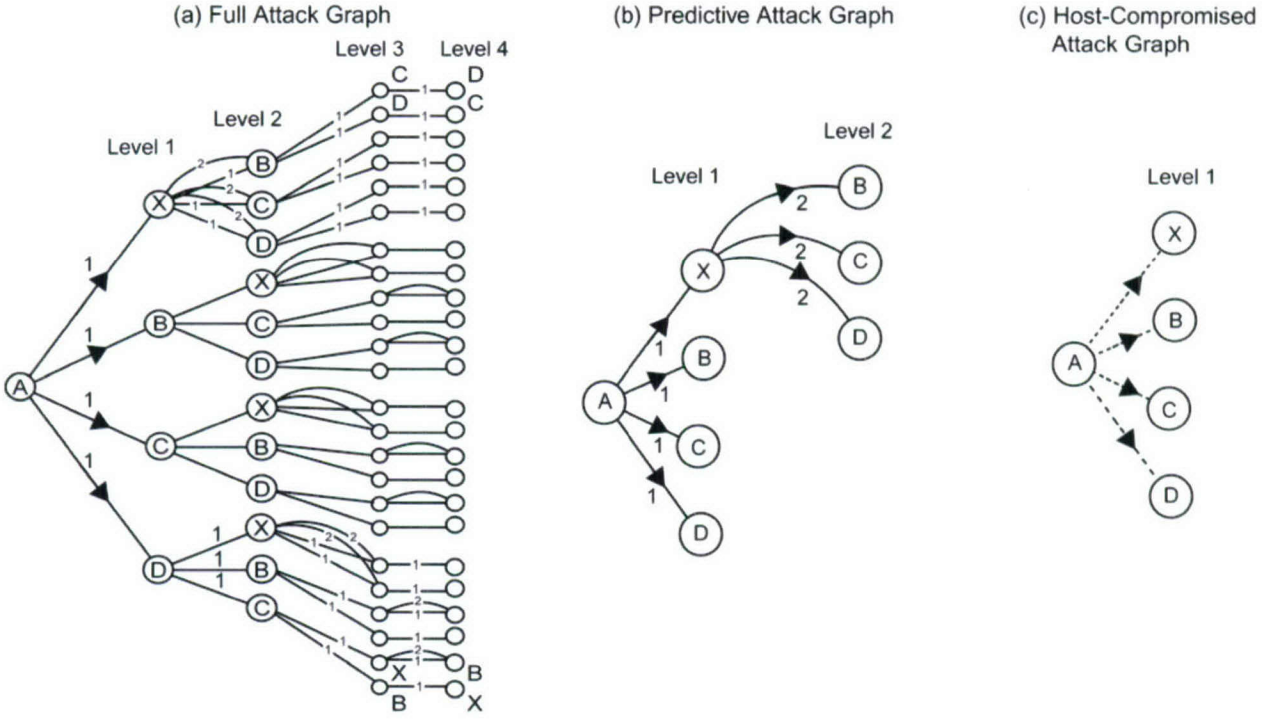


Figure 7. Three types of attack graphs are (a) Full, (b) Predictive, and (c) Host-Compromised. All attack graphs are for the network in Figure 4, and they start on the left with the attacker as an administrator on the attacker host labeled A. Labels for nodes and edges in the full attack graph are provided only for the upper and lower two complete paths for clarity.

5.1.1 Full Attack Graphs

The full attack graph shown on the left of Figure 7 shows all possible paths or sequences of compromised hosts and vulnerabilities that an attacker can use to compromise all hosts in this network. Nodes represent hosts and edges (numbered lines) represent vulnerabilities used to compromise hosts. The attacker starts on the left at the attacker host (node A), and the paths shown represent all possible sequences of attacker actions that compromise as many hosts as possible. This occurs in three levels where the number of hosts at each level exhibits factorial growth from 4 to 12 to 24. The upper two and bottom two paths in Figure 7 are completely labeled to show the sequences of hosts in single attacker paths that compromise all hosts. For example, along the upper path the attacker compromises host X using vulnerability 1, then host B using either vulnerability 2 or 1, then host C using vulnerability 1, and host D using vulnerability 1. The next path is the same except the order of the last two hosts is interchanged and the path is X, B, D, C. Whenever there are multiple vulnerabilities that can be used to compromise a host, a separate line is drawn for each vulnerability. There are thus two edges in the upper

path from host X because the other hosts can be compromised with two different vulnerabilities (vulnerabilities 1 and 2). These additional lines simplify the presentation and formally create a graph instead of a tree.

```
 1 Add NewNode(attacker starting location) to Queue
 2 WHILE (Queue is not empty)
 3  sourcenode <- pop Queue
 4  sourcehost <- the host represented by sourcenode
 5  FOREACH targetport reachable from sourcehost
 6    FOREACH vulnerability on targetport
 7      targethost <- the host that contains targetport
 8      IF no node representing targethost exists on the path
 9         from the root of the graph to sourcenode
10      THEN
11       IF an edge connects sourcenode to a node
12          representing targethost
13       THEN
14          targetnode <- existing node representing targethost
15          add NewEdge(sourcenode, targetnode, vulnerability) to graph
16       ELSE
17          targetnode <- NewNode(targethost)
18          add targetnode to Queue
19          add NewEdge(sourcenode, targetnode, vulnerability) to graph
20       END
21      END
22    END
23  END
24 END
```

Figure 8. Simplified pseudocode to generate a full attack graph.

Simplified pseudocode to generate a full attack graph is shown in Figure 8. The full graph is generated by a breadth-first approach in this pseudocode, but it could also be built depth first. Nodes in the attack graph representing compromised hosts are inserted onto the end of a queue as they are discovered and nodes are taken off the beginning of the queue and used as stepping-stones to try to compromise other hosts. The algorithm begins by placing a node representing the attacker's starting location into the attack graph as the root node and adding that node to the end of the queue (line 1). Nodes are then removed from the beginning of the queue, one at a time (line 3). For each node, reachability is computed to ports on all other hosts (line 5). Whenever a host is reachable via a port, an attempt is made to compromise it using every available vulnerability on the port (lines 5–6). If a vulnerability is exploitable, and the targeted host is not contained in a node on the path from the attack graph root to the current node, then the exploit is added to the graph (lines 8–10).

If a node representing the targeted host is already connected to the current node by an edge, then we add an additional edge representing the additional usable vulnerability (lines 14–15). We otherwise create a new node representing the target first (lines 17 and 19) and also add this new node to the end of the queue for further processing (line 18). This allows the algorithm to attempt further compromise of the

network from that new node when it is removed from the beginning of the queue. The algorithm terminates when the queue becomes empty (line 2).

On a flat network similar to that shown in Figure 7 where there are no filtering devices between hosts and each host has one or more remotely exploitable vulnerabilities, the computation required by this algorithm grows as a factorial of the number of hosts. This occurs because each compromised host at level L in the graph (L edges from the attacker node) can compromise H–L other hosts.

A full attack graph has been generated in many previous studies (e.g., [2, 16, 18–21]). A small segment of a full attack graph could be used to perform computer forensics and determine how a host could have been compromised, given alerts from intrusion detection systems as suggested in [22, 23]. In such a situation, the attacker might have used a long inefficient path to compromise a target host and a full graph might be necessary to find such a path. Unfortunately, it is impractical to generate a full graph even for small networks. For the simple network shown with only two vulnerabilities, the number of nodes in the full graph and the computation grows as $H!$, where H is the number of non-attacker hosts in the network. For example, in subnet with only 10 hosts, the attack graph would already include more than three million nodes. With 10 hosts in the network, one additional host increases the attack graph size and computation requirements by an order of magnitude.

A full attack graph is intractable for almost all real networks, even with only a few vulnerabilities, because it provides more information than is required to determine the security of enterprise networks. Not all paths to compromised hosts are required for this purpose. The minimal information required is whether hosts can be compromised and the privilege level gained by the attacker. This can be provided by a host compromised attack graph.

5.1.2 Host-Compromised Attack Graphs

The right-hand side of Figure 7 shows a host-compromised attack graph for the network of Figure 6. Nodes in a host-compromised graph indicate all hosts that can be compromised by the attacker and the level of privilege gained. Edges indicate one of possibly many sequences of vulnerabilities that can lead to the compromise.

Simplified pseudocode to generate a host-compromised graph is shown in Figure 9. This pseudocode is similar to that used to generate a full graph but simpler. It uses a breadth-first search but adds new edges only once to each node (line 12) and only to hosts that are not already on the graph (line 8).

```
 1 Add NewNode(attacker starting location) to Queue
 2 WHILE (Queue is not empty)
 3   sourcenode <- pop Queue
 4   sourcehost <- the host represented by sourcenode
 5   FOREACH targetport reachable from sourcehost
 6     targethost <- the host that contains targetport
 7     FOREACH vulnerability on targetport
 8       IF no node in the graph represents targethost
 9       THEN
10         targetnode <- NewNode(targethost)
11         add targetnode to Queue
12         add NewEdge(sourcenode, targetnode, vulnerability) to graph
13       END
14     END
15   END
16 END
```

Figure 9. Simplified pseudocode to generate a host-compromised attack graph.

The maximum number of nodes on a host-compromised graph with only remote-to-admin attacks is the number of hosts in the network. An upper bound on the computation required to generate a host-compromised graph assuming reachability has already been computed between all host pairs is $O(H^2)$. This follows from the maximum number of hosts in the graph, H, times the maximum number of hosts that are reachable, H. The computation required to search the current graph and determine if a host is already on the graph (line 8 of Figure 9) can be performed in constant time using a lookup table with a bit for each host indicating whether it is or is not already on the graph.

It is straightforward to extend the host-compromised pseudocode in Figure 9 to include attacks with all the locality and effect categories shown in Table 2. Line 8 of Figure 9 needs to test for the target host existence on the graph but at the privilege level of the vulnerability on the target port (e.g., user, administrator, DoS, or other). In addition, an attacker can proceed forward and a new host is put on the queue on line 11 only if the effect of a vulnerability is to provide user or administrator privilege. Vulnerabilities where the effect is "DoS" or "Other" cannot be used as stepping-stones. They lead to single-host compromises that form one-edge stubs on the graph. Finally, user privilege on a host is a prerequisite for local-to-admin vulnerabilities on the same host.

The host-compromised graph only finds the hosts that can be compromised and one path to achieve the compromise. A single host-compromised graph cannot be used to determine whether eliminating a vulnerability on an edge prevents compromise or whether there are other ways to compromise hosts. Testing any single hypothesis requires rebuilding the entire host-compromised graph.

5.1.3 Predictive Attack Graphs

A host-compromised attack graph provides little insight into networks because it only shows a shortest path to each host. Determining the effect of patching vulnerabilities or adding firewall rules

requires regenerating the graph because there may be many paths to hosts. A full attack graph provides more information about the effect of vulnerabilities, but it provides too much information and is impractical for all but the smallest networks. After exploring a number of alternative graph types and generative algorithms, we developed a graph with two critical properties. First, it determines all hosts that can be compromised for an attacker from a given starting location. Second, without regenerating the graph, it correctly predicts the effect of patching vulnerabilities and adding firewall rules that block specific TCP/UDP ports. Each edge in an attack graph corresponds to a specific vulnerability and network path, and each node corresponds to a host. The effect of removing vulnerabilities and blocking ports can be determined using a predictive graph by removing the associated nodes and edges from that graph. Because this does not involve regenerating the graph, but simply editing the graph, we call a graph that correctly predicts the effect of patching and blocking ports a predictive graph. A predictive graph must make correct predictions when any combination of edges is removed.

The center graph in Figure 7 shows a predictive attack graph for the network of Figure 6. As in the full graph, nodes represent hosts in the network which the attacker has compromised, and edges represent vulnerabilities the attacker could use to achieve the compromise. However, all of the redundant paths have been removed. The remaining structure fulfills the predictive requirement. For example, consider an analysis of the network in Figure 6 to determine the effect of patching vulnerability 1 on host B. Designate this edge/host pair as E(1, B) where, more generally, E(v, h) is the edge with vulnerability v terminating on host h. If every E(1, B) edge/host pair is removed from the predictive graph, then the graph correctly shows that B is still vulnerable due to the path A–E(1, X) – E(2, B) starting from the attacking host. The full graph could also be used to make this prediction by also removing all E(1, B) edge/host pairs from this graph. Unfortunately, it is infeasible to create full graphs for all but the smallest networks.

Simplified pseudocode to generate a predictive graph is shown in Figure 10. The only difference between this code and the code for a full graph, shown in Figure 8, is the dynamic pruning performed on lines 11–13. In a full graph, a new edge/host pair is added to a source node already on the graph only if no node representing the new host already exists on the path from the root of the graph to the current source node. In a predictive graph, there is an additional requirement. A new edge/host pair is added only if this pair is not already attached to the root of the graph or to any node along the path from the root to the current source node. This requirement means that an attacker starting at the root of the graph and following the path to the current source node has not yet used the vulnerability under consideration against the new host. Dynamic pruning makes it possible to remove redundant structure by not duplicating compromises that the attacker has already performed earlier in the attack graph.

```
 1 Add NewNode(attacker starting location) to Queue
 2 WHILE (Queue is not empty)
 3  sourcenode <- pop Queue
 4  sourcehost <- the host represented by sourcenode
 5  FOREACH targetport reachable from sourcehost
 6    targethost <- the host that contains targetport
 7    FOREACH vulnerability on targetport
 8      IF no node representing targethost exists on the path
 9         from the root of the graph to sourcenode
10        AND
11         no node on the path from the root of the graph
12         to sourcenode successfully uses vulnerability to attack
13         targethost
14      THEN
15       IF an edge connects sourcenode to a node
16          representing targethost
17       THEN
18          targetnode <- existing node representing targethost
19          add NewEdge(sourcenode, targetnode, vulnerability) to graph
20       ELSE
21          targetnode <- NewNode(targethost)
22          add targetnode to Queue
23          add NewEdge(sourcenode, targetnode, vulnerability) to graph
24       END
25      END
26    END
27  END
28 END
```

Figure 10. Simplified pseudocode to generate a predictive attack graph.

The effect of dynamic pruning is best illustrated by example. Consider a partially constructed full graph of Figure 7A where hosts X, B, C, and D have already been compromised from host A. The predictive graph will construct the same structure as the full graph up to level 1 because dynamic pruning and full graph conditions for adding an edge/host pair are identical from the root node. When adding structure to level 1, a predictive graph examines the same edge/host pairs as were added to the full graph but does not add all the structure shown in Figure 7A. For example, consider the three edge/host pairs E(1, X), E(1, B), and E(1, C) attached to the node labeled D at the bottom of Level 1 of the full graph. These edge/host pairs are not added to the predictive graph because they already are attached to the root node of the graph. Similarily, the edge/host pairs attached to nodes B and C on Level 1 of the full graph are not attached to the predictive graph. The three edge/host pairs E(2, B), E(2, C) and E(2, D) are attached to node X at the top of level 1 of the predictive graph because these edge/host pairs are not attached to the root node. The additional edge/host pairs attached to these three on levels 3 and 4 of the full graph are not added to the predictive graph because they already are attached to the root node of the predictive graph.

Dynamic pruning constructs true predictive graphs only for vulnerabilities which yield the same effect regardless of when an attack is carried out. Without loss of generality, predictive graphs model attackers who exploit vulnerabilities as soon as possible. Predictive graphs do not need to show every possible order in which attackers could compromise networks. They only need to show necessary prerequisites for available exploits.

We can, therefore, safely prune redundant structure without losing content. If an attacker is able to successfully use a given exploit E(v, h) from a particular node in the graph, there is no need for any child of that node to use the same exploit E(v, h) again. The earliest possible use of E(v, h) on a given path from the root to a leaf will enable at least the same additional exploits as any later use of E(v, h) would. We can therefore omit all uses of E(v, h) from a given node in the graph if any ancestor has already used E(v, h). This type of pruning results in graphs that correctly predict the effect of removing one or more vulnerabilities, which corresponds to removing one or more graph edges. These graphs retain enough information to capture the dependencies an attacker faces when attempting to compromise a network. A full graph clearly satisfies this condition by representing every possible attack from every possible location. A predictive graph does the same by only removing structure that is not necessary for the representation of prerequisites. A node N will have edge E(v, h) dynamically pruned if an ancestor of N has already used E(v, h) because that ancestor is able to effect the compromise with the same set of prerequisites or less.

It is this prerequisite requirement which leads us to prune only when an ancestor has used the attack, not when any arbitrary node has used the attack. If node N is about to add E(v, h) to the graph, and the only other node M which has used this attack is not an ancestor of N, the edge is not pruned. The prerequisites necessary to reach M may not be a subset of the prerequisites necessary to reach N. If we were to prune in this case, we would be building a graph similar to a host-compromised graph, and host-compromised graphs are not predictive.

For example, in the full graph of Figure 7a consider the subgraphs that extend from the right of the node labeled D at the bottom of level 1. There are three subgraphs starting from that node, beginning with E(1, X), E(1, B), and E(1, C). Each of these subgraphs is contained in the corresponding subgraphs that begin earlier from node A on the left of the graph and start with the same edge/host pairs. For example, the subgraph starting at level 1 from node D on edge E(1, C) contains a subset of the subgraph starting from node A on edge E(1, C). This smaller later subgraph includes only two nodes (X and B) after its first link while its corresponding larger, earlier graph contains three nodes (X, B, D). Furthermore, the nodes in common to these subgraphs (X and B) are compromised using the same vulnerabilities. Eliminating any edge/host pair in the earlier and larger subgraph prevents access to the same hosts as eliminating the same edge/host pair in the smaller, later subgraph. The later subgraph is eliminated in a predictive graph. The rules used to build full graphs guarantee that later subgraphs following the same edge/host pair in an attack graph are identical to or contained in earlier subgraphs because these rules can only eliminate nodes in later subgraphs and not add nodes.

