Chapter 5

TOPOLOGICAL ANALYSIS OF NETWORK ATTACK VULNERABILITY

Sushil Jajodia, Steven Noel, Brian O'Berry
Center for Secure Information Systems, George Mason University

Abstract: To understand overall vulnerability to network attack, one must consider attacker exploits not just in isolation, but also in combination. That is, one must analyze how low-level vulnerabilities can be combined to achieve high-level attack goals. In this chapter, we describe a tool that implements an integrated, topological approach to network vulnerability analysis. Our Topological Vulnerability Analysis (TVA) tool automates the labor-intensive type of analysis usually performed by penetration-testing experts. It is ideal for inexpensive what-if analyses of the impact of various network configurations on overall network security. The TVA tool includes modeling of network security conditions and attack techniques (exploits), automatic population of models via the Nessus vulnerability scanner, and analysis of exploit sequences (attack paths) leading to specific attack goals. Moreover, the tool generates a graph of dependencies among exploits that represents all possible attack paths without having to enumerate them. This representation enables highly scalable methods of vulnerability analysis, such as computing network configurations that guarantee the security of given network resources. Finally, this chapter describes some of the open technical challenges for the TVA approach.

Key words: Network vulnerability analysis, network attack modeling, network hardening

1. INTRODUCTION

There are a number of tools available that can scan a network for known vulnerabilities. But such tools consider vulnerabilities in isolation, independent of one another. Unfortunately, the interdependency of vulnerabilities and the connectivity of networks make such analysis limited.

While a single vulnerability may not appear to pose a significant threat, a combination of such vulnerabilities may allow attackers to reach critical network resources.

Currently available tools generally give few clues as to how attackers might actually exploit combinations of vulnerabilities among multiple hosts to advance an attack on a network. After separating true vulnerabilities from false alarms, the security analyst is still left with just a set of known vulnerabilities. It can be difficult even for experienced analysts to recognize how an attacker might combine individual vulnerabilities to seriously compromise a network. For larger networks, the number of possible vulnerability combinations to consider can be overwhelming.

In this chapter, we describe a tool that implements a powerful topological approach to global network vulnerability analysis. Our Topological Vulnerability Analysis (TVA) tool considers combinations of modeled attacker exploits on a network and then discovers attack paths (sequences of exploits) leading to specific network targets. The discovered attack paths allow an assessment of the true vulnerability of critical network resources. TVA automates the type of labor-intensive analysis usually performed by penetration-testing experts. Moreover, it encourages inexpensive "what-if" analyses, in which candidate network configurations are tested for overall impact on network security.

In implementing TVA, we collect extensive information about known vulnerabilities and attack techniques. From this vulnerability/exploit database, we build a comprehensive rule base of exploits, with vulnerabilities and other network security conditions as exploit preconditions and postconditions.

In the network discovery phase of TVA, network vulnerability information is automatically gathered and correlated with the exploit rule base. In the analysis phase, we submit the resulting network attack model to a custom analysis engine. This engine models network attack behavior based on exploit rules and builds a graph of precondition/postcondition dependencies among exploits. The result is a set of attack paths leading from the initial network state to a pre-determined attack goal.

The next section describes the network attack problem, and Section 3 reviews related work. Section 4 describes how TVA specifically addresses the network attack problem. Section 5 applies TVA to the optimal hardening of a network, and Section 6 discusses some of the TVA technical challenges. Section 7 summarizes and concludes this chapter.

2. NETWORK ATTACK PROBLEM

We consider the complex problem of analyzing how attackers can combine low-level vulnerabilities to meet overall attack goals. Solving this problem involves modeling networks in terms of their security conditions, modeling atomic attacker exploits as transition rules among security conditions, and computing combinations of atomic exploits that lead to given network resources.

In this problem, we model the various security conditions a_i of a network as binary variables. In particular, the values model the conditions necessary for the attacker's success. For example, if some a_i represents a vulnerable version of a particular software component, $a_i = 1$ means the component exists and $a_i = 0$ means it does not. Under an assumption of monotonicity[1], a condition may transition from false to true but not back to false. That is, once a condition contributes to the success of an attack, it will always do so.

Next, we model the success of some attacker exploit $s_j \equiv s_j(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$ as a Boolean function of some set of conditions. For simplicity and without loss of generality, we model s_j as a conjunction, i.e., $s_j(a_{i_1}, a_{i_2}, \ldots, a_{i_k}) = a_{i_1} \wedge a_{i_2} \wedge \cdots \wedge a_{i_k}$. If an exploit involves disjunction (e.g. more than one version of a vulnerable program), we simply divide the disjunctive portions into separate conjunctive exploits. The success of an exploit s_j then induces some set of new conditions to become true, i.e., $s_j(a_{i_1}, a_{i_2}, \ldots, a_{i_k}) = 1$ implies $a_{p_1} = 1, a_{p_2} = 1, \ldots, a_{p_q} = 1$. In other words, s_j is a mapping from $s_j^{\text{pre}} = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$ (s_j's *preconditions*) to $s_j^{\text{post}} = \{a_{p_1}, a_{p_2}, \ldots, a_{p_q}\}$ (s_j's *postconditions*) such that if all the preconditions in s_j^{pre} are true then all the preconditions in s_j^{post} become true.

