ATTACK GRAPH COMPRESSION

TAO LONG

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS

SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

MARCH 2009

Your file Votre référence
ISBN: 978-0-494-63328-1
Our file Notre référence
ISBN: 978-0-494-63328-1

ABSTRACT

Attack Graph Compression

Tao Long

Attack graph has emerged as a useful tool for defending against multi-step network attacks involving correlated vulnerabilities. However, most current representations of attack graphs are not scalable [35]. Even the attack graph of a reasonably large network is usually incomprehensible to the human eyes. For realistic networks with tens of thousands of hosts and hundreds of vulnerabilities, even computing the attack graph may become infeasible. On the other hand, an attack graph of a real-world network usually has much redundancy due to the presence of hosts with similar configurations, such as those in an office or computer lab. To out best knowledge, existing work can at best hide such scalability issues through visualization techniques but cannot remove the redundant information, which does not comprise real solutions.

This thesis presents a scalable representation of attack graphs for removing such redundancy. The representation is based on a well known compression technique, namely, reference encoding. More precisely, we use one host as the reference to other hosts with similar vulnerabilities and connectivity; details of the latter can then be omitted in the resultant attack graph. We introduce our compression model step by step. We start with a simple case where hosts have identical connectivity and vulnerabilities. We show that a one-host model can be used in some cases but it has limitations in representing remote exploits across different machines. We then introduce a two-node model to address the limitation and show that the one-host model is actually a special case of the two-node model. Next, we study the more realistic case where hosts may have different connectivity and vulnerabilities. We show that in some cases small differences are better hidden in textual rules while in other cases the differences are better handled by leaving the involved hosts outside the compression model. To evaluate the proposed compression model, we will describe a case study on a small network. We will also show experimental results based on random network topologies generated by existing tools. Both results confirm that our model can significantly reduce the complexity of attack graphs.

Acknowledgments

I would like to express my appreciation to all the people who have ever helped and instructed me for this thesis and in my classes. In particular, I thank Mr. Ji Lou for his help on the experimental section of this thesis.

I would like to express my sincere thanks to my supervisor, Dr. Lingyu Wang, for his help, guidance and advice to my study at Concordia University. I must say, his guidance allows me to experience the joy of research, which will benefit me for the rest of my life.

Finally, I am very grateful to my parents, brother, wife and sister for their constant supports. Without their inspiration and encouragement, I would not have been able to tackle this challenging research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Today's computer systems play a critical role in almost every sector of society. Such systems constitute the central component of information technology infrastructures in enterprises and in critical infrastructures including power grids, financial data systems, and emergency communication systems. Protecting these systems against network intrusions is crucial to the economy and to national security. However, the scale and severity of intrusions have continued to grow at an ever-increasing pace despite over twenty years of research in vulnerability analysis and intrusion detection [2, 8].

In the everlasting war against attackers, security researchers and administrators seem to always fall behind their opponents in technology. Firewalls and access control mechanisms may thwart intrusion attempts made by amateur attackers, but these same mechanisms can be easily circumvented by experienced attackers. Intrusion detection systems (IDSs) and vulnerability scanners may help system administrators to identify incidents or threats of individual attacks, but such systems are usually unaware of the relationships among attacks. An attacker can gradually elevate his/her privileges through multiple interdependent attacks on intermediate victims before finally reaching the attack goal. Such a cleverly crafted *multi-step attack* creates nightmares to system administrators because it is usually difficult to manually identify correlated attacks from the large volume of intrusion alerts.

To defend against multi-step attacks, researchers have recently proposed techniques for correlating isolated alerts and vulnerabilities (Chapter 2 will describe related work in more details). In particular, a *vulnerability-centric* approach to the defense against multi-step intrusions reflects the ancient saying: *Know Your Enemy, Know Yourself, Fight A Hundred Battles, Win A Hundred Battles*. Such an approach combines the knowledge about one's own network with the knowledge about ongoing attacks for hardening a network and for correlating and predicting attacks.