Dynamic pruning is made possible by the simple method used to model vulnerabilities in NetSPA. This fact, in combination with dynamic pruning rules, make predictive graphs efficient to compute in

NetSPA. As described below, more complex and less common vulnerabilities might require more complex pruning rules to build predictive graphs.

As with a host-compromised graph, it is possible to extend the predictive-graph pseudocode in Figure 10 to include attacks with all the locality and effect categories shown in Table 2. Line 8 of Figure 10 needs to test for the target host, but at the privilege level of the vulnerability on the target port (e.g., user, administrator, DoS, or other). In addition, an attacker can proceed forward and a new host is put on the queue on line 22 only if the effect of a vulnerability is to provide user or administrator privilege. As with the extended host-compromised graph, vulnerabilities in which the effect is "DoS" or "Other" cannot be used as stepping-stones, and user privilege on a host is a prerequisite for local-to-admin vulnerabilities on the same host.

Most attacks used on real networks satisfy assumptions required to construct predictive graphs and have the same effect whether they occur before or after other attacks. There are a few important exceptions. First, DoS attacks affect following attacks because they disable services on hosts. These attacks reduce the subgraph following a host compromise by preventing further compromises. The effect of such attacks is modeled correctly by predictive graphs because they reduce and do not extend subgraphs. Second, some types of information-gathering attacks affect following attacks. In one type of information-gathering attack, an attacker compromises one primary host and steals and then decrypts passwords that provide access to other secondary hosts. If all secondary hosts are directly reachable from the primary host containing the passwords, then these attacks are also correctly modeled by the predictive graph algorithm. The subgraph extending from the primary host contains at most all the secondary hosts independent of where in the graph the compromise occurs. This is not true for information-gathering attacks in which secondary hosts are not reachable from the primary host containing passwords. NetSPA does not currently model these types of attacks. We are exploring future approaches to handle this situation using predictive graphs when and if this information becomes available. A final class of attacks that affects following attacks are those against network filtering devices including routers, firewalls, and intrusion-prevention systems. Attacks that make filtering more restrictive can be modeled by predictive graphs in the same way as can denial-of-service attacks by adding the attack as a dead-end edge that leads nowhere. Some attacks that make filtering less restrictive and make new hosts reachable from hosts already on an attack graph are not currently modeled by predictive graphs. For example, an attacker may add a new firewall rule to allow access from a normally blocked outside IP address. We are exploring future approaches to handle these types of attacks with predictive graphs.

It is difficult to create computation bounds for predictive graphs because the graph size depends on the network topology. On a flat network, where all hosts are interconnected with no filtering devices, the computation required to compute a predictive graph is $O(H^2 \log_2 H)$, where H is the number of hosts. This is larger than the general bound for a host-compromised graph because the search indicated in lines 10 to 14 of Figure 10 is bounded by $O(\log_2 H)$. The time required to compute a host-compromised graph on a flat network also depends on the number of hosts that can be compromised. If only a fixed number of hosts can be compromised, then the computation grows linearly in N. Section 8.3 discusses flat networks in greater depth. The next section describes a situation in which the computations required for a predictive graph with filtering devices can become much larger than this.

Host-compromised graphs are a potential alternative to predictive graphs. As noted above, they are generally not useful for making predictions because they include only one of possibly many paths to compromise each host, requiring the system to rebuild the graph from scratch to test any one hypothesized change to the network. With a limited number of edge/host pairs to explore, this would be practical because the computation required for a host-compromised graph grows only as $O(H^2)$. It becomes impractical when searching for combinations of edge/host pairs to patch in large networks with many vulnerabilities when the goal is to find a set that provides the greatest protection but requires patching the fewest hosts. With V vulnerabilities and N nodes, exploring combinations of all edge/host pairs in the worst case would require generating a number of host-compromised graphs that grows combinatorially as NV. Such an approach is clearly impractical.

5.1.4 Node-Predictive Attack Graphs

Predictive graphs occasionally produce large amounts of seemingly redundant structure. As an example, consider the simple network on the left of Figure 11, consisting of an Attacker Host (A), a firewall, a Web Server (W), a Database Server (D), and 200 User Desktop hosts (U). Every host behind the firewall has a single remote-to-admin vulnerability. The firewall allows the attacker to directly compromise the web server and the database server, but not the user desktops. This scenario's predictive attack graph is shown in the middle of Figure 11.

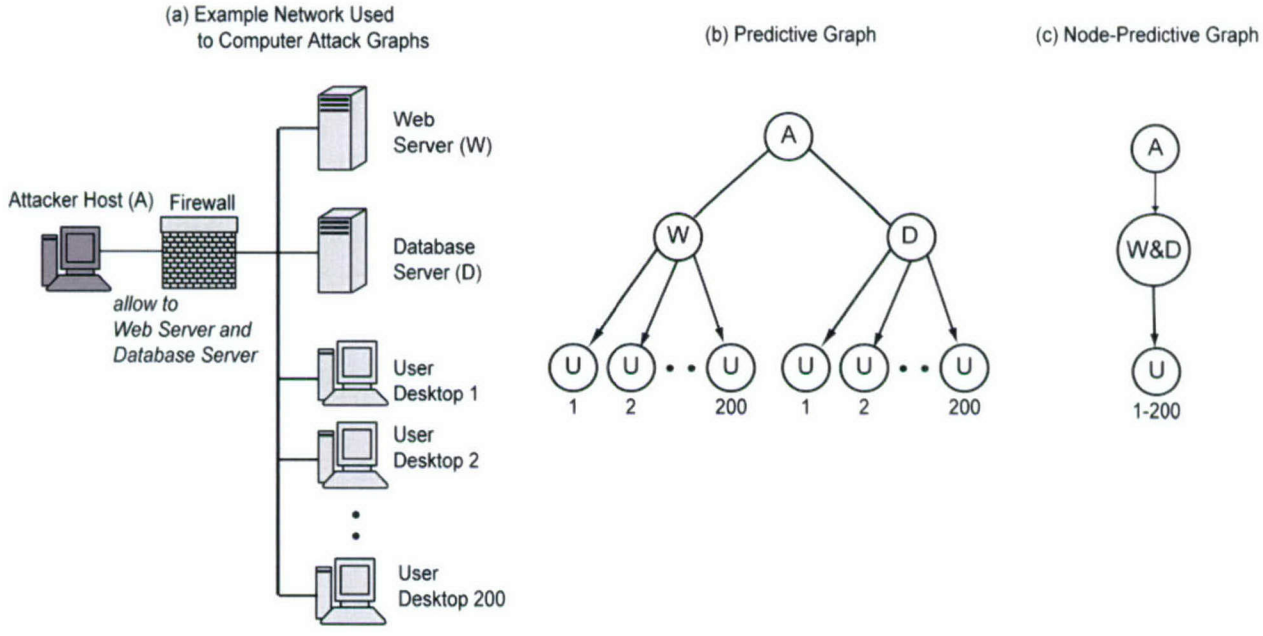


Figure 11. Example of simple network used to compute attack graphs (a). The predictive graph (b) has repeated structure. The attack on the 200 user desktops is shown from both the web server and the database server. The node-predictive tree (c) simplifies this topology. The web server and the database server are collapsed into a single host group labeled W&D, and all user desktops are collapsed into a single host group labeled U.

This predictive graph is accurate. It correctly predicts the effect of patching or protecting either the web server or the database server independently. It also correctly predicts the effect of patching other hosts. Unfortunately, there are now 400 nodes on the attack graph representing the user desktop hosts instead of 200, and the subgraphs below nodes W and D are duplicated. Firewall filtering has caused redundant subgraphs in the predictive graph by breaking the attacker's action into two stages. First, a direct-compromise stage compromises hosts directly reachable through the firewall. Second, an indirect-compromise stage uses directly compromised hosts to attack the rest of the vulnerable hosts behind the firewall. To remain predictive, the subgraph that represents indirect compromises must be duplicated.

It can be shown that the number of nodes in a predictive graph is greatest with one firewall if the number of directly and indirectly compromised hosts is equal. Consider a network similar in structure to the network of Figure 11 with H vulnerable hosts where DC hosts can be directly compromised through the firewall and thus $H - DC$ hosts are indirectly compromised. The number of nodes in the attack graph (excluding the root node) is $DC(H - DC)$ because each of the directly compromised hosts compromises the remaining $H - DC$ hosts. This equation is maximized when $DC = H/2$ and there are $(H/2)^2$ nodes. Even a subnet with 200 hosts could produce a graph with 10,000 nodes if 100 hosts can be compromised directly through the firewall.

If there are multiple firewalls or filtering devices between an outside attacker and an internal protected subnet, the graph size can increase in a similar fashion after each filtering device. For example, Figure 12 shows an example with two firewalls between an outside attacker and protected hosts on internal subnet B. This two-firewall topology occurs at many sites that put external servers in a DMZ (demilitarized zone) and internal hosts in a more protected subnet. Although this usually is accomplished using one physical firewall with three network interfaces, two firewalls are shown to illustrate locations where network filtering occurs. Each subnet in Figure 12 contains three hosts that can be directly compromised through each firewall from hosts on the left of the firewall and three that can be indirectly compromised. To reach hosts on the innermost subnet B, an outside attacker must pass through firewall A, compromise stepping-stone hosts in subnet A, and then pass through firewall B. Only the three indirectly compromised hosts in subnet A are allowed to connect through firewall B to subnet B. The left of Figure 13 shows a predictive attack graph for this network. It can be seen that the number of nodes is multiplied by 9 after each firewall. The left node in this graph is the attacker host, the first column of three nodes represents the directly compromised hosts in subnet A, the next column of 9 nodes represents the three indirectly compromised hosts in subnet A being compromised three times each. Each of these 9 nodes then directly compromises three hosts each in subnet B through firewall B and each of these compromises three hosts indirectly. In this worst-case topology, a network with only 12 nodes creates an attack graph with 120 nodes.

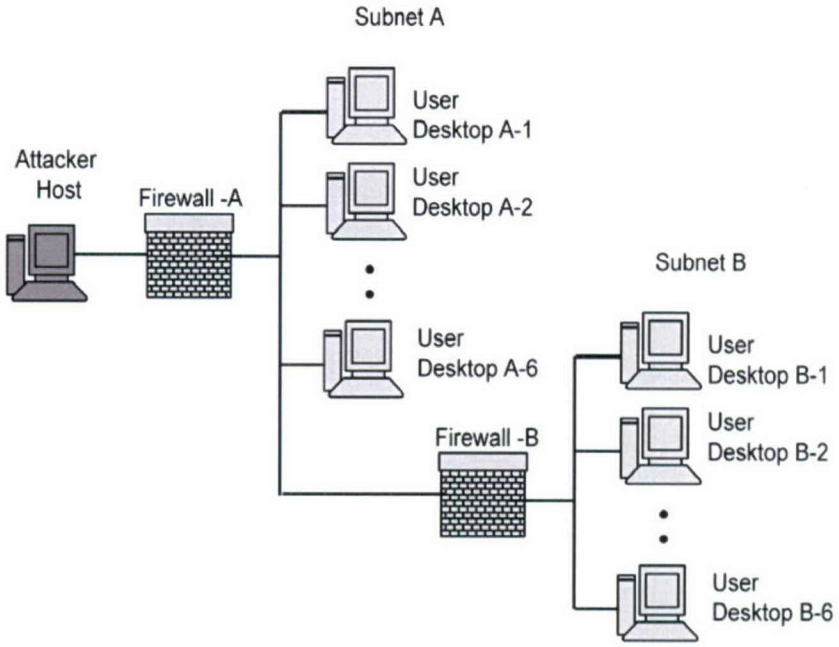


Figure 12. A network designed to create the largest firewall explosion possible with 12 hosts. Two subnets contain three hosts that can be directly compromised through each firewall from hosts on the left and three that can be indirectly compromised from these hosts. To reach hosts in the innermost subnet B, an outside attacker must pass through firewall A, compromise hosts in subnet A, and pass through firewall B. Only the three indirectly compromised hosts in subnet A are allowed to connect through firewall B to subnet B.

The above analysis can be extended to more than two filtering devices. If L is the number of levels or number of filtering devices in a network, and each subnet contains the same number of nodes (H / L), then predictive graph size is maximum when $L = H / 2e$ and $DC = H / 2L$, where L and DC must be rounded to a nearest integer. When these equations are satisfied, then the number of nodes in the graph grows exponentially in H and is bounded by $O(e^H)$. Most networks include at most one to three filtering devices in series and not the large numbers required to achieve exponential graph growth. Even with two filtering devices, however, the size of a predictive graph grows as a fourth power of the number of hosts. This type of growth is impractical to model for large networks when many hosts can be compromised. Further examples of this phenomenon, which we call a *firewall explosion*, are provided in Section 8.

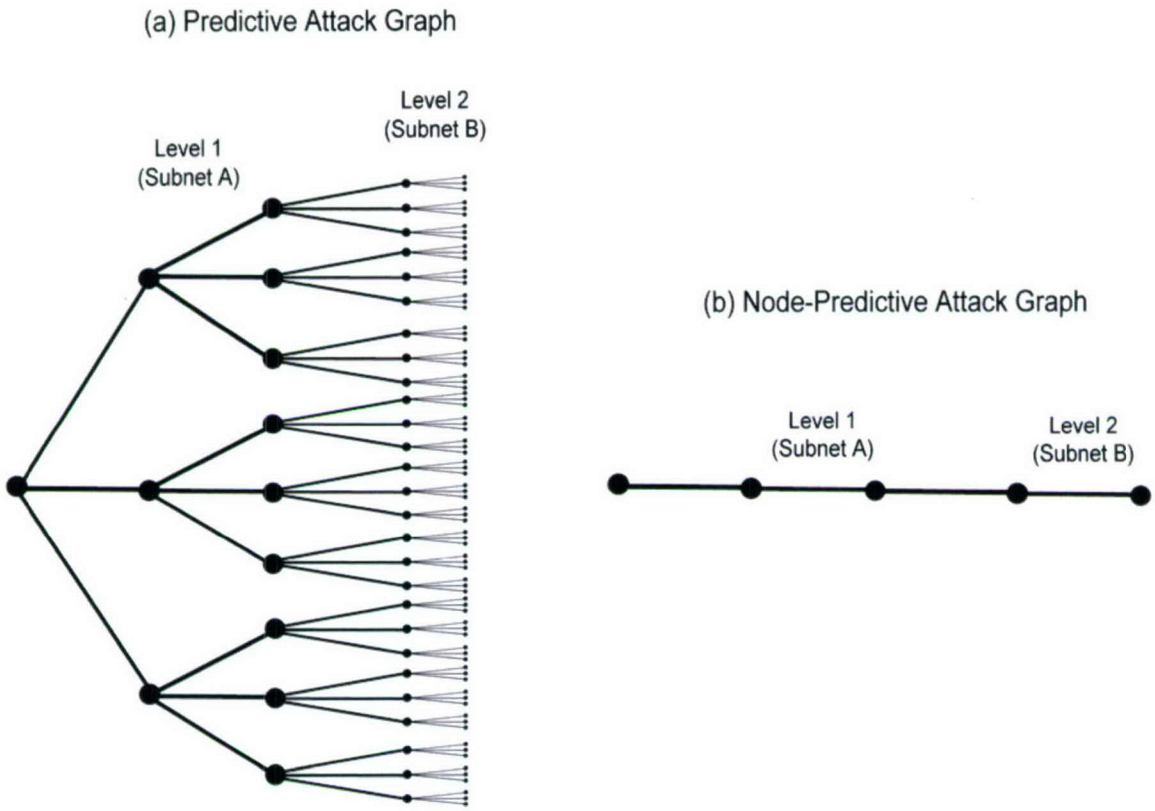


Figure 13. The predictive attack graph (a) is for the network in Figure 9, and it illustrates the effect of two levels of firewall explosion. Each firewall multiplies the number of paths in this graph by 9, leading to a total of 120 nodes. This expansion could continue if there were more subnets and firewalls. The left-most node in the attack graph represents the attacker host; the next two columns of nodes represent hosts in subnet A and the first level of the firewall expansion; and the next two columns of nodes represent hosts in subnet B and the second level of firewall expansion. A node-predictive attack graph (b) is much simpler. Nodes on this graph from left to right represent the attacker host, directly compromised hosts in subnet A, indirectly compromised hosts in subnet A, directly compromised hosts in subnet B, and indirectly compromised hosts in subnet B.

A variant of a predictive graph, called the node-predictive graph, prevents attack graph size from growing exponentially. A node-predictive graph accurately predicts the effect of completely patching any combination of nodes, where nodes represent either individual hosts or groups of hosts. It also predicts the effect of patching individual vulnerabilities on all hosts except for a select set of hosts that are grouped together using a technique called Dynamic Host Collapse (DHC). Node predictive graphs are built using the predictive graph pseudocode but after incorporating DHC. DHC places hosts into host groups when they have equivalent inbound compromisability. In other words, if two hosts can be compromised at the same level (user, administrator, other, DoS) from every host the attacker has already compromised, then the hosts are equivalent from the point of view of the attack graph. This also means that if you can compromise one of the hosts in a host group, then you can compromise all of them.

A host group is treated like a single host by the attack graph generator. Host groups are consistent across the graph as well. If hosts A, B, and C are in host group G, for example, then every occurrence of G in the graph represents A, B, and C. Additionally, none of the three hosts A, B, and C appear in any other group. Singleton groups are also possible.