Given a network attack model, the next step is to determine how the application of exploits (in terms of security conditions) impacts network vulnerability. This step involves discovering combinations of exploits that lead to the compromise of a given critical resource. That is, some security condition a_{goal} is designated as the goal of the attack. An *attack path* is then a sequence of exploits $s_{j_1}, s_{j_2}, \ldots, s_{j_l}$ that leads to a_{goal} becoming true. Of particular interest are *minimal* attack paths, such that all exploits in the path are necessary for achieving the attack goal.

Attack paths can help network administrators determine the best way to harden their networks. To ensure complete security, all attack paths must be accounted for. Some approaches in the literature do not report all paths, while other approaches explicitly enumerate all of them. For scalability, what is needed is a representation that allows the (implicit) analysis of all possible attack paths without explicitly enumerating them. For example, in

terms of network hardening, it is sufficient to know that a particular exploit is required for all possible paths, without explicitly generating all of them.

In network hardening, it is also necessary to distinguish between two types of network security conditions. One type appears only as exploit preconditions. The only way that such conditions can be true is if they are true in the initial network conditions, since they are postconditions of no exploit. These initial conditions are precisely the ones we must consider for network-hardening measures. The other type of condition appears as both exploit preconditions and postconditions. We can safely disregard such conditions for network hardening, since attacker exploits can potentially make them true despite our hardening measures.

Given a set of initial conditions $A_{\text{init}} = \{c_1, c_2, \ldots, c_k\}$, we therefore wish to compute assignments of condition values (hardening measures) in A_{init} that guarantee the safety of a set of goal conditions $A_{\text{goal}} = \{g_1, g_2, \ldots, g_p\}$, i.e., $g_i = 0, \forall i$. Moreover, we wish to compute hardening measures that minimize assignments of $c_i = 0$, since such assignments generally have some cost associated with them, e.g., the application of a security patch or the disabling of a service.

3. PREVIOUS APPROACHES

Several aspects of the TVA problem have been studied previously. While these studies have tended to focus on specific TVA-related subproblems, our goal is to develop TVA to its full potential.

For example, Swiler *et al.* [5] presents a tool for generating network attack graphs. In our TVA tool, we apply an alternative attack graph representation that is considerably more efficient, making the graphs feasible for larger networks. Templeton and Levitt [6] and Dawkins *et al.* [7] describe approaches for specifying attacks that are similar in spirit to our exploit modeling. These approaches focus primarily on modeling, but we include a subsequent analysis phase.

The application of model checking for network attack models was first proposed by Ritchey and Ammann [8]. More recently, Sheyner *et al.* [9] modified the Symbolic Model Verifier (SMV) model checker to find all possible attack paths rather than a single attack path.

We experimented with SMV as an initial TVA analysis engine, because we could deploy it off the shelf. But scalability problems with SMV led us to develop a custom analysis engine. Our analysis engine applies an efficient graph-based representation of exploit dependencies, as described in Section 4.2. The application of such a representation to network vulnerability analysis was first described by Ammann *et al.* [10].

A central aspect of TVA modeling is connectivity among machines. A layered connectivity structure is needed to represent the various network architectures and protocols. Our connectivity model mirrors the Transmission Control Protocol/Internet Protocol (TCP/IP) reference model and is described in more detail in [11].

4. DESCRIPTION OF TVA TOOL

In this section we describe our TVA tool for analyzing vulnerability to network attacks. The description includes the modeling of network attacks and the analysis of network attack models for discovering attack paths to given critical resources.

Figure 5-1 shows the overall architecture of our TVA tool. There are three components: (1) a knowledge base of modeled exploits, (2) a description of a network of interest, and (3) a specification of an attack scenario (attacker target, initial attack control, and network configuration changes). The TVA analysis engine merges these three components and then discovers attack paths (exploit combinations) based on the merged model.

We model exploits in terms of their preconditions and postconditions. That is, each exploit is a rule in which the occurrence of a particular set of preconditions induces a particular set of postconditions. The resulting set of exploit rules comprises an attack knowledge base. The exploits in the knowledge base are generic, i.e., independent of any particular network.

A network discovery component gathers configuration and connectivity information to produce a TVA network description. Here we use "network discovery" in a more general sense, i.e., it may include traditional network discovery tools, vulnerability scanners, and code to convert such tool outputs to a TVA network description. The network description and exploit knowledge base share a common name space, which enables the mapping of generic exploits to actual network elements.

4.1 Modeling Network Attacks

Keeping pace with evolving threats and vulnerabilities requires an on-going effort in collecting information on network attacks that can be leveraged for TVA. The set of exploit rules in the TVA knowledge base must be comprehensive and up to date, since discovered attack paths will contain only those exploits that are actually included in the knowledge base.