A popular model of correlated network vulnerabilities is called *attack graph*. An attack graph is usually represented as a directed graph with two types of vertices corresponding to exploits, and the pre and post-conditions of exploits, respectively. Directed edges point from the pre-conditions to an exploit and from the exploit to its post-conditions. Figure 1 shows a toy example of attack graphs. We assume a simple scenario where a file server (host 1) offers the File Transfer Protocol (ftp), secure shell (ssh), and remote shell (rsh) services; a database server (host 2) offers ftp and rsh services. The firewall only allows ftp, ssh, and rsh traffic from a user workstation (host 0) to both servers. In the attack graph, exploits of vulnerabilities are depicted as predicates in ovals and conditions as predicates in clear texts. The two numbers inside parentheses denote the source and destination host, respectively. The attack graph represents three self-explanatory sequences of attacks (attack paths). For example, the right path is: $sshd\_bof(0,1) \rightarrow ftp\_rhosts(1,2) \rightarrow rsh(1,2) \rightarrow local\_bof(2)$.

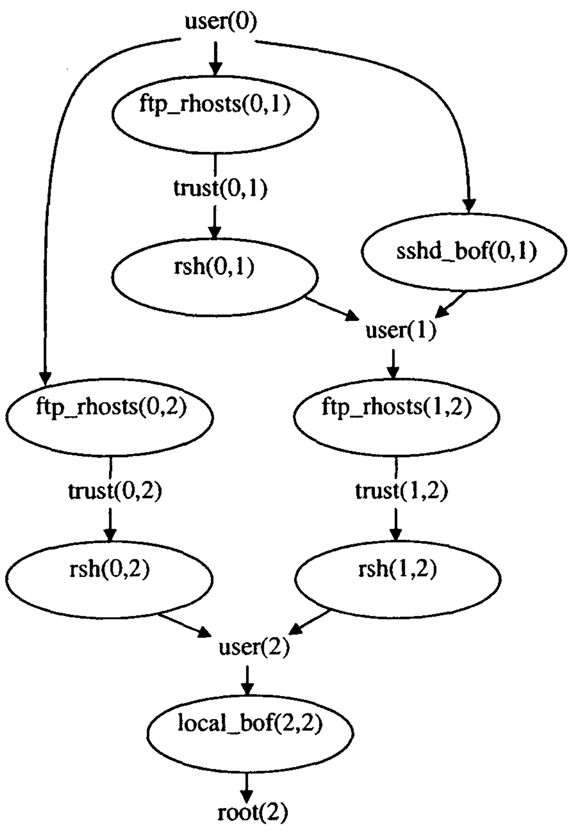

Figure 1: An Example of Attack Graph

However, even attack graphs for a reasonably large network could be incomprehensible to the human eye. The number of vertices in an attack graph is at least quadratic in the number of hosts multiplied by the number of vulnerabilities, because each exploit involves a source host, zero or more intermediate hosts, and a target host. Consequently, the size of an attack graph, which is quadratic in the number of its vertices, grows quickly with the number of hosts and vulnerabilities. For large networks with tens of thousands of hosts and hundreds of vulnerabilities, even computing the attack graph may become infeasible. In contrast to the meaningful model depicted in Figure 1, Figure 2 shows an attack graph model for a subnet of 14 hosts, with less than 10 vulnerabilities on each machine.