An example of a node-predictive graph for the network in the left of Figure 11 is shown at the far right of Figure 11. This graph shows the attacker directly compromising a host group labeled W&D containing both the web server and the database server. From there, the attacker compromises all 200 user desktops shown as a single host group labeled U. The redundant subgraphs shown in the predictive graph in the middle of Figure 11 don't occur in the node-predictive graph, and hosts have been placed into two groups. The right side of Figure 13 shows a node-predictive graph for the network of Figure 12. This graph is also much smaller than a predictive graph and contains the attacker host and four host groups. Two hosts groups contain three directly compromised hosts from subnets A and B and two contain three indirectly compromised hosts from these subnets. The dramatic reduction in graph size provided by node-predictive graphs comes at the expense of losing some details concerning attacker actions. The graph no longer differentiates between attack paths that compromise the web or database servers or which hosts in a host group enable further compromise. As a result, we can no longer make recommendations based on the removal of arbitrary edges in the attack graph. We can only make recommendations based on removal of nodes that represent host groups, of edges that do not terminate on host groups, and of other nodes that represent single hosts.

Node-predictive graph generation is begun by placing hosts into the largest possible host groups, assuming the groups are correct, and then splitting groups when the attack graph discovers counterexamples. All of the hosts in an unfiltered reachability domain join the same host group, with a few exceptions designed to ease implementation. For example, gateways between subnets are placed in singleton groups. Graph generation is the same as with a predictive graph but with several modifications as shown in the simplified pseudocode of Figure 14. We will discuss the modifications out of order, illustrating the basic technique first, followed by the other necessary adjustments.

The attack graph generator operates normally, originating attacks from a source node to ports on individual hosts and producing graph nodes and edges to represent successful attacks. However, these resulting nodes are not placed directly into the graph or into the queue. Instead, the DHC algorithm evaluates them for counterexamples—cases in which some, but not all, of the hosts in a given host group are compromised. If no counterexamples are found, the algorithm replaces all of the target nodes with nodes for the appropriate host groups instead. Each edge, instead of pointing to a node representing the target host, points to the node representing the target host group. These host group nodes are then added to the end of the queue and the algorithm proceeds normally.

Notice that nodes representing host groups are placed on the queue, as ordinary attack graph nodes. When reachability is evaluated for such a node (line 5), the reachability algorithm computes the reachability for every host in the host group and presents the graph-generation algorithm with the union of that reachability. The graph generator then attacks hosts reachable from the host group, without knowing which host or hosts within the group make each particular attack possible.

```
1 Add NewNode(attacker starting location) to PriorityQueue
 2 WHILE (PriorityQueue is not empty)
 3  sourcenode <- pop PriorityQueue
 4  sourcehostgroup <- the host group represented by sourcenode
 5  FOREACH targetport reachable from sourcehostgroup
 6    targethost <- the host that contains targetport
 7    FOREACH vulnerability on targetport
 8     IF no node representing targethost exists on the path
 9        from the root of the graph to sourcenode
10     THEN
11       IF
12        no node on the path from the root of the graph
13        to sourcenode successfully uses vulnerability to attack
14        targethost
15       THEN
16        IF an edge connects sourcenode to a node
17           representing targethost
18        THEN
19          targetnode <- existing node representing targethost
20          add NewEdge(sourcenode, targetnode, vulnerability) to graph
21        ELSE
22          targetnode <- NewNode(targethost)
23          add NewEdge(sourcenode, targetnode, vulnerability) to graph
24        END
25       ELSE
26        record the attack as a "potential attack"
27       END
28      END
29    END
30  evaluate the new targetnodes to see if there are any
31    counterexamples to our host groups
32  IF there are incorrect host group(s)
33  THEN
34    FOREACH incorrect host group
35      FOREACH occurrence of the host group in the graph
36        grpnode <- attack node which represents the host group
37        DELETE all edges inbound to grpnode
38        put grpnode's parent onto the PriorityQueue
39        DELETE grpnode and any graph structure beneath it
40      END
41      create the correct host group(s) and remove the incorrect one
42    END
43  ELSE
44    replace the targetnodes with nodes representing host groups
45    put the targetnodes into the PriorityQueue
46  END
47 END
```

Figure 14. Simplified pseudocode to generate a node-predictive attack graph.

The algorithm becomes complicated when a counterexample is found (lines 30–32). Imagine, for example, that there is a host group containing hosts A, B, and C. If a source node in the attack graph is able to compromise hosts A and B at the administrator level, but cannot compromise C, this shows that the host group is incorrect. The group must be split into two new host groups, one containing A and B (both compromisable at the administrator level), and the other containing only C (not compromisable). We must also make sure the group is split everywhere it has already been used; there may be nodes elsewhere in the attack graph which already use the original group. The algorithm, therefore, walks the attack graph, looking for nodes representing the original group (lines 34–35). Whenever such a node is found, it is destroyed, along with all inbound edges and all of the graph structure beneath it (lines 36–37, 39). The node's parent is then entered into the priority queue (line 38) and the subgraph is rebuilt using the new groups.

Support for subgraph deletion and reconstruction requires a priority queue. Nodes within the queue are prioritized by depth in the graph. This does not change the breadth-first nature of the graph generation. The priority queue allows the algorithm to easily rebuild the deleted section or sections breadth-first only to the depth of the rest of the graph. It then proceeds breadth-first on the entire graph from that point on.

A final alteration is necessary to prevent erroneous splits. The predictive graph's pruning algorithm prevents child nodes from using attacks already used by the parent node. However, this can cause problems with DHC. Imagine again the host group containing A, B, C. Now imagine host D is able to compromise all three at the administrator level via vulnerability 1, as well as host E. Next, host E is able to compromise B via vulnerability 2 and can still compromise A and C via vulnerability 1. The pruning algorithm would keep only E's compromise of B via vulnerability 2; the attacks on A and C are redundant because the parent node (representing host D) has already successfully used those attacks. This gives the grouping algorithm incorrect information; according to the graph, E can compromise B, but not A or C. This would force an incorrect group split. We therefore track "potential attacks" which would normally be removed by the predictive pruning algorithm (line 26 in Figure 10). The DHC algorithm is then able to track all of E's potential attacks and correctly see that the group containing A, B, and C should remain intact (and, furthermore, the attack graph should show E attacking a node representing that group). The node-predictive pseudocode can be extended to support all of the attack types shown in Table 2 in much the same way as the predictive graph pseudocode. Additionally, the code must be able to split one group into as many as three groups. Separate groups are required for members that cannot be compromised, those that can be compromised at user level, and those that can be compromised at admin level.

The node-predictive algorithm is very complex and has many opportunities to underperform. Splitting groups is a very complicated and involved process, and the optimistic nature of the algorithm all but guarantees that splits will be necessary. However, we have found this algorithm performs well in practice. Further examples of the dramatic reduction in graph size and graph-building time provided by this approach compared to predictive graphs are presented in Section 8 on simulations. The primary disadvantage to the algorithm's node-predictive graph is the loss of granularity; the graph can no longer be used to make recommendations concerning individual hosts in a host group. We have some ideas for increasing the analytic value of a node-predictive graph by tracking additional information during its construction, and intend to pursue this in future work.

6. AUTOMATED RECOMMENDATIONS

Attack graphs quickly become too large for hand analysis even though a predictive graph is much smaller than a typical full graph. We have developed an analysis system that automatically extracts information from the graph and presents it to an administrator as an easily understood, prioritized list of recommended changes to the network. This allows an administrator to first apply patches that provide the greatest reduction in the NCP or network compromised percentage described in Section 3.7.

The analysis system hypothesizes patches to hosts and evaluates their effects on the network's security. Each hypothetical patch or group of patches is called a recommendation. We make our recommendations host-based, rather than vulnerability-based, in order to offer recommendations that are easy for an administrator to understand and implement. A recommendation to patch three vulnerable services on Host A, for example, is usually more practical than a recommendation to patch vulnerability X everywhere it occurs in the target network.

Each recommendation has an associated NCP value. Whenever possible, we compute a recommendation's value using the predictive graph. If a predictive graph is too large and cannot be generated, we use a node-predictive graph. With a node-predictive graph, we use the same algorithm except each host group is treated as a host. We first remove the edges that target the host or hosts the recommendation suggests patching. Once the edges are removed from the predictive graph, we can again compute the total asset percentage obtained by the attacker. The predictive graph allows us to hypothesize a set of changes to the network's vulnerabilities and predict the results of those changes without rebuilding the graph. The recommendation's value is the difference between the original and new NCPs.

We first formulate recommendations for individual hosts, examining each host one at a time. Recommendations are ordered by determining the impact of patching each host compromised in an attack graph. The algorithm recommends patching only the vulnerabilities the attacker could exploit on the given host, which are not necessarily every vulnerability on the host. We compute this list of exploitable vulnerabilities by walking the attack graph and recording all edges that the attacker can use to compromise the target host. We do not explore subgraphs beneath a compromise of the target host.

The individual host recommendations are often valuable because they reduce the list of vulnerabilities that must be patched by focusing only on vulnerabilities which exist and are exploitable from the attacker's starting location. In many networks, however, the impact of patching two or more hosts is much greater than that of patching any single host because the attacker must go through a bottleneck. This bottleneck is often a small set of hosts that can be accessed directly through a border firewall as in the network of Figure 11. An attacker can use this small set of stepping-stone hosts to compromise the rest of the hosts behind the firewall. Patching an individual host in the small set of stepping-stones does not protect the rest of the hosts. We need to be able to formulate and offer a recommendation which calls for the administrator to patch the entire group of stepping-stone hosts.

We, therefore, additionally formulate group recommendations, each representing the impact of patching several hosts at once. We begin at the attacker's starting node. For each child node (each representing a host the attacker could compromise directly), we compute the set of all hosts compromised below the child node. We then collect the child nodes into groups. A child node C joins group G if the group already contains a node D such that the set of hosts compromised below C has a non-null intersection with the set of hosts compromised below D. The algorithm thus forms groups of hosts which are stepping-stones to some common subset of additional hosts. We recursively explore any singleton groups; the single node in the group is treated as the attacker's starting node and the algorithm runs again. The two types of recommendations allow us to find and report both single-host and multiple-host stepping-stones in the attack graph.

The NetSPA system generates its attack graphs and its recommendations in roughly the same amount of time. This is acceptable, but we have some directions for future work which should improve the performance of the recommendation algorithm and increase its fidelity as well.

Because the above system results in at least one recommendation per compromised host, we must present the recommendations to the administrator carefully; we have still generated a large amount of data. The algorithm sorts the collection of recommendations by value and presents them as an ordered list. This presents the recommendations in order of effectiveness; the recommendation which reduces the NCP value the most is displayed first. Only the most effective recommendations are presented. This prevents the system from overwhelming the administrator with every possible recommendation. We present examples of these recommendations in Section 8, in which simulation results are presented.

7. FIELD TRIALS ON THREE ACTUAL NETWORKS

NetSPA has been evaluated on three networks under the guidance and cooperation of system administrators of these networks. One network was at a military site and two were at MIT Lincoln Laboratory. They contained from 200 to 3,400 computers and two contained perimeter firewalls that we were permitted to analyze. Analyses were performed on site on a computer provided by the host organization and not connected to any network to insure the privacy and security of the analysis. We imported firewall configuration files and the results of Nessus or ISS vulnerability scans provided by the host organizations. Agreements with these organizations prevent us from providing analysis details that might compromise the security of these networks. The following two paragraphs summarize the results in a manner that has been accepted for open publication.

For all networks, the analysis required only a few minutes. The initial vulnerability scans typically found vulnerabilities on hundreds of computers. When firewall information was available, NetSPA was able to identify hosts that can be directly compromised through the firewall. Recommendations NetSPA produced successfully reported a small number of "stepping-stone" or "bottleneck" hosts in which vulnerabilities found by the scanner, if present, would enable attackers to penetrate the network. Only these few hosts had to be carefully examined by hand to confirm the presence of the vulnerabilities found by the network scanner. The NetSPA analysis usually reduced the number of hosts that needed to be hand examined from the many hundreds found by the vulnerability scanner to the fewer than ten critical "stepping-stone" hosts. Hand examination discovered incorrect firewall rules, servers that needed to be patched, and false positives from the vulnerability scanner.

In summary, NetSPA successfully imported vulnerability scanner and firewall configuration information and was able to produce attack graphs and make recommendations in only a few minutes for three actual networks with 200 to 3,400 computers. When firewall information was available, recommendations allowed systems administrators to focus on a few critical "stepping-stone" hosts to confirm the presence of vulnerabilities or to focus on a few firewall rules that enabled attackers to penetrate networks from outside a perimeter firewall. After careful review by local analysts of these few hosts, either networks were made secure from outside attacks or it was determined that networks were secure. The ability to focus resources was found to be very valuable, and future tests are planned.

8. SIMULATION STUDIES AND SCALING

This section describes simulation studies designed to explore the performance of NetSPA on medium and large networks with hundreds to thousands of hosts. A rationale for the studies and a detailed description of the simulator is presented, along with seven experiments. Results and analyses accompany each experiment. Experiments demonstrate scaling performance discussed in prior sections of this report and complement results obtained on real networks described in the previous section.

Simulation studies were performed for three main reasons. First, it is easier to create and validate simulated networks than acquire and analyze actual network data. Simulated networks and their characteristics are predefined and can be easily verified. Characteristics of large, complex real networks are difficult to determine, and the dynamic nature of large networks makes verification difficult. Second, real data are sensitive and reveal network weaknesses valuable to an attacker. As noted above, most system administrators require us to perform analyses of real networks on site in a physically protected area and preferably on a computer not connected to any network. They also do not permit release of attack graphs for real networks. None of these restrictions apply to simulated networks. Finally, simulation studies allow analytical evaluation of system performance, including the effects of scaling to very large networks containing tens of thousands of hosts. By generating data, we can selectively examine how computation times vary with changes in the network topology and important network parameters such as the number of hosts and firewall rules.

Two methods could have been used to create simulation data for NetSPA. First, network scans (e.g., from Nessus or ISS), firewall configuration files, and user-supplied text files could be generated by the simulator and imported with the NetSPA importers. Second, as imported data are first written to a database from which the NetSPA binary creates the in-memory model, simulation data could be written directly to the database without attempting to re-create several complex file formats. The second method was chosen for the simulation generators, largely because it avoids the necessity of reproducing Nessus and/or ISS scan files, firewall configuration files, and our own user-information files.

8.1 NETWORK MODELS

Three simplified network models were developed for these studies: (1) a simple collection of fully connected hosts co-located on a single segment, called a *flat network*, (2) an *enclave* defined by a security perimeter protecting several subnets and associated assets, and (3) an *enterprise* comprising two or more enclaves. These models represent a range of network topologies similar to those encountered on real networks. Network parameters such as the numbers of hosts and firewall rules can be varied in these models to explore performance for larger networks. A fourth *customized enterprise* model is also used. It demonstrates NetSPA's ability to discover multistep attack paths where a single vulnerability in one enclave results in the compromise of two other protected enclaves.

Network database entries for each model are created with a Perl script. The script takes a simple set of command-line arguments that define the characteristics of the network model being created. These allow the user to specify the number of enclaves, subnets, *tenants* (a subnet with a firewall between its hosts and the rest of the enclave), hosts, firewalls, and firewall rules, as well as the number of ports and vulnerabilities per host.

8.1.1 Flat Network

A flat network contains only **H** fully connected hosts as shown in Figure 15. Each host has **P** open ports, and each port permits access to a single remote-to-other vulnerability. Port numbers are chosen serially from 1 to **P**. Host addresses are chosen serially from a private class A IPv4 space starting at 10.0.0.1/8. The total host count for the flat network ignores a single virtual host used as an attacker starting location. In addition to the number of hosts and the number of ports, the percentage of compromisable hosts can be specified. Hosts in this compromisable group have their remote-to-other vulnerability on port 1 replaced with a remote-to-admin vulnerability.

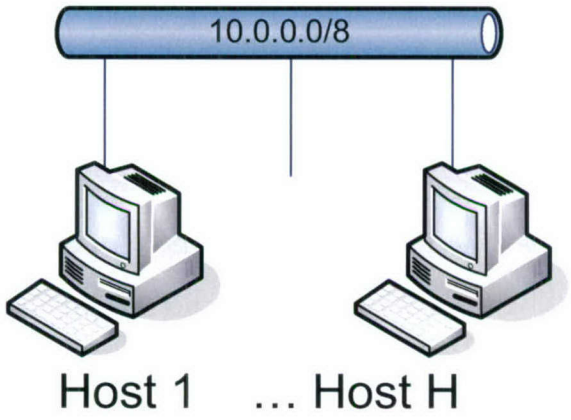


Figure 15. Flat network block diagram.

Since there are no firewalls, only a single subnet, a single attacker, and a simple distribution of vulnerabilities across the network's hosts, this model provides an easy way to explore how computation time scales as the number of hosts and as the percentage of compromisable hosts increases.

8.1.2 Enclave Network

The flat network model is simple and does not contain many of the topological features characteristic of real networks. The *enclave* model, shown in Figure 16, was developed to represent a network at a single corporate or government site. The enclave introduces a security perimeter defined by a perimeter firewall. Internal hosts are separated into a minimum of five subnets via the firewall and an internal router. This model represents a general topology seen at single enclave installations ranging from home office to medium-sized college or corporate networks.