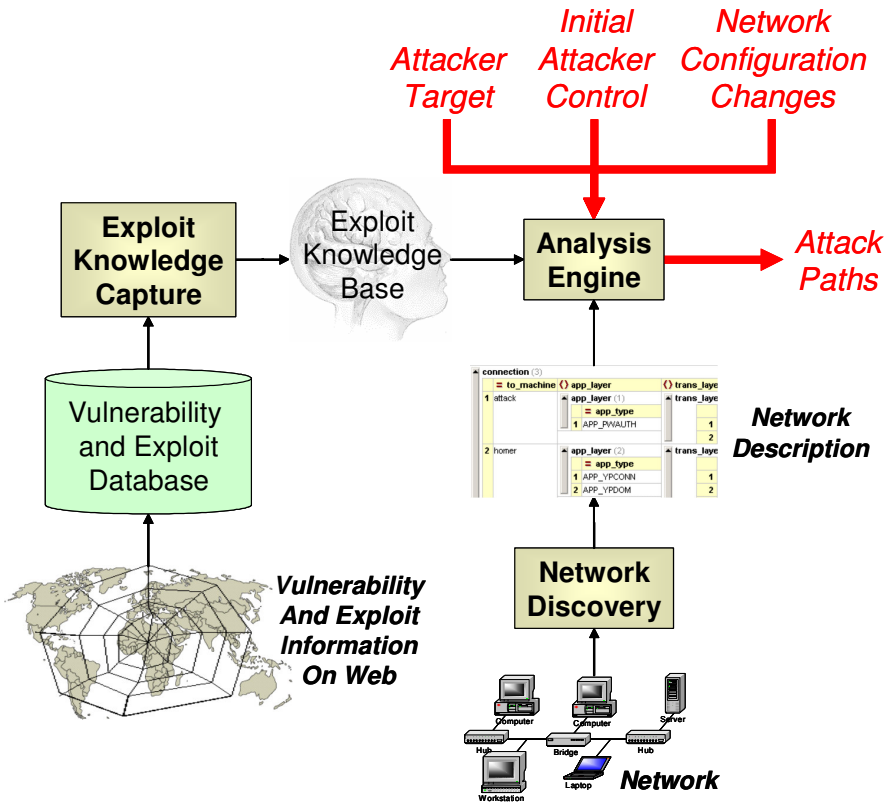

Figure 5-1. TVA Architecture

Once raw information related to network attacks is gathered, we model it in terms of exploit preconditions/postconditions. For comprehensive and accurate results, this modeling requires a good understanding of attacker strategies, techniques, and tool capabilities. Exploit conditions can be any generic attributes that potentially impact network security.

Our TVA model structure is a hierarchical framework that serves as a taxonomy of model elements. The TVA model structure evolved as exploits were developed for various types of vulnerabilities. The evolving structure supports the effects of firewalls and other connectivity-related devices. Also important is the modeling of machine groups, such that a successful attack against one group member applies equally to other machines in the group[2].

In our experience, the TVA model structure in Figure 5-2 is flexible enough to address a full range of vulnerability types and network configuration variations. For example, we have implemented exploit rules for traffic sniffing, password capturing and cracking, file transfers, command shell access, X Window access, secure shell (ssh) public key authentication,

buffer overflows that grant elevated user privileges, port forwarding, machine identity spoofing, and denial-of-service attacks.

In the next paragraph, we begin describing a way to automatically populate *network* models for TVA. However, it is much more difficult to automatically populate sets of modeled *exploits*. In particular, it is difficult to automatically capture the semantics needed for exploit preconditions and postconditions, because the vulnerability-reporting community has defined no standard formal language for specifying such semantics. Instead, databases of reported vulnerabilities usually rely on natural language text to describe vulnerabilities and ways of exploiting them. We have begun investigating how exploit semantics can be specified via web-based ontologies.

For TVA to be practical for real networks, it is important to automate the network discovery process. We have integrated our TVA tool with the open-source Nessus [1] vulnerability scanner. Nessus maps known vulnerabilities to network machines, reporting scan results using the eXtensible Markup Language (XML) [2]. The XML representation allows us to leverage the eXtensible Stylesheet Language (XSL) [3] to easily convert Nessus output to TVA input (which is also in XML).

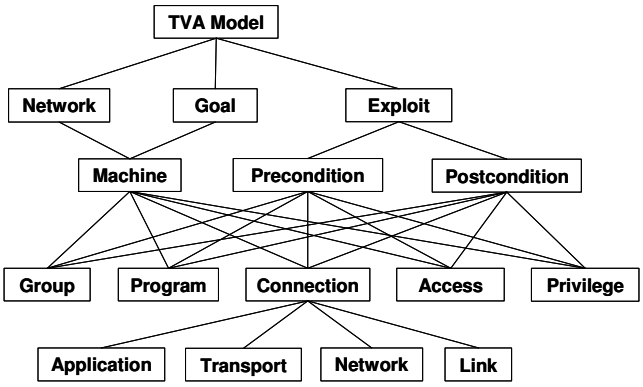

Figure 5-2. TVA Model Structure

To transform a Nessus report into a TVA network description, each reported Nessus vulnerability is cross-referenced against a list of known exploits. If a match is found, the Nessus vulnerability is applied as the name of a machine-connection precondition in the resulting network description. Nessus-based exploits may also have preconditions and/or postconditions for access type (e.g., execute or file transfer access) and privilege level (e.g., user or super user).

TVA maintains network connectivity details in separate tables that describe each machine's connections to the rest of the network. This means

that firewalls don't have to be modeled directly because the individual host tables implicitly address their effects. However, multiple Nessus scans are required to correctly populate the connectivity tables when firewalls are present. In general, a separate Nessus scan is required for each network segment to which a firewall connects.

The network generation process merges the external and internal Nessus scans into a single coherent network description. The two-stage (external and internal) dataflow diagram for this process is shown in Figure 5-3. This process can be generalized in a straightforward fashion to handle arbitrary numbers of separate network segments.

In the first step of this process, Nessus generates a vulnerability report for each network segment. In the second step, the Nessus report XML is processed against a Nessus cross-reference (nidxref.xsl), written in XSL. The second step optionally inserts configuration-specific information (contained in config.xsl) as specified by the TVA user. The nidxref.xsl stylesheet is produced by the Nessus exploit generation process described below. This stylesheet enables the network description to be optimized so that it contains only those Nessus connections for which exploits have been developed.