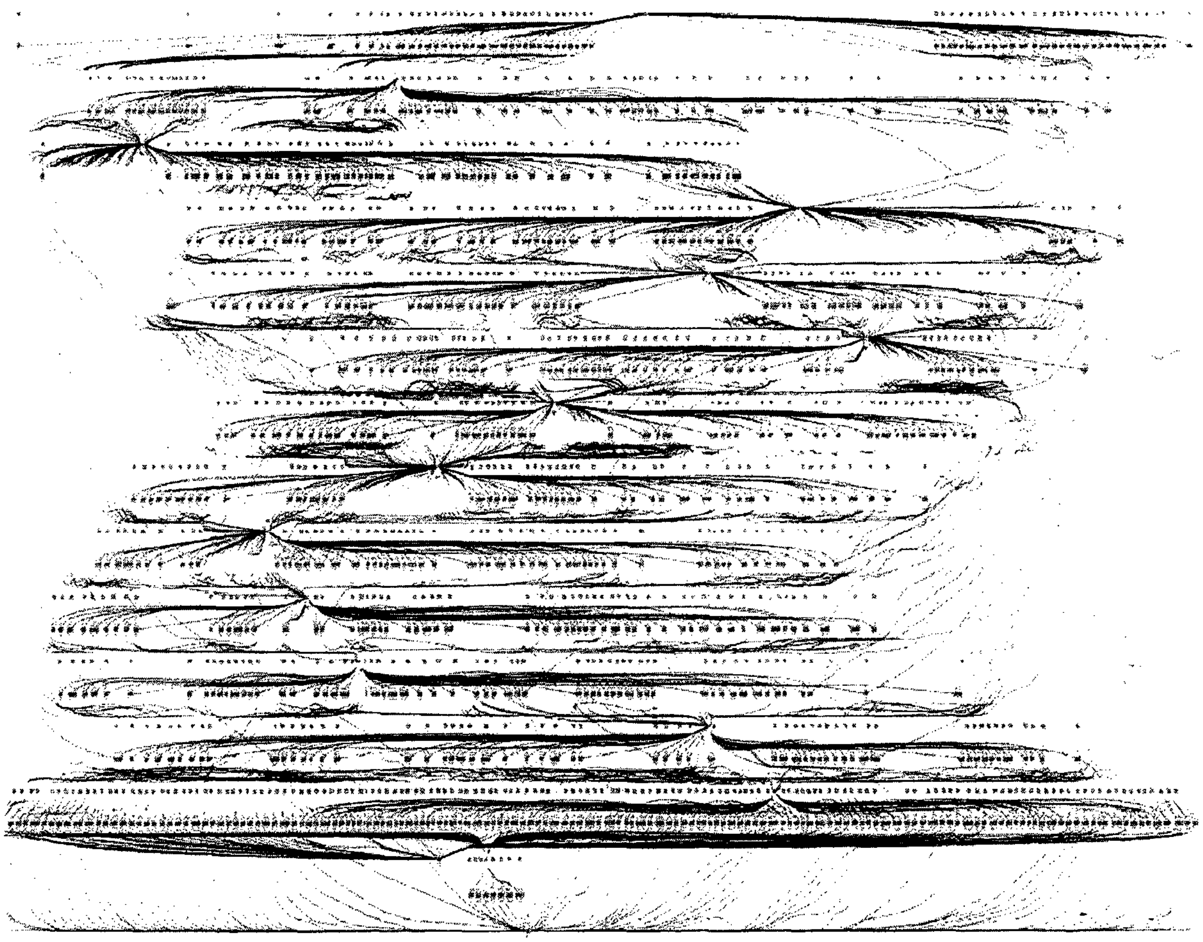

Figure 2: An Attack Graph of a 14-Machine Network [43]

On the other hand, the explosion in size of attack graphs is not entirely inevitable. In fact, an attack graph may actually carries a large amount of redundant information. In the scenario depicted by Figure 1, if the hosts 1 and 2 have the same vulnerabilities and network connectivity, then it follows that whatever happens between host 0 and host 1 can also happen between host 0 and host 2, and vice versa. In practice, it is common for a network to have a large number of hosts with the same or similar configurations and connectivity, such as those in offices, computer labs, server farms, etc. Computing an attack graph for such a network using current representation would introduce much redundant information into the result. Such redundancy may practically render a meaningful attack graph incomprehensive and difficult to generate, analyze, and manage.

1.2 Summary of Contributions

This thesis will present a novel scalable representation of attack graphs. The representation is based on a graph compression technique, namely, *reference encoding*. This technique has previously been used in different applications, such as compressing web graphs [1] where one URL may be used as the reference for another URL with similar external links. This allows details about the latter to be omitted in the resultant graph. We borrow this intuition for compressing attack graphs such that redundancy caused by similar configuration and connectivity can be reduced. However, special requirements in compressing attack graphs lead to a few subtleties in the current problem that prevent any trivial application of the reference encoding and many other standard graph compression methods.

First, the compressed attack graph must be meaningful and can reveal similar threats as in the original version. That is, the main goal of the compression of attack graphs is not to reduce storage requirements. By examining the compressed attack graph, a security administrator should be able to identify any kinds of attack strategies that he may identify from the full attack graph, even though some details about how such strategies may be applied to specific hosts may be missing. For example, reference encoding cannot be applied to Figure 1 simply by removing h1 or h2, because what happens between these two hosts would no longer be observable in the result. Second, the compression must be lossless in the sense that it should allow all attack sequences to be recovered, if necessary, as they can be from the original attack graph. This capability would completely eliminate the need for generating the original attack graph.