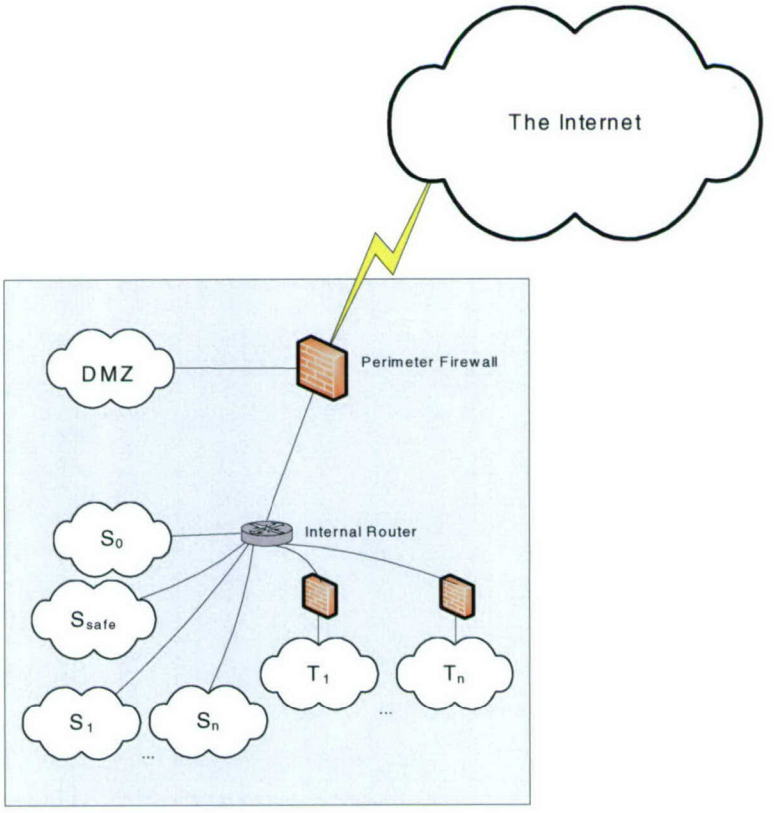


Figure 16. Enclave network block diagram.

In the minimal configuration, an enclave model includes five subnets and two "infrastructure" devices. The first subnet is "The Internet," and uses addresses from the IPv4 space 172.16.0.0/12. The Internet subnet contains a virtual host and nothing else. The enclave perimeter firewall has three interfaces; the first is on "The Internet" subnet, with address 172.16.0.1. The perimeter firewall's second interface is on its own "DMZ" subnet and has address 192.168.0.1, and its third (192.168.0.9) is on a

dummy subnet which connects the perimeter firewall to the internal router. This subnet is required to create a point-to-point link between the firewall and router because NetSPA's run-time network model does not yet include such links.

The internal router connects, at minimum, two subnets per enclave, known as "Subnet Zero" and "Subnet Safe." Subnet zero (S_0) has between zero and 254 hosts using addresses taken serially from the IPv4 space 10.0.0.0/24. The internal router's interface on S_0 uses the first address, 10.0.0.1, on that subnet. Subnet Safe (S_{safe}) contains between zero and 254 hosts which **never** contain any vulnerabilities. S_{safe} hosts use the IPv4 address space 10.0.1.0/24, with the internal router using the first address.

Up to 253 additional subnets can be added and up to 30 of these can be tenants. Each of the S normal subnets is denoted as S_i, $(1 \le i \le 253)$ and each of the T tenants is denoted as T_j, $(1 \le j \le 30)$. As noted above, a tenant subnet is one in which there is an additional firewall separating the members of the tenant from the other subnets connected to the internal router. The number of normal subnets S and tenants T can be specified independently. All hosts in subnet and tenant groups are configured alike. The number of hosts, the number of open ports per host, and the number of vulnerabilities per host can be specified separately for tenants and normal subnets. The number of inbound, port-specific "allow" firewall rules can also be specified for each tenant firewall.

The number of hosts, ports-per-host, and vulnerabilities-per-host on both the DMZ and S_0 subnets are independently configurable. For each configured subnet (DMZ, S_0, S_{safe}, S_i, T_j), the user specifies P, the number of ports per host on that subnet. Port numbers are assigned serially from 1 to P as in the flat network.

Four vulnerability types are available: remote-to-admin, remote-to-user, remote-to-other, and local-to-admin. These vulnerabilities are associated with one or more ports on each host, as specified by command-line arguments, and can be set independently for the configured subnets that contain vulnerable hosts (DMZ, S_0, S_i, T_j). For explanatory purposes, NetSPA currently associates vulnerabilities with software versions. For completeness, in the simulation there are exactly four software versions which directly correspond to the four vulnerability types.

Tenant subnets are provided to measure the impact of network partitioning. A tenant contains a dummy subnet representing a point-to-point connection from the internal router to the tenant firewall (external interface), a firewall with exactly two interfaces, and a set of up to 253 hosts. IP addresses in tenant T_j are assigned the addresses 10.0.($S + j$).0/24, where S is the number of normal subnets and j is the index for the tenant subnet. Tenant firewall rules include only user configured, inbound, port-specific allow rules. Each of these rules specifies a single port number as acceptable from the firewall's external zone to the internal zone. Ports are chosen from 1 to P, where P is the number of allow rules specified in the configuration.

Tenant firewalls are simplified and include only port-specific rules while perimeter firewalls include IP-specific firewall rules. This has important implications concerning the extra complexity created by adding rules to the two types of firewalls. The additional computational cost for attack graph

generation when adding a new IP-specific rule is significantly higher than the cost of adding a new port-specific rule. New IP-specific rules in the perimeter firewall add another source address for each virtual attacker host, as described in Section 4.3 and also another firewall rule to process. New port-specific rules in tenant firewalls only add another firewall rule to process and do not add new virtual source addresses.

8.1.3 Enterprise Network

The enterprise network model, shown in Figure 17, is a direct extension of the enclave network model. An enterprise contains E identical enclaves. There can be up to 256 enclaves containing up to 16,582,656 total hosts (256 enclaves * ((255 subnets * 254 hosts) + 6 DMZ hosts)). The total host count shown in figures and tables for enclave and enterprise networks includes hosts that are requested by command-line parameters to populate subnets as well as perimeter firewalls, enclave internal routers, tenant firewalls, and virtual hosts that are automatically added and depend on the network topology. These additional virtual and infrastructure hosts account for the unusual total host numbers that are all slightly larger than expected, given the requested number of hosts.

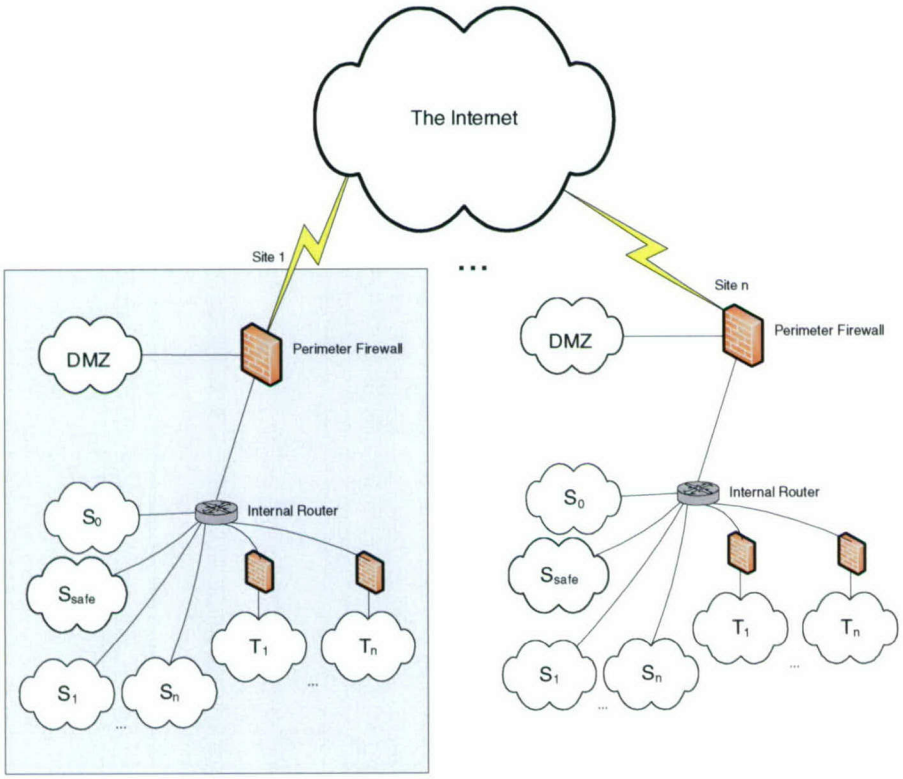


Figure 17. Enterprise network block diagram.

8.1.4 Characteristics of Simulated Networks

Many simulation parameters are varied across experiments and some are set to default values for most studies to reduce the number of simulation runs required. When possible, these default values were selected to be similar to values found in real networks. Table 3 lists important network parameters. It shows whether parameters are fixed at default values or varied across experiments and describes the impact we expect the parameter to have on results.

TABLE 3
Values and the Expected Impact on Results of Important Simulation Parameters

Parameter Name	Explanation	Value Used	Values and Expected Impact on Results
Hosts	Total hosts including infrastructure	285–128,000	This will have a direct and substantial impact on computation time and is, therefore, varied substantially.
Host asset values	The asset value to an attacker obtained by compromising a host	10	The exact value makes no difference and it is only important that all hosts have the same value. This has only a small effect on graph analysis times.
Percentage of hosts with remote-to-admin vulnerabilities	The percentage of hosts that can be compromised at the administrator level by remote attackers	0–100%	This percentage is varied to explore the effect of patching hosts to reduce the number of compromisable hosts. It is 5% for initial flat-network experiments, varies for one flat network experiment, and is 50% for other experiments. A high percentage will increase the size of the attack graph (because more hosts are compromisable), and this will increase computation time.
Firewall filtering and address translation rules	The number of rules used by firewalls in the network	0–2,000	This will have a direct and substantial impact on reachability computation time and is, therefore, varied substantially.
Ports per host	The number of unique TCP and/or UDP ports open on a particular host and accepting connections	10–40	NetSPA's reachability time is strongly affected by this value. Many Windows hosts have roughly 10 ports open per host and this is used as a default value. Some experiments for enterprise networks explore the effect of increasing this value to 40.
Software versions per host	The number of software version instances on a particular host. The versions may or may not be unique	2	NetSPA's graph-generation and analysis algorithms are unaffected. Load times from the database will be artificially shortened. The analysis file may be written more quickly.
Subnets	Number of internal subnets including tenants	1–160	Adding subnets increases the number of source IP addresses in source and destination virtual hosts, increasing reachability computation time. It also may increase the complexity of reachability computations. The number of subnets is varied across experiments.
Unique ports	The number of unique TCP and/or UDP port numbers in use on the network	10	NetSPA's core algorithms are unaffected by this number except where firewall rules specifically refer to port numbers, and in that case the impact on performance is very minor (influencing only how often the rule engine hits an ALLOW or DENY condition on the port-specifying rule).
Unique software versions	The number of unique software versions present in the network	5	NetSPA's graph generation and analysis algorithms are unaffected. Load times from the database will be artificially shortened. The analysis file may be written more quickly.
Unique vulnerabilities	The number of unique vulnerabilities found in the network. Each vulnerability may occur on multiple vulnerable hosts within the same network	5	NetSPA's graph generation and analysis algorithms are unaffected. Load times from the database will be artificially shortened.
Vulnerabilities per host	The number of vulnerability instances on a particular host. The vulnerabilities may or may not be unique	~10	Underestimating this value will reduce the number of edges generated in the graph, lowering attack graph generation and analysis times.

As can be seen from this table, the parameters that are set to one default value are not expected to have a major effect on computation time. The most important parameters that do have a major effect (total hosts, percentage of hosts with remote-to-admin vulnerabilities, number of firewall rules, ports per host, and the number of subnets) are varied across experiments.

8.2 EXPERIMENTS

Four sets of experiments were performed to illustrate NetSPA's capabilities, ranging from a simple demonstration of scaling to discovery of complex interrelations on enterprise networks. The first experiment, "Flat Network Scaling," shows that computation for the core attack graph generation and analysis algorithms grows as the square of the number of hosts present. The second, "Enclave Network," demonstrates a baseline capability to analyze hosts at a single enclave with subnets and a perimeter firewall. It shows that the attack graph accurately corresponds to the expected attack vectors and the automated analysis correctly recommends patching hosts that protect the greatest number of other vulnerable hosts. It elaborates on the firewall explosion in the analysis and shows the effectiveness of our mitigation technique. We build off the understanding created by the firewall explosion results to show how the system is affected by the number of firewall rules present. The third set of experiments, "Enterprise Network Scaling," demonstrates the ability to handle large, highly structured networks. The final experiment, "Customized Enterprise Network," is a hand-modified enterprise network model experiment which demonstrates that NetSPA can discover complex interrelations.

All of the experiments were run on a single processor Pentium 4 1.80-GHz machine with 1024 MB PC133 SDRAM, running Microsoft XP. The following software was installed and running at the time of experimentation: ActiveState Perl 5.6.1-638; MySQL Ver 12.22 Distrib 4.0.20a; Cygwin (Setup version) 2.427, MS Windows taskmanager.

All times reported in the following results are taken directly from instrumentation built into NetSPA. The instrumentation is based on calls to GetProcessTimes, which reports times in 100-nanosecond units. An initial call is made, and at each subsequent call a difference is calculated and the elapsed time is attributed to the current subprocess. We use GetProcessTimes to record the user and kernel times for each NetSPA subprocess. All times reported here are the total of those for each reported timing statistic. Total memory usage required by NetSPA is not reported individually for the following experiments because memory usage never exceeded 250 MB. This is a relatively small requirement and is equivalent to running a few large applications on a Windows workstation.

8.3 EXPERIMENT 1: FLAT NETWORK SCALING

In section 5.1.3 of this report, analysis showed that computation time for the attack graph generation algorithm grows as $O(H^2 \log H)$ for a flat network with H hosts. Experiment 1a substantiates this prediction, and Experiment 1b shows that the attack graph generation time grows linearly as a function of the percentage of compromised hosts.

8.3.1 Experiment 1a.

This experiment uses the flat network model with between 500 and 128,000 hosts. In all nine runs, there are 10 ports per host and one vulnerability per port. All vulnerabilities are remote-to-other, except in the first 5% of hosts, where the first remote-to-other vulnerability has been replaced with a remote-to-admin vulnerability. These networks have no filtering rules and are similar to fairly secure local area networks in which only five percent of all hosts can be compromised.

Table 4 shows the experimental parameters and the data gathered from the instrumentation in NetSPA. Each row in the table represents the recorded data from one run and contains run-time configuration information, as well as data recorded by NetSPA instrumentation. The first column, "% Rootable," shows the percentage of the hosts with remote-to-admin vulnerabilities. The second column, "Total Hosts," shows the number of hosts in the network ignoring the single virtual attacker host. The third, fourth, and fifth columns show times taken from the instrumentation in NetSPA, listed in seconds. The "Overall Time" is an aggregation of the times reported by NetSPA for database interaction (load), attacker address selection, reachability computation, attack graph generation, attack graph analysis, attack graph file write, and full analysis report file write. Load, address selection, and file write times are not included in the table to focus on attack graph generation and analysis times. The column labeled "Reachability and AG Generation" shows the time for both attack graph generation and reachability computation. The column labeled "AG Analysis" shows the time to produce human-readable recommendations from the graph.

TABLE 4
Simulation Times for a Flat Network as the Number of Hosts Varies

% Rootable	Total Hosts	Overall Time	Reachability and AG Generation	AG Analysis
5.00	500	3.41	0.02	0
5.00	1,000	6.58	0.09	0.05
5.00	2,000	13.25	0.3	0.16
5.00	4,000	28.28	1.16	0.63
5.00	8,000	60.05	4.55	2.47
5.00	16,000	145.34	18.25	9.83
5.00	32,000	383.8	73.06	44.67
5.00	64,000	1163.45	292	216.17
5.00	128,000	3877.16	1165.77	834.28

Figure 18 displays the computation time for attack graph generation and analysis versus the number of hosts on a log-log plot. This plot shows that, on a flat network, the computation for both tasks grows as roughly the square of the number of hosts. Recall our predicted runtime in this scenario is $O(H^2 \log H)$. The observed slope of the plot is roughly two, corresponding to $O(H^2)$. The factor of log H is not visible

on the graph. The attack graph generation algorithm grows as the square of the number of hosts. On the first pass of the predictive graph algorithm, the attacker's starting location attempts to compromise all H hosts in the network. On the second pass, each of those compromised hosts (H_c) needs to attempt to compromise the (H – 1) other hosts resulting in $H + H_c(H - 1)$ operations, which is $O(H^2)$ when H_c is proportional to H as it is in this experiment.