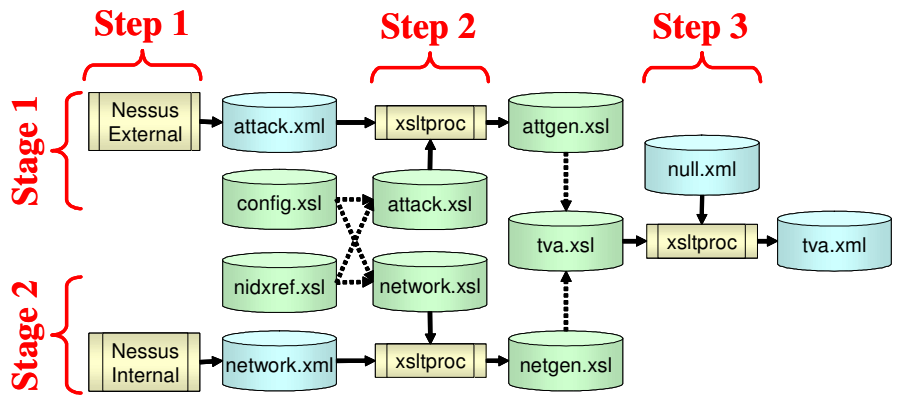

Figure 5-3. Generation of Network Description via Nessus

The last step merges the intermediate files from the second step into a single network description (tva.xml) that also incorporates an attack goal specification from the TVA user. The null.xml document is a dummy file that satisfies the XSL processor requirement [4] for an input XML file.

The process for generating TVA exploits from Nessus is shown in Figure 5-4. It begins with Nessus *plugins*, which contain the detailed information that Nessus needs for detecting vulnerabilities. We have developed a program (np2xp) to convert the Nessus plugins list into XML.

The resulting plugins.xml is then processed against the conditions.xsl stylesheet. This stylesheet is produced manually through researching the plugin information, e.g., consulting the relevant data in our vulnerability/exploit database. As we discussed earlier in this section, it is difficult to totally automate this manual step. The processing against conditions.xsl inserts the preconditions and postconditions developed through this exploit-modeling process. Finally, the resulting exploits.xml is transformed into Java modules and compiled into the TVA analysis engine. This process also generates the Nessus identification cross-reference file (nidxref.xsl) described earlier, which is in turn used to generate TVA network descriptions from Nessus scans.

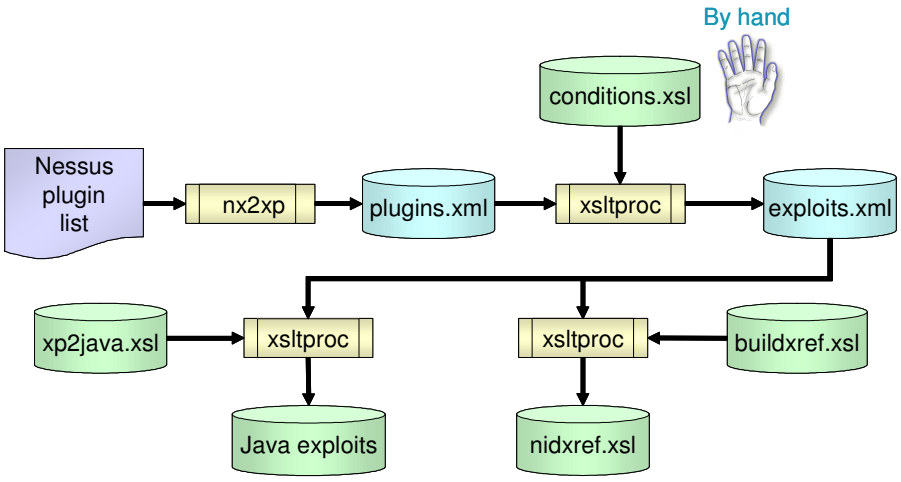

Figure 5-4. Generation of Exploits via Nessus

4.2 Network Attack Analysis

Given a particular TVA model (network description and set of exploits), we analyze the model to discover attack paths to critical network resources. From these attack paths we can then derive an expression for network safety in terms of the initial configuration. This safety expression in turn supports decisions about hardening the network against attacks.

We begin with a set of exploits $S = \{s_1, s_2, \ldots\}$ in terms of security conditions $A = \{a_1, a_2, \ldots\}$. These exploits and conditions conform to the modeling framework described in Section 4.1. The network attack model (network conditions and exploits) can be built by hand, automatically generated, or a combination of both.

The attack paths we compute are based on a directed graph of the dependencies (via preconditions and postconditions) among exploits and conditions. One way is to represent conditions as graph vertices and exploits as (labeled) graph edges. The dual of this representation is also possible, with exploits as graph vertices and conditions as labeled graph edges.

We employ a third representation that is a bit more flexible. This representation has both conditions and exploits as vertices. Edge labels then become unnecessary, with directed edges simply representing generic dependency. In this representation, a dependency edge $e = (a, s)$ going from condition a to exploit s means that s depends on a, i.e., a is a precondition of s. Similarly, a dependency edge $e = (s, a)$ going from exploit s to condition a means that a depends on s, i.e., a is a postcondition of s.