Figures 3 provides a high level overview of our approach. At the upper left corner are the given inputs, that is, network configuration and domain knowledge about vulnerabilities. The upper right corner depicts an attack graph that is generated from those inputs using existing methods (keep in mind the attack graph is typically very large in size although here we only show a small example). The lower left corner denotes our compression model, which is derived from the given inputs. The lower right corner is desired analysis results, such as a sequence of attacks leading to a given condition.

The rest of this thesis will explain the details of the proposed compression model and how we can compute the model from given inputs. We introduce our model in several stages. First, we start with a simple case where hosts have identical configuration and connectivity. We then extend the model to more complicated cases where the configuration and connectivity may slightly vary among different hosts. Finally, we consider the case where a network is consisted of several densely connected subnets and only limited connectivity

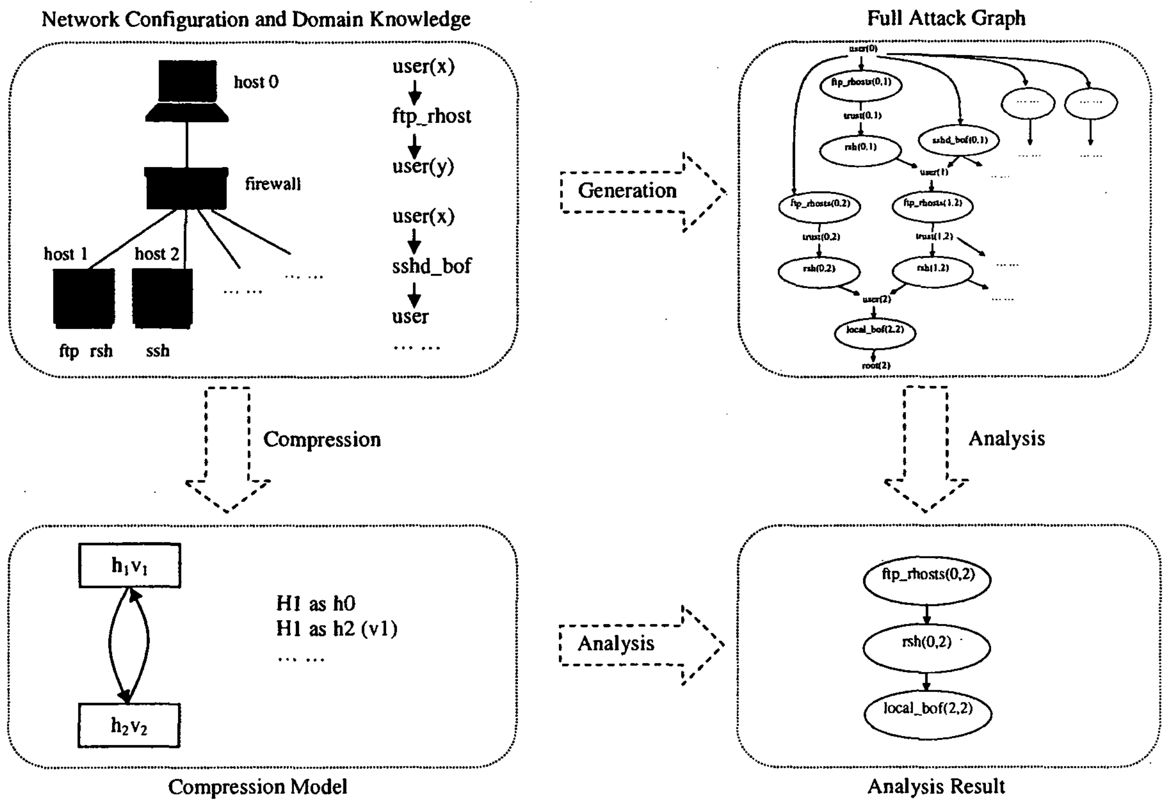

Figure 3: An Overview of Our Approach

exists between different subnets.

As we shall show, the compression model will be significantly smaller in size than the full attack graph, which enables an administrator to easily identify potential sequences of attacks leading to the compromise of a network. Moreover, we shall show that the compression is lossless in that sense that our compression model will indicate exactly the same sequences of attacks as the full attack graph does. Finally, we shall also discuss examples of obtaining the same analysis result using our compression model as with the full attack graph. In particular, we will discuss how our compression model can be used for finding paths that may be taken by attackers in compromising given victim hosts.