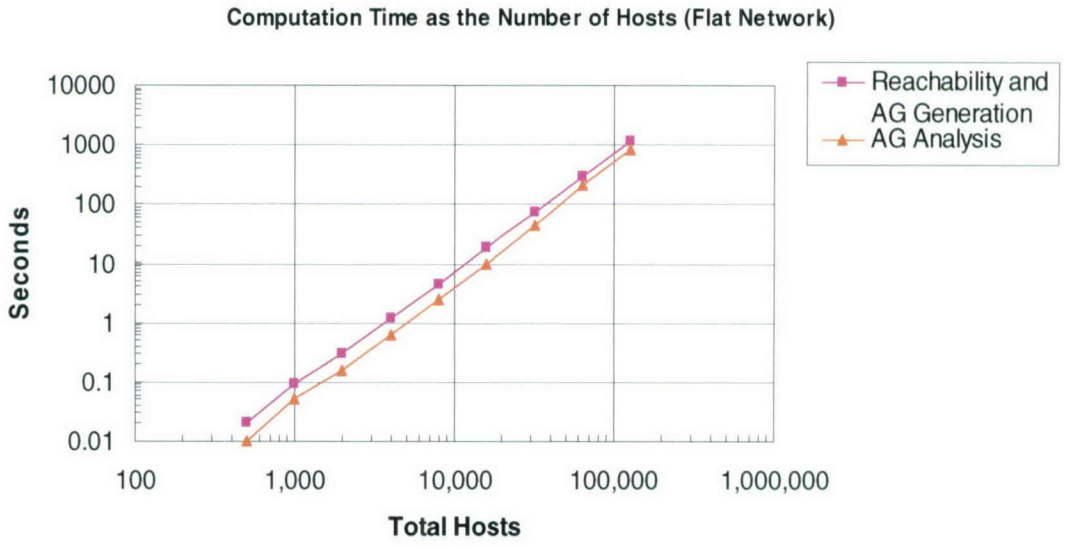


Figure 18. Attack graph generation and analysis times for a flat network as the number of hosts varies.

The attack graph analysis algorithm must consider the removal of each compromised host in turn (H_c), and then evaluate the impact of that removal on the remainder (H_c – 1) resulting in $H_c(H_c - 1)$ operations, again $O(H^2)$when H_c is proportional to H as it is in this experiment.

8.3.2 Experiment 1b.

This experiment uses flat networks with 10,000 hosts and varies the percentage of compromisable hosts from 1% to 100%. Table 5 contains the data for this experiment in the same format as previously described and Figure 19 shows the time required for attack graph generation and analysis. As can be seen, attack graph generation times grow linearly as the fraction of compromisable hosts, P, increases as predicted in Section 5.1.3. From above, the predictive graph computation requires on the order of $H + H_c(H - 1)$ operations. Substituting $H_c = PH$, yields $H + PH(H - 1)$ operations which are linear in P as found when the number of hosts is constant as in these experiments.

TABLE 5
Simulation Times for a Flat Network as the Percentage of Compromisable Hosts Varies

% Rootable	Total Hosts	Overall Time	Reachability and AG Generation	AG Analysis
1.00	10,000	68.61	1.45	0.53
2.50	10,000	72.14	3.53	1.56
5.00	10,000	77.48	7.14	3.80
10.00	10,000	92.70	14.64	10.59
25.00	10,000	175.14	39.81	70.45
50.00	10,000	374.84	90.30	216.42
100.00	10,000	1360.77	224.86	1068.70

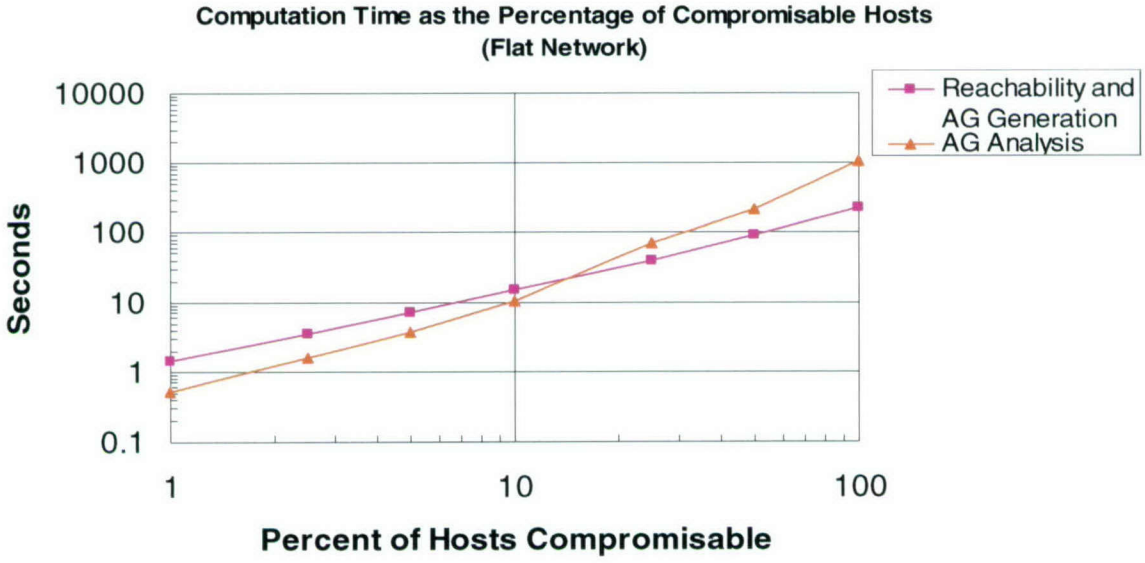


Figure 19. Attack graph generation and analysis times for a flat network as the percentage of compromisable hosts varies.

Attack graph analysis growth is not linear. Figure 19 shows that the analysis time is less than quadratic when fewer than 10% of the hosts are compromised and becomes quadratic when 10% to 100% of the hosts are compromised. From above, the number of operations required for attack graph analysis of flat networks grows as $H_c(H_c-1)$. Substituting $H_c = PH$ and simplifying yields $P^2H^2 - PH$ operations which grow quadratically in P. The apparent change in growth is probably due to fixed computation

requirements that do not depend on P. When P is small, and the run time is less than a few seconds, these additional computations flatten the lower end of the curve and make the growth appear more linear. When P is large, the effect of these constants is negligible and attack graph generation times grow quadratically with P.

8.4 EXPERIMENT 2: ENCLAVE NETWORK

A second set of experiments demonstrates that NetSPA can correctly analyze enclave networks with perimeter firewalls, multiple subnets, and tenant subnets separated from other subnets by firewalls. Experiment 2a shows that NetSPA can process an enclave network and correctly predict network ingress routes based on vulnerability information and firewall rules. Experiment 2b presents a detailed description of a "firewall explosion" and demonstrates our technique for limiting the computational complexity it introduces. Experiment 2c shows that reachability computation time grows as the square of the number of IP-specific firewall rules while attack graph generation and analysis times remain constant.

Figure 16 shows a model enclave network used in Experiments 2a, 2b, and 2c. The single-attacker virtual host starting location is outside the security perimeter, simulating an attacker originating from the Internet. In all three experiments, 50% of the hosts in subnets one through five are compromisable via a remote-to-admin vulnerability on port 1. All hosts on the DMZ, Tenant 1, and in S_0 are compromisable via a remote-to-root vulnerability on port 1. There are five fully patched hosts on S_{safe}. The perimeter firewall has one rule allowing traffic from the external zone to one host in the DMZ, one rule allowing traffic from the DMZ to one host in S_0, and ten inbound, IP-specific block rules. This represents an enclave that is poorly protected and open to attack. Half of the hosts on the interior subnets can be compromised, and although the firewall has only one inbound rule, it allows an outside attacker to compromise a stepping-stone host that provides access to interior subnets.

8.4.1 Experiment 2a: Enclave Network, Host Growth

The attacker's sole path into the protected network is via a single remote-to-admin vulnerability on one machine in the DMZ. From there, the attacker can exploit a single remote-to-admin vulnerability on one machine in S_0. Once access to a machine in S_0 is established, the attacker can directly compromise the remaining vulnerable machines in the network. Table 6 illustrates the impact of varying the number of hosts on the inside of the network. There are two new columns which were not present in Tables 4 and 5. The new columns, "Reachability" and "AG Generation," indicate the computation time required to compute the reachability information needed for attack graph generation and the computation time required to compute the attack graph itself, respectively. Figure 20 shows run times for computing reachability and for attack graph generation and analysis as the number of hosts increases. The number of subnets in these experiments is 80, except for the two highest host values (30,017 and 40,017) for which the number of subnets increases to 120 and 160 to accommodate the large number of hosts.

TABLE 6
Simulation Times for an Enclave Network as the Number of Hosts Varies

Total Hosts	Overall Time	Reachability	AG Generation	AG Analysis
417	5.67	2.92	0.13	0.17
4017	105.84	33.83	16.31	32.45
8017	329.91	66.89	64.92	147.92
12017	765.30	101.41	146.92	437.66
16017	1077.89	133.64	261.22	569.64
20017	1763.53	170.69	409.31	1038.67
30017	3684.81	354.13	920.31	2153.58
40017	8165.11	1256.91	1642.20	4885.73

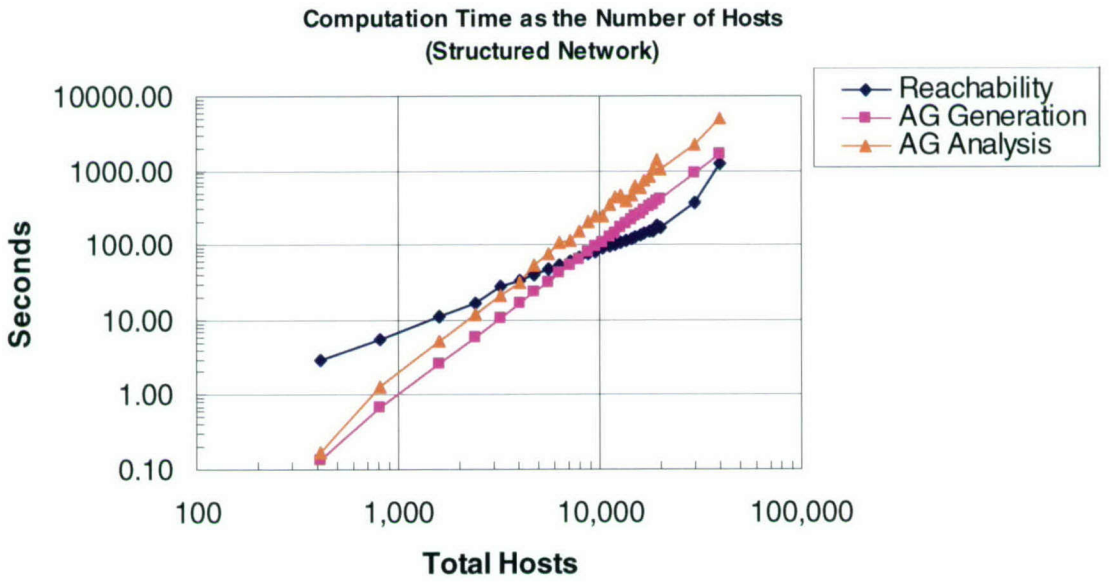


Figure 20. Simulation times for an enclave network as the number of hosts varies.

The observed slope of attack graph generation is roughly two, corresponding to quadratic growth. Attack graph analysis grows similarly. Reachability analysis, on the other hand, exhibits a slope of nearly one because the number of subnets (and thus the number of reachability domains) remains constant as the number of hosts varies. Once reachability domains are formed, reachability computation is linear in the number of hosts. The exception is the highest two values, for which the number of subnets increased, causing a "bump" in the reachability computation required. Adding subnets increases the computation because each new subnet adds another virtual host to the network and also adds a new address to the address-selection algorithm. The attack graph generation and analysis time curves have slopes that are similar to those observed in Experiment 1. The attack graph itself has two stepping-stones leading to the internal network, at which point the attack behaves as a flat-network attack. The overall run times for the enclave network are much greater than for the flat network. For example, overall run times with roughly 16,000 hosts are 145 seconds for the flat network and 1,078 seconds for the enclave network. This increase is primarily caused by the increase in hosts with remote-to-admin vulnerabilities and the corresponding increase in the number of hosts compromised and size of the attack graph. The percentage of hosts with remote-to-admin vulnerabilities increased from 5% for the flat network to 50% for the enclave network and run times increased roughly by the same order of magnitude.

8.4.2 Experiment 2b: Enclave Network, Firewall Explosion

Attack graphs become more complicated when there is more than one potential path into the network. Experiment 2b introduces additional vulnerable hosts to subnet S_0, but holds all other variables constant. After compromising the one vulnerable host in the DMZ, the attacker can compromise all of the vulnerable hosts in S_0. The attacker then proceeds to compromise the rest of the internal network from all of the compromised hosts in S_0. Each additional vulnerable host in S_0 creates duplicate child subgraphs. The overall graph size and attack graph computation is roughly multiplied by the number of vulnerable hosts when compared to the baseline Experiment 2a with only one vulnerable host.

TABLE 7
Simulation Times for an Enclave Network as the Degree of a Single-Level Firewall Explosion Varies

Degree of Explosion	Total Hosts	Tree Nodes	Overall Time	Reachability	AG Generation	AG Analysis
1	1017	513	11.03	2.38	1.05	1.73
4	1020	2046	18.78	2.41	4.08	6.77
9	1025	4601	31.16	2.39	9.20	13.39
16	1032	8178	48.44	2.39	16.52	23.25
25	1041	12777	71.61	2.41	26.03	36.72
36	1052	18398	107.97	2.50	37.89	60.92

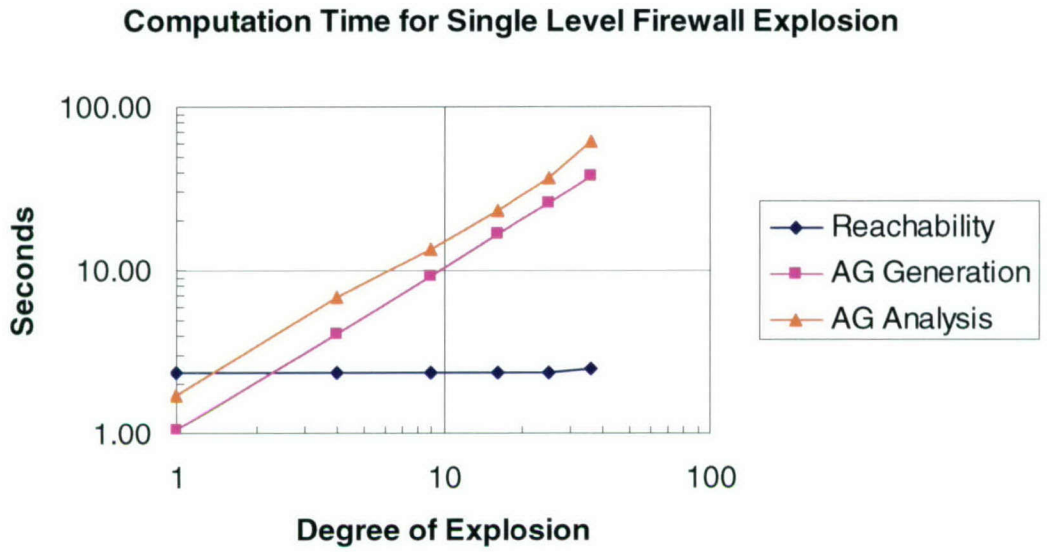


Figure 21. Simulation times for an enclave network as the degree of a single-level firewall explosion varies.

Experiment 2b illustrates this phenomenon. An enclave network was used with 10 subnets and roughly 1,000 hosts, and then hosts with remote-to-admin vulnerabilities were added to S_0. Figure 21 illustrates how computation time grows roughly linearly as vulnerable hosts are added to S_0. This is a *firewall explosion*, as defined in Section 5.1.4. Table 7 shows how run times and the attack graph size vary as the "Degree of Explosion," equal to the number of vulnerable hosts in S_0, varies.

Section 5.1.4 predicts that firewall explosions in series are multiplicative. This implies that a degree C explosion followed by a degree D explosion has an impact equivalent to a single degree CD explosion. We verify this relationship with the following two-stage firewall explosion. First, C vulnerable hosts are placed in the DMZ. Then D vulnerable hosts are placed in S_0. This creates a two-stage explosion with a predicted degree of CD. For illustrative purposes, we set C = D. The results in Figure 22 verify the prediction; the performance of the algorithm is nearly identical to that in Figure 21. This experiment utilizes two firewall explosions in series, but is otherwise identical to the former experiment. In the two-level case, we use an explosion of degree C followed by another of degree C, and in the single-level case it is an explosion of degree C^2. Table 8 contains the full experimental results for the two-stage case and indicates the two-level makeup of the network. The first column, "Degree of First Explosion," indicates the number of hosts placed in the DMZ. The second, "Degree of Second Explosion," indicates the number of hosts placed in S_0. The third column, "Product of Degrees," shows the overall multiplicative degree of the explosion, allowing easy comparison to corresponding rows from Table 7. The result demonstrates that firewall explosions are multiplicative as predicted.