We build the dependency graph through a multi-step process. We first build the set of all exploits $S_{\text{exec}} \subset S$ that can be successfully executed by the attacker. Working from S_{exec}, we then build a dependency graph D_{init} starting from the initial condition exploit s_{init}. That is, we start from s_{init}, search S_{exec} for exploits whose preconditions match the postconditions of s_{init}, add exploit dependencies for any s_{found} found, and then remove s_{found} from S_{exec}. We continue by iteratively adding dependencies to D_{init} by searching S_{exec} and removing s_{found} from S_{exec}. The resulting graph D_{init} represents forward dependencies from s_{init}, i.e., exploits in D_{init} are those that are forward-reachable from s_{init}.

Next we do a backward traversal of the forward-reachable dependency graph D_{init}, starting from the attack goal exploit s_{goal}. The resulting dependency graph D includes exploits that are not only reachable from the initial conditions, but are also relevant to (i.e., reachable from) the attack goal. In fact, D comprises the necessary and sufficient set of exploits with respect to the initial and goal conditions, i.e., all exploits can be executed, and all exploits contribute to the attack goal. Thus D represents the set of minimal attack paths, in which no exploit can be removed without impacting the overall attack.

Given a dependency graph D, we then construct an expression that concisely represents all possible attack paths. This construction involves the recursive algebraic substitution of exploits (via precondition/postcondition dependencies) in the backward direction, starting from the goal-condition exploit s_{goal}. That is, we start from s_{goal} and algebraically substitute it with the conjunction of its preconditions, i.e. $s_{\text{goal}} \rightarrow \{a_{\text{goal}_1}, a_{\text{goal}_2}, \ldots, a_{\text{goal}_k}\}$.

We then substitute each of the goal-condition preconditions a_{goal_i} with the exploit that yields it as a postcondition, since these are logically equivalent. In the event that more than one exploit yields this postcondition,

we form the disjunction of all such exploits, since logically any one of them could provide the postcondition independent of the others.

We continue in a recursive fashion, substituting the newly generated exploit expressions in the same way we treated the goal-condition exploit expression. In doing this recursive algebraic substitution, we make direct use of the exploit-condition dependency graph by traversing it breadth first. Once the dependency graph has been fully traversed, the result is a concise expression that represents all possible attack paths to the goal.

Initial-condition assignments of false mean that the corresponding network services are unavailable. It is desirable to choose assignments with minimal impact on network services. We can immediately choose one assignment over another if all of its disabled services also appear disabled in the other set. This choice is desirable because the selected set represents a comparative increase in available services. Moreover, this choice is neutral with respect to relative priorities of network services, since no service is disabled in the chosen set in comparison to the other.

This analysis yields all possible hardening measures (sets of initial-condition assignments) that have minimal impact on services. The analyst can now compare the various sets and select the one that offers the best combination of offered services.

5. EXAMPLE TVA APPLICATION

In this section, we demonstrate by example how TVA combines vulnerabilities in a network to find attack paths to a particular goal. We then analyze the TVA results to determine the best way to harden the network against attack.

In this example, a restrictive firewall protects the machines that support public web and email services, as shown in Figure 5-5. This example shows how connectivity-limiting devices affect the TVA model and how vulnerable services on a network can be exploited even when direct access to services is blocked.

The firewall implements the following policy to restrict connectivity from the attack machine:

1. Incoming ssh traffic is permitted to both *maude* and *ned*, although only *ned* is running the service (this is a common practice under the assumption that it is safe because ssh is a secure protocol);
2. Incoming web traffic is permitted only to *maude*, which is running Microsoft's Internet Information Server (IIS);
3. Incoming email is permitted to *ned*, which is running the sendmail server;

4. Incoming File Transfer Protocol (FTP) traffic is blocked because *ned* is running the wu_ftpd server, which has a history of vulnerabilities;
5. All outgoing traffic is permitted (this is a common practice under the assumption that outgoing traffic won't harm the internal network).

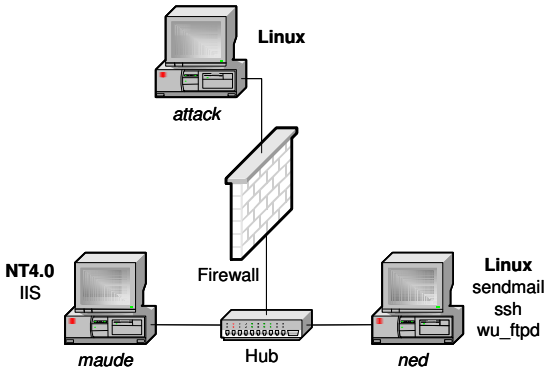

Figure 5-5. Network Diagram for Example TVA Application

The attack goal for this example is to obtain super user (root) access on *ned*. This is not directly possible because (1) no known exploits exist for the version of sendmail running on *ned*, and (2) the firewall blocks access to the vulnerable wu_ftpd service from the attack machine. The question now is whether the attack goal can be realized indirectly, i.e., through a sequence of multiple exploits.

The initial locus of attack is on the attack machine, since only that machine has user access and privilege defined as an initial condition, via the TVA network description. In general, the initial attack machine will also tend to have a complete set of programs used by the exploits in the model. Network connectivity is represented at the machine level by listing all possible connections from the given machine to all other destination machines in the network description. The effect of a firewall or other connectivity-limiting device is to reduce the size of each machine's connectivity table, but such devices generally will not appear as specific machines in the network description unless they run their own services to which other machines can connect. For this scenario, the firewall did not support any such services.