1.3 Organization

The rest of the thesis is organized as the follows. Chapter 2 reviews relevant related work. Chapter 3 devises our compression model and discusses its applications to attack graph analysis. Chapter 4 presents a case study and describes our implementation and experiments. Chapter 5 concludes the thesis.

Chapter 2

Literature Review

In this chapter, we review related works on defending against multi-step attacks, including attack graph, alert correlation, security metrics. We also review related work on graph compression techniques. We discuss why existing work are not satisfactory with respect to the problems addressed in this thesis.

2.1 Attack Graph

Various vulnerability scanners, such as Nessus [18], can find known vulnerabilities in a network. However, they cannot reveal how such vulnerabilities can be combined in a multi-step attack to infiltrate a network. To evaluate the security of a network against multi-step attacks, a security analyst must take into account the effects of interactions between different vulnerabilities and find global security flaws. Traditional vulnerability analysis of a network typically involves heavy human intervention by the so-called red team. First, they use vulnerability scanners to identify vulnerabilities on individual hosts. Combining such identified vulnerabilities with other information about the network, such as connectivity between hosts, the red team produces sequences of attacks, namely, attack paths. Each attack path leads to an undesirable state, for example, a state where the intruder has administrative accesses to a critical host. The limitation of the red team approach is that its effectiveness heavily depends on the skills of the team; the manual process is error-prone, tedious, and not scalable for large networks.

Many early efforts address the defense against multi-step network attacks [13, 20, 46, 71]. The general concept of attack tree is mentioned in [59] where trees with logical connectives AND and OR are used as a formal methodology for analyzing the security of systems. A detailed attack graph model is described in [52] and a tool is proposed to build an attack graph using forward search in [63]. The inputs of an attack graph include configuration files, attacker profiles, and a database of attack templates, which must be manually created. The nodes of the attack graph are attack templates instantiated with particular users and machines while edges are labeled by probabilities of success or cost of attacks. The graphs are analyzed to find the shortest paths between given start and end nodes. The idea of grouping similar nodes is mentioned although its correctness would critically depend on identical configuration among such nodes.

A *require and provide* approach to automatic attack graph generation is mentioned in [64], which has later been widely adopted in defending against multi-step attacks. Attack scenarios can be generated by linking subgoals through their preconditions requirements and postconditions capabilities. Each successful attack helps the attacker to gain more capabilities and move closer to the final goals. *JIGSAW*, an attack specification language is described to model attack components. The language requires low level details in terms of capabilities and requirements of attacks, which in practice may be hard to obtain. This require and provide approach brings flexibility in discovering potentially new attack scenarios.

Model checking is applied to the analysis of multi-step network attacks in [57]. Information such as known vulnerabilities on network hosts, connectivity between hosts, the initial capabilities of the attacker are described as states, while exploits as transitions between states that can be executed by attackers. This formal model is given to a model checker as the input, while the reachability in terms of given goal states is given as a query. The model checker will produce a counterexample if a sequence of exploits may lead to the goal states. Such a sequence of exploits indicates a potential attack graph that must be removed to secure the network. A later work [58] provides more details on how connectivity should be modeled at different layers. The term *topological vulnerability analysis* is introduced.

Model checking is used for a different purpose in [30, 60], that is, to enumerate all attack paths. A modified model checker is applied to the finite-state machine created from network information. The model checker can provide all counterexamples to a query stating the safety of goal states, which are essentially the collection of possible attack paths. Other types of analysis are also discussed, such as finding a *cut set* in the attack graph, such that goal conditions can no longer to reached. The problem of finding the minimum attack leading to given goal conditions is shown to be intractable. One apparent limitation of this approach is its scalability. All attack paths are explicitly enumerated in its result, which leads to a combinatorial explosion when the number of hosts or vulnerabilities increases.

To address the scalability of model checking-based approaches, a *monotonic assumption* is proposed in [3], which states that the further exploits will never cause the attacker to relinquish any obtained privileges. Attack paths can thus be implicitly modeled as paths in a directed graph that includes exactly one copy of each exploit and its pre- and post conditions, with edges interconnect exploits to these conditions. This assumption thus reduces the complexity of attack graph representation from exponential in the number of hosts to polynomial. However, it also renders some attacks that may disable services or invalidate vulnerabilities impossible to be included in the model. Attack graphs are created through a two-pass search, which first connects exploits by starting from the attacker's initial state and then prunes those irrelevant states by searching backward from the goal state. Other analyses are also discussed, such as finding minimal attacks leading to given goal conditions.