TABLE 8
Simulation Times for an Enclave Network as the Product of the Degrees of Two Firewall Explosions Varies

Degree of First Explosion	Degree of Second Explosion	Product of Degrees	Total Hosts	Tree Nodes	Overall Time	Reachability	AG Generation	AG Analysis
1	1	1	1017	513	10.77	2.42	1.03	1.73
2	2	4	1019	2047	18.83	2.41	4.09	6.61
3	3	9	1021	4603	30.53	2.39	9.19	13.00
4	4	16	1023	8181	48.06	2.41	16.41	23.16
5	5	25	1025	12781	73.89	2.38	25.73	39.28
6	6	36	1027	18403	96.48	2.38	36.75	50.84

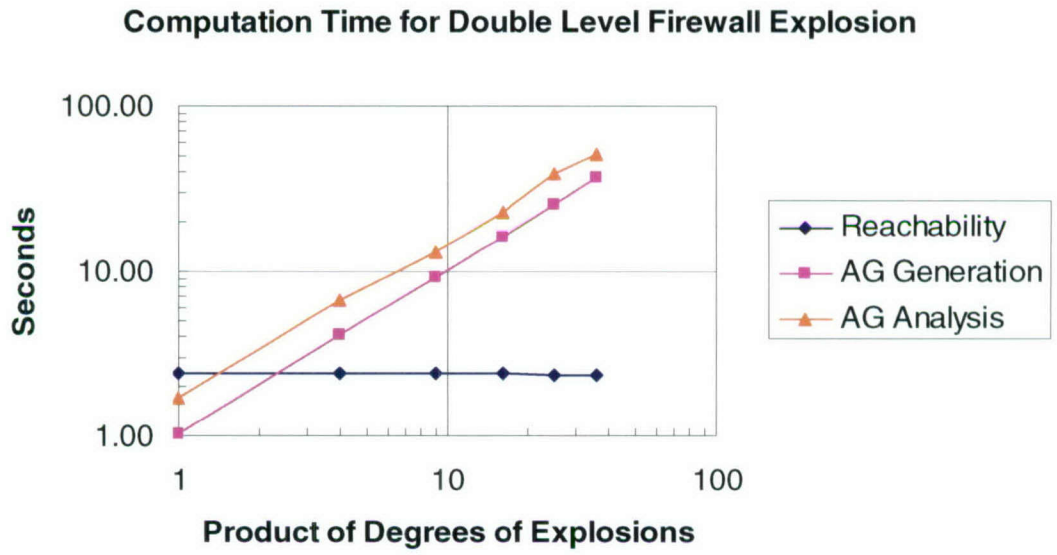


Figure 22. Simulation times for an enclave network as the product of the degrees of two firewall explosions varies.

Real networks may create this phenomenon. Each pass through a filtering device can potentially introduce an explosion, and a network with many separated segments could introduce several explosions. Even a handful of low-degree explosions, when combined, can cause a very large overall degree and substantially degrade performance or render a modestly sized network's attack graph intractably large.

We developed the dynamic host collapse (DHC) graph to mitigate the impact of firewall explosions. DHC is discussed in Section 5.1.4. When the two-level firewall explosion experiment is re-run using DHC, the runtimes drop dramatically, as seen in Table 9 and Figure 23. Note that the scales are different in Figures 22 and 23, and the times for attack graph generation and analysis with DHC are so small that they wouldn't even show up on Figure 22. DHC has reduced the size of the graph from more than 18,000 nodes to only 5 nodes and made attack graph generation and analysis times insignificant. DHC can also reduce run times substantially even in networks without a firewall explosion because it reduces the number of graph nodes and, therefore, reduces the overall number of outbound edges the graph algorithm must consider and discard. A DHC graph presents the administrator with less information than a normal predictive graph, but in some networks with a high-degree firewall explosion, it may be computationally infeasible to build a normal predictive graph instead.

TABLE 9
Simulation Times for an Enclave Network as the Product of the Degrees of Two Firewall Explosions Varies, Using Dynamic Host Collapse

Degree of First Explosion	Degree of Second Explosion	Product of Degrees	Total Hosts	Tree Nodes	Overall Time	Reachability	AG Generation	AG Analysis
1	1	1	1017	5	7.91	2.36	0.03	0.02
2	2	4	1019	5	8.11	2.39	0.05	0.01
3	3	9	1021	5	8.11	2.39	0.06	0.02
4	4	16	1023	5	8.08	2.38	0.06	0.02
5	5	25	1025	5	8.00	2.38	0.05	0.02
6	6	36	1027	5	8.02	2.36	0.09	0.02

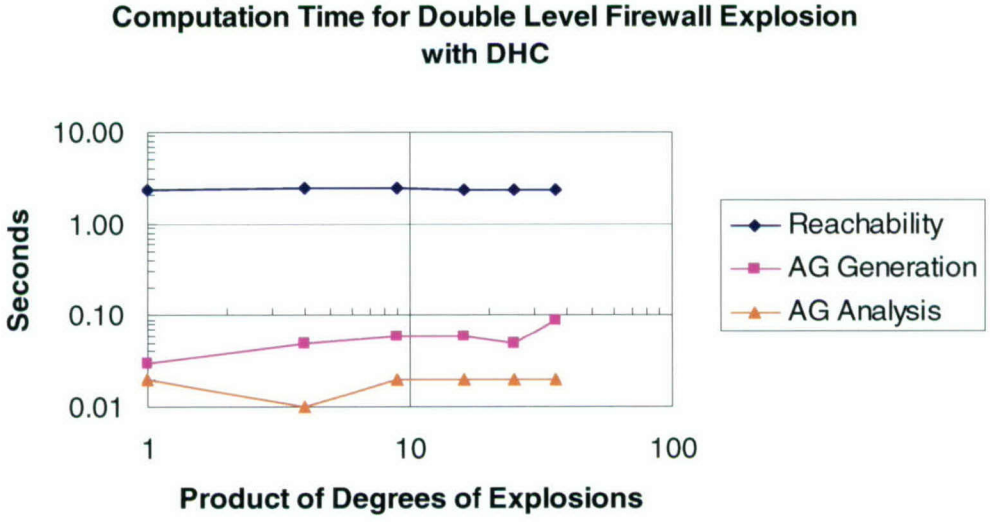


Figure 23. Simulation times for an enclave network as the product of the degrees of two firewall explosions varies, using dynamic host collapse.

8.4.3 Experiment 2c: Firewall Rules

Experiment 2c explores the effect of adding IP-specific rules to the perimeter firewall in an enclave. In this experiment, the network structure (517 hosts, 10 subnets) is constant and similar to that used in Experiment 2a. Only the number of IP-specific blocking firewall rules on the perimeter firewall is varied. Each blocking rule blocks one unique source IP address. This has no impact on the actual graph because the attacker will simply choose the original unblocked source IP address, but it does illustrate the expected performance impact. Similar results would be obtained if accept or pass rules were added using IP addresses of patched or nonexistent hosts; in both cases, only one host can be reached through the firewall and serve as a stepping-stone.

Table 10 and Figure 24 illustrate the impact of firewall rule set size. Figure 24 shows that attack graph generation and analysis times remain low and unchanged as the number of firewall rules change. This was expected because blocking rules do not change the attack graph. Only the time required for reachability computations increases as firewall rules are added. Initially this increase grows slowly with fewer than 100 firewall rules, and it then grows quadratically with more than 100 firewall rules. Quadratic growth occurs because, as noted above, each new rule adds a new starting IP address to the attacker virtual host and each new rule increases the complexity of processing required every time the firewall is traversed to compute reachability from the new virtual IP address. We are pursuing alternative approaches to reachability computation with complexity that may grow less than quadratically.

TABLE 10
Simulation Times for an Enclave Network as the Number of IP-Specific Firewall Filtering Rules Varies

Total Hosts	Firewall Rules	Overall Time	Load	Reachability	AG Generation	AG Analysis
517	1	4.39	3.00	0.69	0.25	0.42
517	10	4.61	2.92	1.02	0.27	0.38
517	100	9.77	2.94	6.14	0.25	0.41
517	1000	1164.22	4.81	1158.59	0.28	0.45
517	2000	5388.05	5.77	5381.41	0.25	0.47

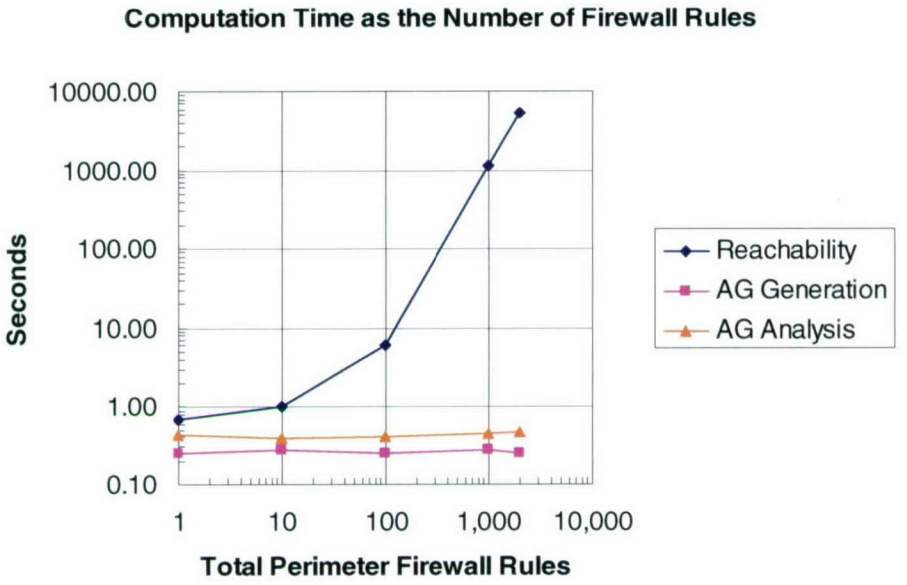


Figure 24. Simulation times for an enclave network as the number of IP-specific firewall filtering rules varies.

8.5 EXPERIMENT 3: ENTERPRISE NETWORK SCALING

A model network that emulates a multi-enclave enterprise is shown in Figure 17. Experiments were performed using an enterprise model with five enclaves to create a high level of topological complexity. Experimental parameters were adjusted to represent a simple and a complex network. The simple network has 10 open ports per host and 10 IP-specific filtering rules per perimeter firewall for a total of 50 rules in all firewalls. The complex network has 30 open ports per host and 400 IP-specific rules per perimeter firewall for a total of 2,000 rules in all firewalls. Other parameters are as in Experiment 2a.

Experiments for the simple enterprise network were performed with 2,000 to 40,000 hosts. Results in Figure 25 and Table 11 demonstrate that attack graphs can be generated for large enterprise networks in short, practical computation times. Total computation times are roughly one minute for 2,000 hosts, 10 minutes for 10,000 hosts, and a half hour for 20,000 hosts. Reachability and attack graph analysis times dominate in these experiments, and attack graph generation is relatively fast. These times would be less if the attack graph only considered an attacker starting from one fixed IP address instead of trying all IP addresses used in firewall rules and included in the virtual attacker host. Alternatively, run times would increase if there were more firewall rules and thus more addresses for the virtual attacker host to use.

TABLE 11
Simulation Times for a Simple Enterprise Network as the Number of Hosts Varies

Total Hosts	Overall Time	Reachability	AG Generation	AG Analysis
2085	70.92	48.94	0.86	8.64
8085	407.31	215.58	13.16	127.78
14085	888.14	357.83	39.52	393.23
20085	1758.77	684.09	80.67	844.95
26085	2713.92	911.41	136.36	1452.55
32085	3539.00	986.59	207.27	2070.61
40085	5785.70	1642.80	324.78	3428.44

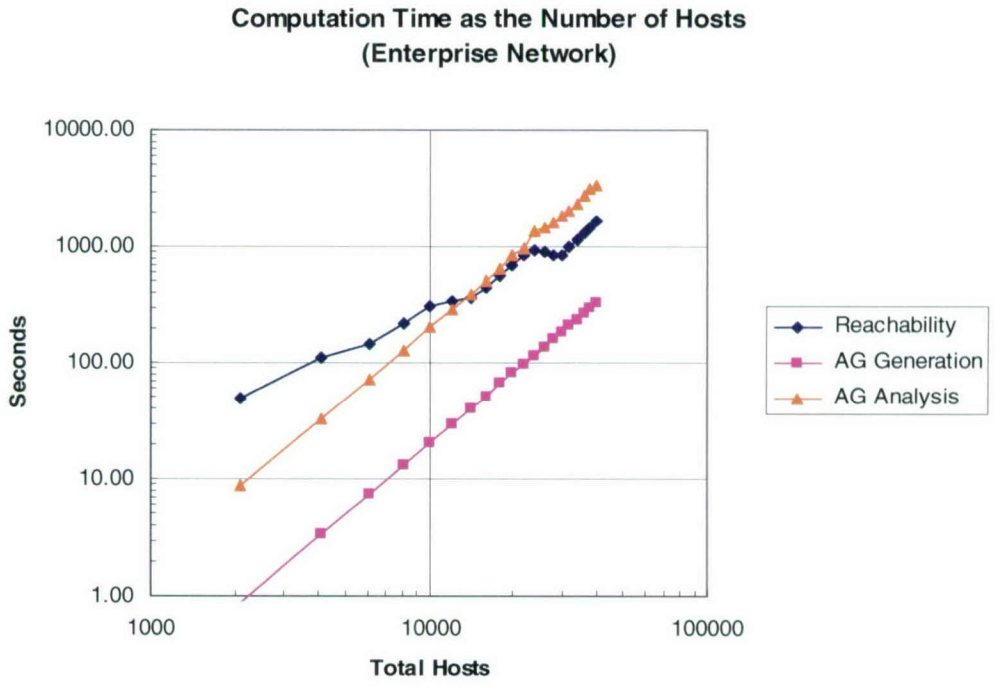


Figure 25. Simulation times for a simple enterprise network as the number of hosts varies.

The effects of these two changes are shown in Figure 26. The lowest curve in Figure 26 shows the total computation time for an attacker starting from one specific external IP address for the simple enterprise network. This can be compared to the next higher curve in this figure, which is the total computation time for the simple enterprise network using the default external virtual attacker host that includes all IP addresses that occur in any firewall rule. The total computational time is smaller with only a single starting IP address because reachability from the attacker's starting location has to be computed only from one IP address instead of from many IP addresses used in firewall rules. Table 12 shows the run times for the simple enterprise network using only a single-attacker IP address. Comparing Tables 11 and 12 indicates that using a single IP address dramatically reduces reachability computations. In this case, this reduces the total run time only slightly because the attack graph analysis time is almost unchanged and dominates the total run time. An attack graph that includes only one IP starting address might be of interest if attackers could not assume or "spoof" arbitrary external IP addresses but were forced to use one or more fixed starting addresses.

TABLE 12
Simulation Times for a Simple, Single-Source IP Enterprise Network as the Number of Hosts Varies

Total Hosts	Overall Time	Reachability	AG Generation	AG Analysis
2085	29.86	7.30	0.89	8.77
8085	226.47	29.70	13.27	130.70
14085	590.22	50.98	40.09	402.80
20085	1127.08	74.03	82.17	822.92
26085	1954.05	94.61	138.69	1507.39
32085	2948.66	121.89	210.31	2338.34
40085	4781.56	156.86	329.89	3929.75

TABLE 13
Simulation Times for a Complex Enterprise Network as the Number of Hosts Varies

Total Hosts	Overall Time	Reachability	AG Generation	AG Analysis
285	277.08	270.34	0.02	0.02
885	824.45	809.00	0.34	1.70
1485	1396.63	1370.55	1.00	3.59
2085	2083.14	2043.02	2.28	8.97
5085	5534.66	5407.50	12.11	49.30
8085	8849.95	8569.53	32.67	135.30
10085	10443.22	10043.61	50.95	210.64

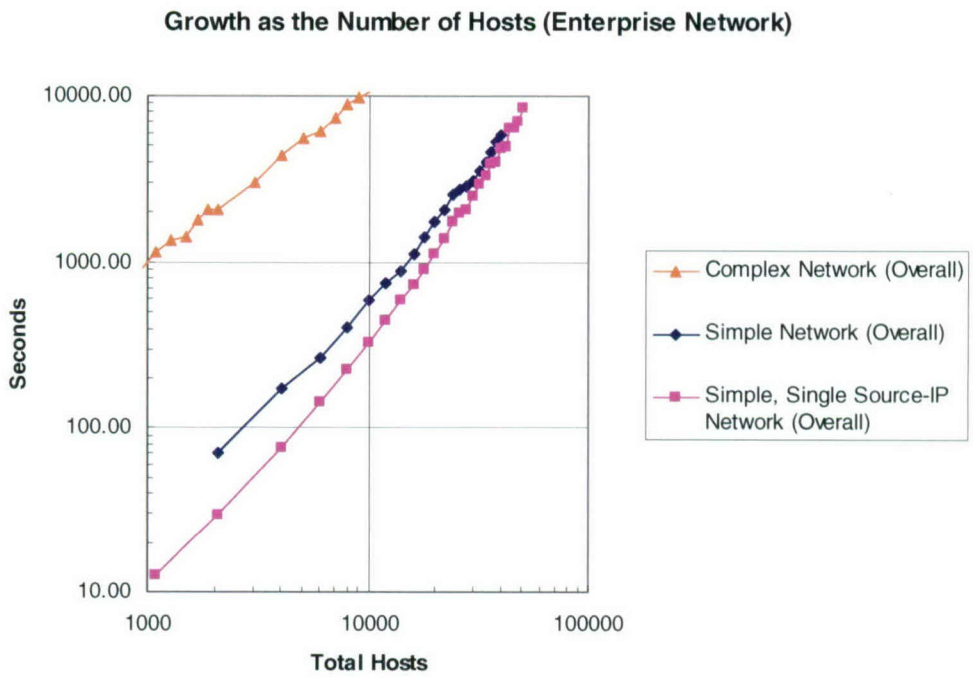


Figure 26. Simulation times for simple, complex, and simple single-source IP enterprise networks as the number of hosts varies.

The upper curve in Figure 26 and Table 13 shows the overall run time for the complex enterprise network with 30 open ports and 400 IP-specific rules in each perimeter firewall. This complex enterprise network has a total of 2,000 IP-specific firewall rules instead of the total of 50 IP-specific rules in the simple enterprise network. The overall run time with 10,000 hosts increases to roughly three hours for the complex network, instead of 10 minutes with the simple network. With 1,000 hosts, the overall run time is roughly 15 minutes. As noted above, the increase in run times is caused by the additional starting addresses for the virtual attacker host, the additional firewall rules that need to be analyzed, and the additional target ports that must be considered when computing reachability. We are exploring approaches that should reduce computation time substantially for situations in which there are many firewall rules, as with this complex enterprise network.