The attack goal is represented in the network description as a particular set of resources on a particular machine (the goal machine could appear elsewhere in the network description, with any set of initial conditions defined for it). In this example, we are only testing whether execute access (the ability to run programs) with super user (root) privilege can be obtained

on *ned*. However, in general it is possible to test any other conditions, such as the appearance of any new connectivity or program in its configuration.

Figure 5-6 shows the resulting TVA attack graph for this example. For clarity, the specific exploit preconditions and postconditions are omitted from the figure, but they are described in Table 5-1. Despite the firewall policy designed to protect it, the external attacker obtains execute access with super user privilege on *ned*. The attack graph shows that the initial exploitation of the IIS vulnerability on *maude* ultimately leads to the compromise of *ned*, e.g., the following:

1. The IIS Remote Data Services (RDS) exploit enables the attacker to execute programs on *maude*;
2. Given the access provided by the IIS RDS exploit, the remote copy[3] (rcp) program on *maude* is executed to download a rootkit[4] from the *attack* machine;
3. A port-forwarding program from the rootkit is then executed to set up access from the *attack* machine through *maude* to the FTP service on *ned*;
4. Finally, the wu_ftpd exploit is executed through the forwarded connection against *ned* to obtain root access there.

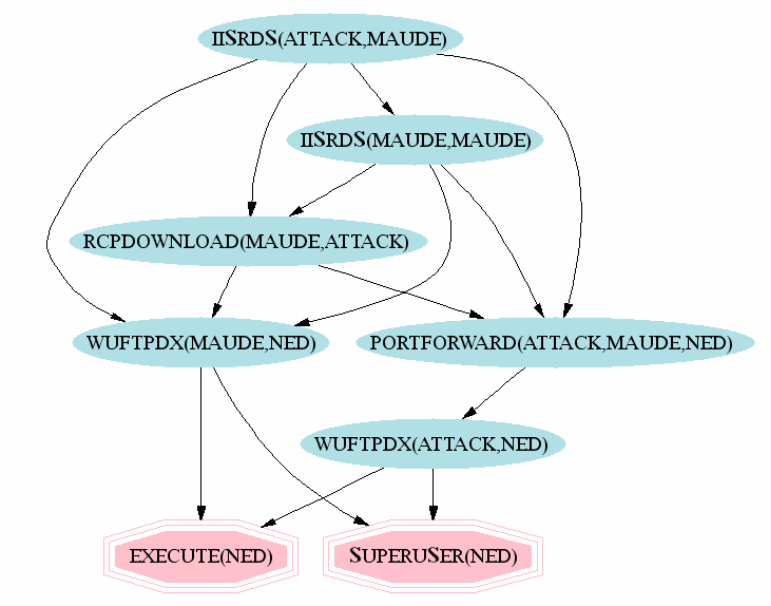

Figure 5-6. Attack Graph for Example Application of TVA

Finding such attack paths is a unique TVA capability. No commercial tool connected outside the firewall is currently capable of reporting more

than an IIS vulnerability on *maude*. Connected inside the firewall, a commercial tool would also report the vulnerable wu_ftpd service, but human analysis would still be required to build an attack path from the outside through *maude* to *ned*. This would be an easy enough exercise for an experienced penetration tester working on such a small network. But it would be infeasible for networks in which voluminous outputs must be analyzed manually for large numbers of machines.

Table 5-1. Exploits for Example Application of TVA

Exploit	Description	Preconditions	Postcondition
IISRDS	One of many exploits associated with Microsoft's Internet Information Server (IIS)	1. Execute access on attack machine 2. Attack machine has connectivity to IIS service on victim	Ability to execute programs on victim at super user privilege level
RCPDOWNLOAD	Binds rsh access to the ability to transfer programs (e.g., rootkits) from victim machine using the rcp program	1. Execute access on attack machine 2. rcp program on attack machine 3. Attack machine has connectivity to victim's rsh service	Copies victim machine's programs to attack machine
WUFTPDX	Yields super user on many Unix platforms that run the Washington University FTP daemon, wu-ftpd	1. Execute access on attack machine 2. wu-ftpd exploit program exists on attack machine 3. Attack machine has connectivity to FTP service on victim	Super user execute access on victim
PORTFORWARD	Enables attacker to work around firewall when foothold obtained on an internal machine. One of few exploits that implements "middleman" machine to direct exploits against victim machine.	1. Middleman and victim are different machines (implicit, not in attack graph) 2. Execute access on middleman 3. Port-forwarding program on middleman 4. Attacker connectivity to transport-layer (unused) port on middleman	Attacker acquires middleman's transport layer connectivity to victim

From a TVA attack graph, we can immediately compute an expression for the attack-goal conditions in terms of the initial conditions. This process involves traversing the attack graph in a backwards direction, algebraically substituting exploits with those exploits that satisfy their preconditions. This computation is done recursively, with the recursion ending when an exploit's precondition is an initial condition.

As we explained in Section 2, the only conditions relevant to network hardening are the initial conditions. An expression $g(c_1, c_2, \ldots, c_k)$ for the attack goal in terms of initial conditions $C_{\text{init}} = \{c_1, c_2, \ldots, c_k\}$ then provides a way to determine if a particular network configuration is guaranteed safe with respect to the attack goal. From the particular form of g, we can determine the safe assignments of A_{init}.