More recently, a logic programming-based approach to the representation of attack graphs is given in [49]. Datalog is used to encode knowledge about attacks in a network. MulVAL [48], a security analyzer built from off-the-shelf tools, is used to retrieve information regarding existing software on networked hosts and their vulnerabilities. The engine takes as input network configuration information and outputs attack steps that an attacker can take to compromise the network. The analysis by MulVAL has polynomial complexity with respect to the size of the network.

One of the first treatments of the scalability issue of attack graph representation is given in [43]. A hierarchical approach is taken to build rules at every level of aggregation, which are integrated through common attribute values of attack graph elements or attack graph connectivity. In such a hierarchical attack graph, attack subgraphs are recursively collapsed to single vertices. This means the compression process can be utilized to a given degree. Moreover, the abstraction of protection domain is particularly effective for complexity reduction when groups of machines have complete connectivity. A quadratic complexity is claimed for the approach. Another effort applied a matrix clustering algorithm to the adjacency matrix of attack graphs in order to construct clustered adjacency matrix, which indicate the feature of protection domain on the main diagonal [44].

Two improvements to the representation of attack graphs are given in [24]. First, a directed graph is used to model subnets as nodes and potential inter-subnet attacks as edges. A dominator tree is then used to determine whether inter-subnet and intra-subnet attacks are useful based on the domination relationships. Second, an abstraction reduces group exploits to virtual nodes in order to increase the readability of the attack graph. These two methods may reduce the complexity of visualized attack graphs and allow human users to quickly grasp imminent threats. Although those work can hide scalability issues of attack graphs from users, they do not directly remove redundant information, and are thus not satisfactory solutions in our understanding.

2.2 Other Related Work

A similar and parallel thread of work focus on reconstructing multi-step attack scenarios from isolated intrusion detection alerts, such as those employ prior knowledge about complete strategies of attacks [12, 16, 17, 19] or the causal relationships between attacks [11, 40, 41]. Some techniques aggregate alerts with similar attributes [10, 15, 61, 65] or statistical patterns [31, 53]. Hybrid approaches combine different techniques for better results [41, 54, 69]. Attack scenarios broken by missed attacks are reassembled by clustering alerts with similar attributes [42], and those caused by incomplete knowledge are pieced together through statistical analyses [53, 54].

Alert correlation techniques are also used for other purposes than analyzing multi-step intrusions, such as to relate alerts to the same thread of attacks [27]. The privacy issue of alert correlation has recently been investigated [70]. Alert correlation is employed to deal with insider attacks in [9, 56]. Efforts in integrating information from different sources include the formal model *M2D2* [38] and the Bayesian network-based approach [72]. Real-Time detection of isolated alerts is studied in [34, 51]. Some products claim to support real-time analyses of alerts, such as the Tivoli Risk Manager [26]. Designed for a different purpose, the RUSSEL language is similar to our approach in that the analysis of data only requires one-pass of processing [23].

Recently, much interest has focused on quantifying the threat of potential multi-step attacks. General reviews of security metrics are given in [4, 29]. The NIST's efforts on standardizing security metrics are given in [39] and more recently in [62] and in the Common Vulnerability Scoring System (CVSS) [37]. Another overview of many aspects of network security metrics is given in [25]. Dacier et al. gave intuitive properties that should be satisfied by any security metric [13, 14, 46]. The difficulty of attacks are measured in terms of time and efforts spent by attackers. Based on an exponential distribution for an attacker's success rate over time, they use the Markov model and the MTTF (Mean Time to Failure) to measure the security of a network. They discussed simple cases of combining individual measures but did not study the general case.

The work by Balzarotti et al. [5] focuses on computing the minimum efforts required for executing each exploit. Based the exploitability concept, a qualitative measure of risk is given in [6]. Another approach measures the relative risk of different configurations using the *weakest attacker* model, that is the least conditions under which an attack is possible [50]. Yet another series of work measures how likely a software is vulnerable to attacks using a metrics called *attack surface* [36]. These work allow a partial order to be established on different network configurations based on their relative security. However, the treatment of many aspects of security is still qualitative in nature.