8.6 EXPERIMENT 4: CUSTOMIZED ENTERPRISE NETWORK

A final experiment demonstrates that NetSPA can find long multi-hop paths through a vulnerable network. This experiment uses an enterprise model with three enclaves connected via the Internet, representing a large corporate network with three business locations. The network is similar in structure to the network used in Experiment 3, but host parameters are selected at random. Each of the 3,000 hosts has a varying number of open ports and vulnerabilities. The average number of open ports is 13, and the

average number of vulnerabilities per host is four. Additionally, the firewall rules are altered to allow only the following connections:

1. The attacker can only directly compromise two vulnerable machines on the first enclave's DMZ
2. A Virtual Private Network (VPN) tunnel exists from the first enclave's internal network to the second enclave's internal network
3. A second VPN tunnel exists from the second enclave's internal network to the third enclave's internal network

The shortest path from an attacker on the Internet to the third enclave's DMZ has many stepping-stone hosts. The attacker must compromise the following hosts, in order:

1. One of two vulnerable hosts on the first enclave's DMZ
2. Any vulnerable host within the first enclave's internal subnet
3. The host in the first enclave with VPN privileges to a host on the second enclave
4. The host in the second enclave on the opposite end of the VPN connection
5. The host in the second enclave with VPN privileges to a host on the third enclave
6. The host in the third enclave on the opposite end of the VPN connection
7. A host in the third enclave's DMZ

The NetSPA system successfully produced an attack graph for the above scenario in roughly an hour, using an attacker starting location with a single fixed IP address. The resulting graph is far too large to view. We have instead presented a visually simplified portion of the graph in Figure 27. In the original graph, an edge is labeled with the single vulnerability it represents, and a node is labeled with the host and access level it represents. In the simplified graph, multiple edges between two nodes are collapsed together and, therefore, may represent multiple vulnerabilities, and leaf nodes are collapsed to a single node and are labeled with the number of hosts they represent. Additionally, two-stage attacks against the same host (remote-to-user followed by local-to-admin) are collapsed into a single node.

Each node corresponding to one of the seven stepping-stones outlined above is indicated by number. The attacker starting location is not shown, but an edge from this starting location to one of the two vulnerable hosts in the first enclave's DMZ, labeled as step 1, is shown. Note that the graph also captures the attacker's ability to compromise other hosts at each level. For example, consider the node labeled as step 2, representing one of the vulnerable hosts in the first enclave's internal subnet. In addition to compromising the host representing step 3, this host is also able to compromise the remainder of the vulnerable machines in the first enclave. These additional compromises are represented by the three additional nodes—one for hosts compromised directly at administrator level, one for direct compromise at user level, and one for direct compromise at user level followed by a local-to-admin attack. A similar structure appears at step 4, where the rest of the second enclave can be compromised, and at step 6, where the rest of the third enclave can be compromised.

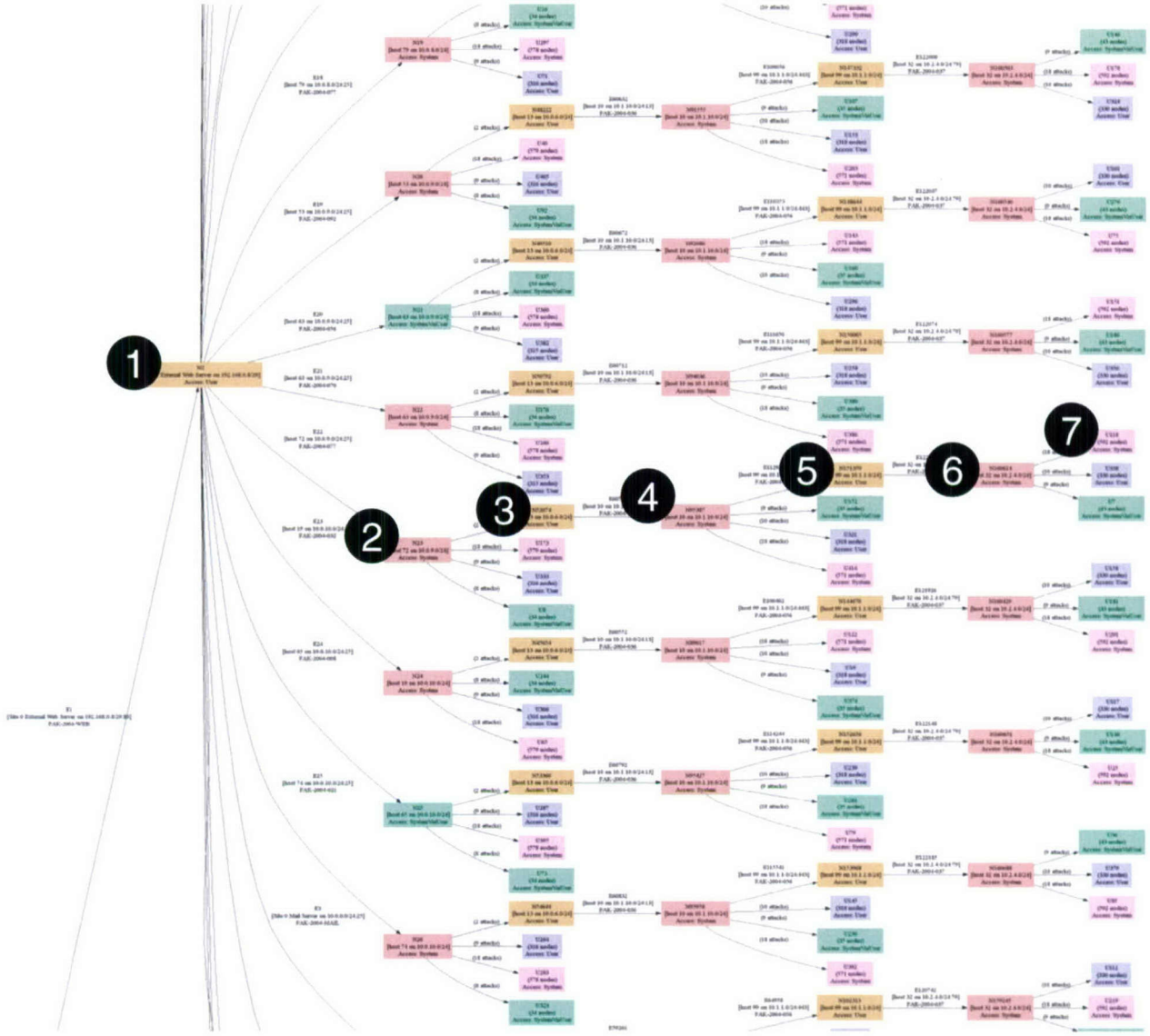


Figure 27. Simplified segment of an attack graph generated for the customized enterprise network.

Because attack graphs are large and difficult to interpret by hand, recommendations are automatically generated from the graph as described in Section 6. Figure 28 shows the first three recommendations generated for the customized enterprise network scenario. These recommendations are those that produce the largest reduction in NCP. The first line of a recommendation indicates the suggested action, the second line shows the reduction in the NCP score if that single action is followed, and the third line shows the number of hosts in a host group if the recommendation applies to a group. The following lines identify the hosts, the ports used to access the vulnerability, and the vulnerabilities

that must be patched. The vulnerabilities listed are only those critical to an attacker's progress and do not necessarily include every known vulnerability on a host. The first recommendation in Figure 28 corresponds to the attacker's first action in Figure 27 and recommends patching the two vulnerable hosts in the DMZ of the first enclave. The second recommendation corresponds to the third action in Figure 27 and suggests patching the host at the end of the VPN in the first enclave. The third recommendation corresponds to the fourth action in Figure 27 and suggests patching the host at the end of the same VPN in the second enclave. Note that the reduction in NCP is greatest for the first recommendation, and then the reduction becomes less for the other recommendations. For this attack path, it would be desirable that the second recommendation correspond to step 2 in the attack graph of Figure 27 because this reduces the NCP more than the second recommendation shown in Figure 28. This doesn't occur with our current recommendation algorithms, and we have designed a new algorithm for a future version of NetSPA designed to correct this limitation. The current algorithm does find the single best recommendation that provides the greatest reduction in NCP, but the second and following recommendations may not be those that would be found by following the first recommendation, recomputing the attack graph, and generating the first recommendation again for the revised attack graph.

```
1  Recommendation: Patch software on group of hosts or block ports at gateway(s)
   Vulnerability score change: -19.1
   Hosts in group (2 total)
   Host 1 of 2: Site 0 External Web Server on 192.168.0.8/29
     Port: 80 a software version 0.0
       FAK-2004-WEB       (MT2) r2u port 80 vuln
   Host 2 of 2: Site 0 External Mail Relay on 192.168.0.8/29
     Port: 25 a software version 0.00
       FAK-2004-MAIL      (MT2) r2u port 25 vuln

3  Recommendation: Patch software on host or block ports at gateway(s)
   Vulnerability score change: -18.7
   Host: host 49 on 10.2.10.0/24
     Port: 25 a software version 0.2
       FAK-2004-092       (MT3) synthetic r2r vulnerability
     Port: 445 a software version 0.57
           445 a software version 0.10
       FAK-2004-045       (MT3) synthetic r2r vulnerability
       FAK-2004-055       (MT3) synthetic r2r vulnerability
     Port: 515 a software version 0.21
           515 a software version 0.21
       FAK-2004-060       (MT2) synthetic r2u vulnerability
     Port: 1103 a software version 0.33
       FAK-2004-055       (MT3) synthetic r2r vulnerability

4  Recommendation: Patch software on host or block ports at gateway(s)
   Vulnerability score change: -17.8
   Host: host 31 on 10.1.2.0/24
     Port: 540 a software version 0.3
           540 a software version 0.90
       FAK-2004-013       (MT3) synthetic r2r vulnerability
     Port: 8081 a software version 0.5
       FAK-2004-020       (MT3) synthetic r2r vulnerability
```

Figure 28. Excerpt of the recommendations generated by the customized enterprise network.

8.7 SIMULATION EXPERIMENTS SUMMARY

The above experiments confirm NetSPA's ability to handle complex network topologies and firewalls with many filtering rules. They demonstrate that the predictive graph and node-predictive graph techniques outlined in Sections 5.1.3 and 5.1.4 scale well even with large enterprise networks with 10,000 and more hosts. Total times to compute reachability, generate attack graphs, and analyze graphs to produce recommendations depend on network complexity but can be less than 10 minutes even for networks with 10,000 hosts. Run times increase as the number of IP-specific rules in firewalls increases, as the number of open ports per host increases, and as the number of hosts that can be compromised at administration level increases. These simulation studies also demonstrate that the time required to compute attack graphs is often much less than the time required to compute reachability and analyze attack graphs to make recommendations. They suggest that future work should explore approaches to reduce the computation time required for these two tasks.

9. RELATED WORK

A comprehensive annotated review of past research on attack graphs is provided in [24]. This review demonstrates that although research has made significant progress, no past system has analyzed networks with more than 20 hosts, and computation for most approaches scales poorly and would be impractical for networks with more than even a few hundred hosts. It also shows that past approaches are limited because many require extensive and difficult-to-obtain details on attacks, many assume that host-to-host reachability information between all hosts is already available, and many produce an attack graph but do not automatically generate recommendations from that graph.

The most recent version of NetSPA described in this report greatly extends the initial NetSPA system developed by Artz [19] that is also reviewed in [24]. This new version of NetSPA incorporates efficient approaches to computing reachability, generating predictive and node-predictive graphs, and automatically making recommendations. It also includes tools to automatically import firewall and vulnerability information and has been applied to real networks with more than 3,000 hosts and simulated networks with more than 40,000 hosts.

Solutions to problems of scaling, modeling attacks, computing reachability, and formulating recommendations incorporated in the most recent version of the NetSPA system were often motivated by approaches followed by other researchers. The most important problem addressed by our work is that of scaling to large enterprise networks. The most capable past systems were the Topological Vulnerability Analysis (TVA) tool developed at George Mason University [1, 15, 25–27] and the initial NetSPA tool developed at MIT Lincoln Laboratory [19]. Both systems could construct attack graphs for small 17-host networks that were simulations of actual networks, and scaling for both systems was poor. Computation required for the TVA tool described in [25] is bounded by H^6, where H represents the number of hosts in a network. Although this is not combinatorial and is the best upper bound reported to date, as stated in [25], this approach will only scale to networks with at most "tens or hundreds of hosts."

Timing measurements and an algorithmic analysis demonstrate that scaling for the initial NetSPA tool [19] was also poor. This approach is more complete than that described in [25] because instead of assuming that reachability between all hosts is provided, it computes the reachability between all hosts before generating attack graphs. It also simultaneously builds an attack graph that shows how all targets in a network can be compromised by an attacker from a given starting location. A small 17-host network was analyzed in less than 90 seconds (including reachability computations), but scaling to larger networks was poor because this approach constructs a full attack graph whose size grows combinatorically as the number of hosts increases. Attack graph generation in this first version of NetSPA scaled as poorly as other systems that also used full attack graphs [2, 16, 20, 21]. Initial collaborative experiments [18] demonstrated that model checking also exhibits similar poor scaling for large networks.

In practice it is desirable to compute attack graphs for enterprise networks with 10,000 to 100,000 hosts. Past studies suggest a few approaches to improve scaling. One approach followed by [25] is to

produce a restricted graph that is designed solely to answer questions required to analyze network security and that restricts attack types to those that do not disable prerequisites for other future attacks. Our predictive graphs are designed solely to determine hosts that can be compromised and the vulnerabilities required to exploit them. We also require that attacks do not disable prerequisites for other future attacks and model attacks using only information on locality (remote, local) and effect (remote-to-admin, remote-to-user, local-to-admin, DoS, and other). This has been adequate to model most important vulnerabilities that can be detected with a network vulnerability scanner. It also leads to a graph-building algorithm that scales roughly as H^2 on many real and simulated networks.

Another approach suggested to improve scaling is to group or aggregate similar hosts together. This reduces H by replacing many individual hosts by representative hosts. Aggregation was suggested in [2] for this purpose and in [27] to simplify the visual presentation of attack graphs. We perform three important types of aggregation. First, when computing reachability, hosts in one or more fully connected subnets are grouped into unfiltered reachability domains. Second, hosts that have the same reachability to all other hosts external to a subnet are grouped into filtered reachability domains. On many real networks, these two types of grouping reduce the computation required to compute reachability by one or two orders of magnitude. A third type of grouping is used when computing node-predictive graphs. Here, all hosts with equivalent inbound compromisability are formed into host groups. This often prevents the "firewall explosions" described in Section 5.1.4 and can dramatically reduce computation required to compute attack graphs and the graph size. Finally, we have explored approaches to group similar hosts on a subnet that have the same operating systems and are maintained together and thus have the same vulnerabilities. This has proved relatively ineffective in reducing computation because too few hosts in networks we have analyzed are similar enough to permit this type of grouping. In addition, much of the computational savings that would be gained by this type of similar host grouping is already provided by the three other types of grouping described above.

A second major problem addressed by our work concerns obtaining information on prerequisites and post-conditions for exploits. Many past studies required extensive attack details (e.g., [1, 17]) that could only be obtained by hand. To solve this problem, we require only limited information concerning attacks and fill in unknown attack and vulnerability details with reasonable default values as suggested in [2]. We also automatically import vulnerability information from vulnerability scanners and the ICAT database and use pattern classification approaches to automatically determine attack characteristics from multiple sources. In many cases we find that attack graph analysis identifies a small set of critical vulnerabilities on a few stepping-stone hosts that enable an outside attacker to progress to internal targets. This small set of critical vulnerabilities can be verified by hand. This is much more tractable than hand verifying the analysis of all vulnerabilities.

A third major problem addressed by our work is performing reachability computations to determine which other hosts in a network can be reached from a given host by modeling the effect of gateways and firewalls. A weakness of many past approaches is the assumption that reachability information is available prior to computing attack graphs (e.g., [18]) or that it can be obtained using a vulnerability scanner from each subnet in the analysis (e.g., [1]). Determining reachability between all hosts in large networks with many firewalls is a computationally complex task. It is almost impossible to determine

reachability using vulnerability scanners, and this approach can severely underestimate reachability. Firewalls can contain hundreds to thousands of access control rules, network address translation (NAT) rules, and network objects that represent groups of IP addresses (e.g., [28]). A single scan from one IP address will only exercise a few of these rules. Such scans will miss rules that apply to other source IP addresses or to destination IP addresses not included in the scan. It is difficult, in general, to know which source and destination addresses will be treated differently by a firewall due to NAT rules and the complexity of other rules. In addition, a single scan will not discover the effect of time-dependent firewall rules. To accurately determine the reachability between hosts in separate subnets, we download and analyze configuration files for firewalls and the output of network vulnerability scanners. Simulation studies and experiments on actual networks have demonstrated that computing reachability is often more computationally intensive than constructing attack graphs. Much of our effort has focused on reducing reachability computations by grouping hosts into filtered and unfiltered reachability domains, computing reachability on demand, and using efficient data structures and algorithms. Scaling for our current reachability algorithms depends on the complexity of the network topology and the number and complexity of firewall rules. For many simple networks, computation grows only quadratically as hosts and firewall rules are added. Reachability algorithms have been applied successfully to networks with thousands of firewall rules and thousands of hosts and completed in minutes on general-purpose computing hardware.