Figure 5-7 again shows the TVA attack graph for this example, this time with the initial conditions included. For convenience, the figure includes algebraic symbols that correspond to our analysis of network hardening. In particular, exploits are denoted by Greek letters, and initial conditions are denoted by c_i.

By examining Figure 5-7, we can traverse the attack graph backwards, starting from the goal condition g, and recursively perform algebraic substitution according to precondition/postcondition dependencies.

$$
\begin{aligned}
g &= \delta + \phi \\
&= (\alpha + \beta)\chi c_6 + \varepsilon c_8 c_9 \\
&= (\alpha + \alpha c_3)\chi c_6 + (\alpha + \beta)\chi c_7 c_8 c_9 \\
&= \alpha(\alpha + \beta)c_4 c_5 c_6 + \alpha(\alpha + \beta)c_4 c_5 c_7 c_8 c_9 \\
&= \alpha c_4 c_5 c_6 + \alpha c_4 c_5 c_7 c_8 c_9 \\
&= c_1 c_2 c_4 c_5 c_6 + c_1 c_2 c_4 c_5 c_7 c_8 c_9 \\
&= c_1 c_2 c_4 c_5 (c_6 + c_7 c_8 c_9)
\end{aligned}
\qquad (1)
$$

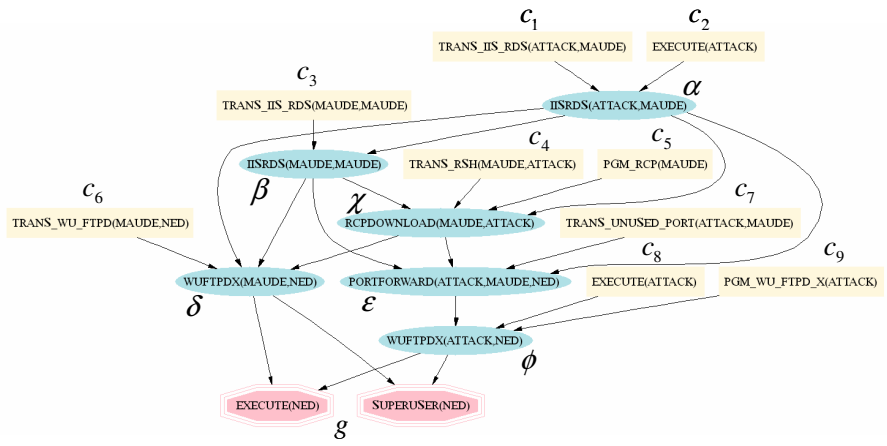

Figure 5-7. Attack Graph with Exploit Preconditions Included

In terms of the problem domain, some initial conditions are outside the network administrator's control. In particular, the administrator has no control over conditions like programs and user access/privilege on the attacker's machine. Thus we have $c_2 = c_8 = c_9 = 1$, so that Eq. (1) becomes

$$g = c_1 c_4 c_5 (c_6 + c_7) \tag{2}$$

From Eq. (2), four assignments of initial conditions are apparent that provide network safety. While other safe assignments are also possible, these four minimize the cost of hardening the example network:

1. Patch or disable the IIS RDS web server on *maude* ($c_1 = 0$);
2. Disable outgoing rsh from *maude* ($c_4 = 0$);
3. Remove the rcp program from *maude* ($c_5 = 0$);
4. Patch or disable wu_ftpd from *maude* to *ned*, and block all unused ports on *maude* ($c_6 + c_7 = 0$).

When considered separately, each of these four options has a minimal hardening cost, in the sense that no hardening measure can be ignored without jeopardizing the attack goal. The network administrator can then choose the option that has overall minimum cost, based on the relative costs of the individual hardening measures.

6. TECHNICAL CHALLANGES

The TVA modeling framework supports the full range of network and exploit information needed for realistic scenarios. But to make TVA feasible for large networks, automatic model generation methods are needed.

As described in Section 4.1, we currently create TVA network descriptions via the Nessus vulnerability scanner. But Nessus lacks the ability to provide certain types of information. For example, with Nessus we must assume that firewalls enforce generic policies for the individual network segments. Although this may be an acceptable approximation of firewall effects, real policies often include host-specific rules.

While host-specific rules could be handled by individual Nessus scans from each machine in the network, this procedure is not very efficient. A more efficient solution would be to build TVA models directly from firewall filter tables. Also, while transport and application layer information is available from Nessus, additional topology information is needed to delineate between the link and network TCP/IP layers.

Although Nessus can guess a remote machine's operating system, it is not always correct and often cannot determine a specific version. Many

exploits depend on detailed information about the operating system. Vulnerabilities are often removed by applying a patch to the applicable operating system or application. Patch-level information is therefore required for accurate exploit modeling.

Nessus scans for vulnerabilities from a remote location, so it can only detect network service information. However, many vulnerabilities are local and are not exploitable or detectable over a network. Processes are required to gather program-specific information from individual hosts, e.g., from host configuration files. For example, some trust relationship and group membership information is difficult to obtain remotely. This information is valuable for TVA, to determine whether an exploit is really possible or whether it affects machines other than the immediate target.

As one can imagine, TVA attack graphs might become huge for large, poorly secured networks. Analytical and visual methods are necessary for handling such (quadratic) attack-graph complexity, such as aggregating parts of the graph as summary information or culling parts of the graph not of immediate interest. We have developed a prototype drill-down visualization tool that shows great promise in solving the attack graph management problem.

A current bottleneck for TVA implementation is the process of modeling exploits manually. The problem is that much of the domain knowledge is available only as natural-language text. What is needed are exploit specifications written in a standard, machine-understandable language.

It appears that this requirement can be met by the emerging Semantic Web [12] under development by the World Wide Web Consortium. The vision is that web content of the future will be defined and linked in a way that it can be used for automation, integration, and reuse across various applications, not just for display purposes as with Hypertext Markup Language (HTML). With the Semantic Web, standardized rule-based markup provides the actual semantics (meaning) for web content.

TVA has potential application beyond penetration testing and network hardening. For example, it can be applied to the tuning of intrusion detection systems. In practice, network administrators must often balance the risk of attack against the need to offer services. Even with network hardening guided by TVA, administrators may still decide to tolerate some residual network vulnerability from services they absolutely need. The intrusion detection system could be configured to consider only this residual vulnerability and thus generate alarms only in the context of genuine threats to critical network resources.

At a minimum, vulnerabilities that do not significantly contribute to overall risk can be ignored, reducing the effective false-positive rate. It may

also be possible to infer new intrusion signatures from TVA results, in turn increasing the number of true positive detections.

But there is a limit to what can be accomplished with network hardening and intrusion detection. The need to offer services is at odds with network hardening, and effective intrusion detection will remain challenging, particularly in the face of novel attacks.

To augment methods of avoidance and detection, TVA can be applied to attack response, both defensive and offensive. For defensive response, the network is dynamically hardened in the face of attacks. A less conservative approach is to launch an offensive counterattack in response to an attack against one's own network. While approach may be extreme, it could be the only available option for allowing a network to function after being attacked.

7. SUMMARY AND CONCLUSIONS

This chapter describes a tool for Topological Vulnerability Analysis (TVA), a powerful approach to global network vulnerability analysis. The tool analyzes dependencies among modeled attacker exploits, in terms of attack paths (sequences of exploits) to specific network targets. While the current generation of commercial vulnerability scanners generates voluminous information on vulnerabilities considered in isolation, they give little clues as to how attackers might combine them to advance an attack.

The tool automates the type of labor-intensive analysis usually performed by penetration-testing experts, providing a thorough understanding of the vulnerabilities of critical network resources. It encourages inexpensive what-if analyses of the impact of candidate network configurations on overall network security.

Also, the tool employs a comprehensive database of known vulnerabilities and attack techniques. This database includes a comprehensive rule base of exploits, with vulnerabilities and other network security conditions serving as exploit preconditions and postconditions.

During TVA network discovery, network vulnerability information is gathered and correlated with exploit rules via the open-source Nessus vulnerability scanner. Our custom TVA analysis engine then models network attack behavior based on the exploit rules, building a graph of precondition/postcondition dependencies. This graph provides attack paths leading from the initial network state to a specified goal state. From the attack graph, we can determine safe network configurations with respect to the goal, including those that maximize available network services.

Our TVA tool provides powerful new capabilities for network vulnerability analysis. It enables network administrators to choose network

configurations that are provably secure and minimize the cost of network hardening. TVA also has potential application to other key areas of network security, such as identifying possible attack responses and tuning intrusion detection systems.

ACKNOWLEDGEMENTS

We gratefully acknowledge the software development efforts of Michael Jacobs in support of this chapter. The work of Sushil Jajodia was partially supported by the Virginia Commonwealth Information Security Center (www.cisc.jmu.edu).

NOTES

1. In the context of network security, our assumption of monotonicity is quite reasonable. It simply means that once an attacker gains control of a resource, he need never relinquish it in order to further advance the attack. In other words, no backtracking is necessary.
2. An example of machine group effects is that guessing a Windows NT domain user password would probably allow login to all machines in the domain.
3. The rcp program is installed by default with Windows NT 4.0.
4. A "rootkit" is a hacker term that refers to tools an attacker often transfers to a compromised machine for the purpose of expanding access or escalating privileges.

REFERENCES

1. R. Deraison, *Nessus*, retrieved May 2003, from http://www.nessus.org, 1999.
2. World Wide Web Consortium®, *Extensible Markup Language (XML)*, retrieved May 2003, from http://www.w3.org/XML/, 2003.
3. World Wide Web Consortium®, *The Extensible Stylesheet Language (XSL)*, retrieved May 2003, from http://www.w3.org/Style/XSL/, 2003.
4. World Wide Web Consortium®, *XSL Transformations (XSLT) Version 1.0*, retrieved May 2003, from http://www.w3.org/TR/xslt, 1999.
5. L. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-Attack Graph Generation Tool," in *Proceedings of the DARPA Information Survivability Conference & Exposition II*, Anaheim, California, June 2001.
6. S. Templeton, K. Levitt, "A Requires/Provides Model for Computer Attacks," in *Proceedings of the New Security Paradigms Workshop*, Cork, Ireland, September 2000.
7. J. Dawkins, C. Campbell, and J. Hale, "Modeling Network Attacks: Extending the Attack Tree Paradigm," in *Proceedings of the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Johns Hopkins University, June 2002.
8. R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, May 2000.

9. O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, May 2002.
10. P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of CCS 2002: 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.
11. R. Ritchey, B. O'Berry and S. Noel, "Representing TCP/IP Connectivity for Topological Analysis of Network Security," in *Proceedings of 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2002.
12. World Wide Web Consortium®, *Semantic Web*, retrieved May 2003, from http://www.w3.org/2001/sw/, 2003.