A final problem addressed by our work is the generation of recommendations from attack graphs. Past studies have demonstrated that attack graphs can become large and complex even for networks with fewer than 20 hosts (e.g., [1, 19]). Such large graphs, once generated, can be difficult to analyze and understand. An alternative approach is to not only generate attack graphs, but to also automatically analyze attack graphs to address security issues and make recommendations to improve security. Recommendations could, for example, suggest changes in the network architecture or patches for installed software that protect important hosts but result in few changes. One past study [26] developed approaches to make such recommendations, but it is not clear that this approach would scale well to large networks. We developed an approach that uses predictive or node-predictive graphs to make recommendations concerning patching a single host or groups of hosts. This algorithm has been applied to large networks with more than 10,000 hosts and typically requires computation that is equivalent to that of our graph-building algorithms.

10. SUMMARY

A new version of a tool named NetSPA (Network Security Planning Architecture) has been developed to assess the security of large enterprise networks. Firewall configuration files, data from network vulnerability scanners, and attack descriptions from vulnerability databases are automatically imported and used to build attack graphs that show how far inside and outside attackers can progress through a network by successively compromising exposed and vulnerable hosts. This analysis can be used for many purposes including comparing the security of alternative network designs, identifying patches or firewall configuration changes that strengthen security, determining the security risk caused by new vulnerabilities or proposed changes in firewall rules, and identifying the most critical hosts to patch first when a new vulnerability is announced.

Extensive simulation experiments and field trials on actual networks demonstrate the practicality and utility of our approach. During field trials, NetSPA successfully imported vulnerability scanner and firewall configuration information and was able to produce attack graphs and make recommendations in only a few minutes for three actual networks with 200 to 3,400 computers. When firewall information was available, recommendations allowed systems administrators to focus on a few critical "stepping-stone" hosts to confirm the presence of vulnerabilities or to focus on a few firewall rules that enabled attackers to penetrate networks from outside a perimeter firewall. After careful review by local analysts of these few hosts, either networks were made secure from outside attacks or it was determined that networks were secure. Simulation studies were used to benchmark the performance of algorithms used to compute reachability, to generate graphs, and to formulate recommendations. These studies led to many algorithmic improvements. The current system has successfully analyzed simulated networks with more than 10,000 hosts that are similar in topology to large enterprise networks. Performance on many simulated networks scales roughly quadratically in both the number of hosts and firewall rules. This is much better than previous systems. Theoretical analyses of two other recent attack graph algorithms suggest scaling on the order of H^6 [1, 25] or H^4 [29]. On networks with thousands of hosts, our approach requires many orders of magnitude less computation than these approaches.

Simulation studies also demonstrate that computing reachability is often more computationally intensive than constructing attack graphs. Although many past studies assume reachability information is available, much of our effort has focused on making reachability computations more efficient by grouping hosts into filtered and unfiltered reachability domains, computing reachability on demand, and using efficient data structures and algorithms. Scaling for our current reachability algorithms depends on the complexity of the network topology and the number and complexity of firewall rules. For many simple networks, reachability computations grow only linearly as hosts are added and quadratically as firewall rules are added. Reachability algorithms have been applied successfully to networks with thousands of firewall rules and thousands of hosts and complete in minutes on general-purpose computing hardware.

Field trials, simulation studies, and an analysis of attack and network design trends suggest new directions for future research to improve the current NetSPA system. A first new direction is to extend

graph generation to model additional types of attacks. These include attacks against network filtering devices, including routers and firewalls that make filtering less restrictive and make new hosts reachable from hosts already on an attack graph. For example, an attacker may add a new firewall rule to allow access from a normally blocked outside IP address. It would also be useful to extend the "Other" effects category to more accurately model attacks that provide limited privileges on database servers, web servers, and other networked software. Client-side attacks in which a client (e.g., web browser) is lured to or uses a malicious server to enable an attack from the server could also be modeled. Client-side attacks would require information on host client software that is not available from network vulnerability scans but may be provided by automated patch systems, network management systems, or host-based vulnerability scanners. Finally, sniffing attacks and other attacks in which information such as a password or certificate is obtained and then used later could be modeled more accurately by changing our modeling of prerequisites.

A second new direction for future work is to model the filtering performed by other network devices, such as routers. Both routers and firewalls affect traffic flow, and we need to import router configuration files to model this correctly. A third direction is to improve the analytic value of node-predictive paths by storing additional information during graph construction. This extra information could, for example, identify which host in a host group was permitted to communicate through a firewall to enable an attacker to compromise hosts in a separate subnet. A fourth direction is to improve the efficiency of the recommendation algorithms. A final direction is to obtain and use more information on network infrastructure devices, including switches, hubs, and virtual private network (VPN) devices.

GLOSSARY OF TERMS

Administrator Privilege The privilege level of an administrator or root user on a single computer. This is provided by a remote-to-administrator attack on a host.

Attack Graph A graph that shows the hosts that can be affected by one or more attackers starting from one or more network locations, the level of privilege obtained on each host, and the sequences of actions taken to obtain these privilege levels. Nodes represent the state of network hosts (including infrastructure and user hosts) and attacker privilege levels on hosts. Edges represent specific attacker actions, exploits, or attack components used to gain additional privileges.

DoS (Denial of Service) The effect of an attack when the attack disables a host or service running on that host.

Dynamic Host Collapse (DHC) An approach to simplify attack graph construction that groups hosts with equivalent inbound compromisability into host groups. DHC places hosts into host groups when they have equivalent inbound compromisability. In other words, if two hosts can be compromised at the same level (user, administrator, other, DoS) from every host the attacker has already compromised, then the hosts are equivalent from the point of view of the attack graph. A host group is treated like a single host by the attack graph generator.

Edge An edge between nodes in an attack graph that represents specific attacker actions, exploits, or attack components used to gain additional privileges. It usually represents using a specific vulnerability to compromise a host.

Filtered Reachability Domain A group of hosts is said to compose a filtered reachability domain when the hosts' reachability to all other hosts accessed through traffic filtering devices is identical.

Full Attack Graph An attack graph that shows all possible paths an attacker can use to compromise all hosts from a specific location.

Host Asset Value The value of a host to an attacker. It represents the value captured or gained by an attacker who compromises a host, including loss of confidentiality, availability, and integrity.

Host-Compromised Attack Graph An attack graph that contains all hosts that can be compromised by an attacker from a specific location. Hosts are compromised using the shortest sequence of vulnerabilities and stepping-stone hosts.

Host Group In a node-predictive graph, hosts are grouped together into host groups when they have equivalent inbound compromisability. In other words, if two hosts can be compromised at the same level (user, administrator, other, DoS) from every host the attacker has already compromised, then the hosts are equivalent from the point of view of the attack graph.

Local An attack launched from one already compromised host against the same host, often to increase the privilege level of an attacker.

Local-to-admin An exploit in which an attacker on a machine with the privilege of a user elevates privilege to that of an administrator or root user.

NetSPA A system described in this report that computes attack graphs and makes recommendations to improve network cyber security. The term stands for Network Security Planning Architecture.

Network A collection of subnets and network devices, such as routers and gateways, that connect them.

Network Address Translation (NAT) A type of network traffic manipulation in which the source address, destination address, or both are changed as the traffic passes through a firewall or other device. For example, a packet with source address 1.2.3.4 and destination 5.6.7.8 may be changed by a border firewall to source address 12.0.0.1 and destination address 24.0.0.2. The originator of the traffic, 1.2.3.4, need not know that the destination has been changed. The recipient of the traffic, 24.0.0.2, does not know that the traffic came from 1.2.3.4. The translation from 1.2.3.4 to 12.0.0.1 is called Source NAT and is commonly used to conceal many machines behind a single outward-facing IP address. The translation from 5.6.7.8 to 24.0.0.2 is called Destination NAT and is commonly used to redirect incoming connections to a specific machine designated to handle the traffic.

Network Compromised Percentage (NCP) The percentage of the total host asset values across a network that is captured by an attacker. It is the sum of the host asset values across hosts on an attack graph divided by the sum of all host asset values in a network converted to a percentage.

Node-Predictive Attack Graph A variant of predictive attack graphs that prevents the attack graph size from growing exponentially as more filtering devices are added. A node-predictive graph accurately predicts the effect of completely patching any combination of nodes. It also predicts the effect of patching individual vulnerabilities on all hosts except for those that have been combined into host groups.

Occluded IP Address The IP address of a destination host on one side of a firewall is occluded from a source host on the other side of a firewall when the firewall blocks the actual IP address of the destination host, but translates a different destination IP address to that of the destination host.

Other Privilege The effect of an attack that doesn't provide user or administrator privileges or perform a DoS against a victim host, but that provides some information about the host or limited access to the host.

Predictive Attack Graph An attack graph with two critical properties. First, it determines all hosts that can be compromised for an attacker from a given starting location. Second, without regenerating the graph, it correctly predicts the effect of patching vulnerabilities and adding firewall rules that block specific TCP/UDP ports by removing the associated nodes or edges from the graph.

Reachability A property between two interfaces on hosts or network devices attached to a network that indicates when it is possible to communicate over the network between these interfaces. An analysis that determines reachability takes into account the network topology, traffic filtering devices such as firewalls, and the TCP/UDP ports used for the communication.

Reachability Matrix A matrix with rows as outbound interfaces on hosts and columns as all active inbound ports on all hosts. Each entry in the matrix indicates whether the network's topology, routing, and filtering policies permit a connection between the corresponding row interface and column inbound port.

Remote An attack launched from one already compromised host to a different target host.

Remote-to-admin (r2a) An exploit in which an attacker on a remote machine achieves the privilege of an administrator or root user on a target victim machine.

Remote-to-user (r2u) An exploit in which an attacker on a remote machine achieves the privilege of a normal user on a target victim machine.

Subnet A collection of hosts and network devices with interfaces that share the same IPv4 address space and are connected by devices which do not perform filtering. For example, a subnet of hosts using the IPv4 address space 10.0.0.0/24 can use up to 256 addresses including 10.0.0.0, 10.0.0.1, ... 10.0.0.254, 10.0.0.255. The lowest and highest addresses (10.0.0.0 and 10.0.0.255) are typically not used for hosts and this leaves 254 addresses.

Tenant Subnet A subnet in which all hosts in the subnet are behind a firewall. The firewall separates all hosts in the subnet from other external subnets and networks.

Unfiltered Reachability Domain A group of hosts is said to compose an unfiltered reachability domain when they are fully connected and there is no traffic filtering between any two hosts in the domain. The hosts can be on one subnet or on multiple subnets as long as there is no traffic filtering between subnets.

User Privilege The privilege level of a normal user on a single computer. This is provided by a remote-to-user attack on a host.

Virtual Attacker Host A host added to the network under analysis in the subnet where the attacker is located. It contains all the IP addresses that occur in any filtering device and tries to reach all other hosts from these different IP addresses to penetrate as deeply into the network as possible. It is not created when the attacker starting location is a specific IP address.

Virtual Destination Host A host added to every subnet in a network under analysis. It contains all the IP addresses in Destination Network Address Translation (DNAT) firewall rules. When attack graphs are generated, each address in the virtual destination host is treated as a host on the network with a unique IP address to exercise DNAT firewall rules.

Virtual Host A Virtual Attacker or Virtual Destination host as defined above.

REFERENCES

1. Jajodia, S., S. Noel, and B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, and A. Lazarevic, Editors, Dordrecht, Netherlands: Kluwer Academic Publisher, 2003.

2. Swiler, L.P., et al., "Computer-Attack Graph Generation Tool," in *Proceedings DARPA Information Survivability Conference & Exposition (DISCEX) II*, Los Alamitos, CA: IEEE Computer Society, 2001, pp. 307–321.

3. Nessus, *Nessus Security Scanner*, 2004.

4. Mell, P. and T. Grance, *ICAT Metabase CVE Vulnerability Search Engine*, National Institute of Standards and Technology, http://icat.nist.gov, 2002.

5. CVE, *Common Vulnerabilities and Exposures*, The MITRE Corporation, http://cve.mitre.org, 2005.

6. Turner, D., et al., *Symantec Internet Security Threat Report, Trends for January 1, 2004–June 30, 2004*, 2004.

7. CSO-magazine, *E-Crime Watch Survey*, CSO magazine/U.S. Secret Service/CERT Coordination Center, 2004.

8. Gordon, L., et al., *2004 CSI/FBI Computer Crime and Security Survey*, CSI/FBI, 2004.

9. Lippmann, R.P., S.E. Webster, and D. Stetson, "The Effect of Identifying Vulnerabilities and Patching Software on the Utility of Network Intrusion Detection," in *Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002*, A. Wespi, G. Vigna, and L. Deri, Editors, Zurich: Springer Verlag, 2002.

10. Security-Focus, *Bugtraq Vulnerability Data Base*, Security Focus, http://www.securityfocus.com, 2005.

11. Internet-Security-Systems, *X-Force Database*, Internet Security Systems, http://xforce.iss.net/, 2005.

12. Lippmann, R.P., L. Kukolich, and E. Singer, "LNKnet: Neural Network, Machine Learning, and Statistical Software for Pattern Classification," *Lincoln Laboratory Journal*, **6**(2), 1993, pp. 249–268.

13. Fyodor, *Nmap stealth port scanner*, 2002.

14. NIST, *Standards for Security Categorization of Federal Information and Information Systems*, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2003.

15. Ritchey, R., B. O'Berry, and S. Noel, "Representing TCP/IP Connectivity for Topological Analysis of Network Security," in *Proceedings of the 18th Annual Computer Security Applications Conference*, Las Vegas, NV, 2002.

16. Tidwell, T., et al., "Modeling Internet Attacks," in *Proceedings of the Second Annual IEEE SMC Information Assurance Workshop,* United States Military Academy (West Point, NY, USA, June 2001), IEEE Press, 2001, pp. 54–59.

17. Templeton, S. and K. Levitt, "A Requires/Provides Model for Computer Attacks," in *Proceedings of the 2000 Workshop on New Security Paradigms*, New York: ACM Press, 2001.

18. Sheyner, O., et al., "Automated Generation and Analysis of Attack Graphs," in *2002 IEEE Symposium on Security and Privacy*, Oakland, CA, 2002.

19. Artz, M., *NETspa, A Network Security Planning Architecture*, Cambridge, MA: Massachusetts Institute of Technology, 2002.

20. Dawkins, J. and J. Hale, "A Systematic Approach to Multi-Stage Network Attack Analysis," in *Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)*, IEEE Computer Society, 2004.

21. Ortalo, R., Y. Deswarte, and M. Kaaniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," *IEEE Transactions on Software Engineering,* **25**(5), 1999, pp. 633–650.

22. Cuppens, F., "Alert Correlation in a Cooperative Intrusion Detection Framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC: IEEE Computer Society, 2002.

23. Ning, P. and D. Xu, "Learning attack strategies from intrusion alerts," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, New York: CM Press, 2003, pp. 200–209.

24. Lippmann, R.P. and K. Ingols, *An Annotated Review of Past Papers on Attack Graphs*, Lexington, MA: MIT Lincoln Laboratory, 2005.

25. Ammann, P., D. Wijesekera, and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, New York: ACM Press, 2002, pp. 217–224.

26. Noel, S., et al., "Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs," in *Proceedings of the 19th Annual Computer Security Applications Conference,* Las Vegas, NV, 2003.

27. Noel, S. and S. Jajodia, "Managing Attack Graph Complexity Through Visual Hierarchical Aggregation," in *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, New York: ACM Press, 2004.

28. Wool, A., "A Quantitative Study of Firewall Configuration Errors," *IEEE Computer*, **37**(6), 2004, pp. 62–67.

29. Cohen, G., M. Meiseles, and E. Reshef, *System and Method for Risk Detection and Analysis in a Computer Network*, Skybox Security Ltd.: USA, 2004.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
5 October 2005	Project Report	

4. TITLE AND SUBTITLE
Evaluating and Strengthening Enterprise Network Security Using Attack Graphs

5a. CONTRACT NUMBER
FA8721-05-C-0002

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
R.P. Lippmann, K.W. Ingols, C. Scott, K. Piwowarski, K.J. Kratkiewicz,
M. Artz, R.K. Cunningham

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420-9108

8. PERFORMING ORGANIZATION REPORT NUMBER
IA-2

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
US Air Force HQ CPSG/NIS
230 Hall Blvd., Suite 218
Lackland AFB, TX 78243-7056

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ESC-TR-2005-064

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Assessing the security of large enterprise networks is complex and labor intensive. Current security analysis tools typically examine only individual firewalls, routers, or hosts separately and do not comprehensively analyze overall network security. We present a new approach that uses configuration information on firewalls and vulnerability information on all network devices to build attack graphs that show how far inside and outside attackers can progress through a network by successively compromising exposed and vulnerable hosts. In addition, attack graphs are automatically analyzed to produce a small set of prioritized recommendations to enhance network security. Field trials on networks with up to 3,400 hosts demonstrate the ability to accurately identify a small number of critical stepping-stone hosts that need to be patched to protect against external attackers. Simulation studies on complex networks with more than 40,000 hosts demonstrate good scaling. This analysis can be used for many purposes, including identifying critical stepping-stone hosts to patch or protect with a firewall, comparing the security of alternative network designs, determining the security risk caused by proposed changes in firewall rules or new vulnerabilities, and identifying the most critical hosts to patch when a new vulnerability is announced. Unique aspects of this work are new attack graph generation algorithms that scale to enterprise networks with thousands of hosts, efficient approaches to determine what other hosts and ports in large networks are reachable from each individual host, automatic data importation from network vulnerability scanners and firewalls, and automatic attack-graph analyses to generate recommendations.

15. SUBJECT TERMS
Attack graph, attack tree, vulnerability, network, security, firewall, stepping-stone, protect, recommendation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	Same as Report	93	