Wang et al. [67] proposed a framework for using combining functions to determine the combined effect of vulnerabilities in a network. They proposed the idea of using an analogy to the resistance of electrical circuits in [68] and address the issue of additional dependency between exploits although the solution is not entirely satisfactory since cycles in attack graphs are largely ignored. Wang et al. also proposed a probabilistic network security metric based on attack graphs [21, 22, 33]. They propose the use of probability scores for each vulnerability to represent the likelihood that one attacker will exploit the vulnerability or the percentage of attackers that successfully exploit the vulnerability. Another work adopt this same concept but will use it to develop conditional probability tables for each exploit and then demonstrate how the use of DBNs can be used to determine network security. The work on minimum-cost network hardening represents an early effort toward the quantitative study of network security [66]. This work quantifies the cost of removing vulnerabilities in hardening a network, but it does not consider other hardening options, such as modifying the connectivity. It also has the limitation of adopting a qualitative view of damages (that is, all the given critical resources are equally important) and of attack resistance (that is, attacks on critical resources are either impossible or trivial).

Graph compression techniques have been applied to various applications, such as the so-called web graphs formed by URLs and links [1]. More specifically, the entire web at a particular moment forms a graph with pages as nodes and hyperlinks as the directed edges. The basis of web graph compression is the observation that web pages and links are dynamically created by finding one or more reference pages and copying links from these references. The reference encoding technique can thus be applied to conceal the details in those pages who have copied links from reference pages. This technique is extended in [55] through a two-level representation of web graphs that partitions the set of pages in the repository into a set of small directed graphs. Each such directed graph encodes a densely connected subset of pages. A top level directed graph made up of supernodes and superedges represents the relatively sparse interconnectivity between different subsets of pages. We are partly inspired by those work on compressing web graphs to apply the reference encoding concept to attack graphs. However, as we shall show, the difference between the two applications render a direct application of existing techniques infeasible.

Chapter 3

Attack Graph Compression

In this chapter, we propose a method for compressing attack graphs based on the reference encoding technique. We introduce the basic concepts in Section 3.1. We then introduce our methods step by step starting from the simple case of identical connectivity and vulnerabilities in Section 3.2 to the more realistic cases in Section 3.3.

3.1 Basic Concepts

In this section, we introduce the relevant concepts of *attack graph*, *type graph*, *configuration graph*, and the traditional way of generating attack graphs from given type graph and configuration graph.

By combining the knowledge in type graph and the facts in configuration graph, we can generate an *attack graph* as a model of inter-dependent vulnerabilities on networked hosts, which is formalized in Definition 1.

Definition 1 *An attack graph G is a directed graph $G(E \cup C, R_r \cup R_i)$ where the set of nodes include E, a set of* exploits, *and C, a set of* conditions, *and the set of edges include two relations $R_r \subseteq C \times E$ and $R_i \subseteq E \times C$.*

An attack graph is thus a directed graph whose set of nodes is partitioned into two classes, namely, *exploits* and *security conditions* (or simply *conditions*). An exploit is typically a triple (h_s, h_d, v), where h_s and h_d represent two connected hosts and v a vulnerability on the destination host h_d. A security condition is a pair (h, c), indicating the host h satisfies a condition c relevant to one or more exploits. Notice that h_s, h_d, and v are abstract notations that could in practice possess different semantics, for example, h_s and h_d can be host names, IP addresses, and so on, and v can be the name of a vulnerability or its ID in a vulnerability database.

Corresponding to the inter-dependency between exploits and conditions, the two types of edges in an attack graph have different semantics. First, the *require* relation R_r is a directed edge pointing from a condition to an exploit, which means the exploit cannot be executed unless the condition is satisfied. For example, an exploit (h_s, h_d, v) requires following two conditions, that is the existence of the vulnerability v on h_d and the connectivity between h_s and h_d. Second, the *imply* relation R_i pointing from an exploit to a condition means executing the exploit will satisfy the condition. Notice that there is no edge directly connecting two exploits (or two conditions).

Figure 4 shows a simple attack graph example. The attack graph indicates that by exploiting a buffer overflow vulnerability in the Sadmind service (Nessus ID 11841), an attacker can gain the privilege of using a remote machine.The attack graph shows an attacker having user privilege on host 3 can exploit the vulnerability on hosts 1 and 2 and obtain user privilege on the hosts. Notice that after an attacker has obtained user privilege on host 1, he/she can then exploit host 2 from either host 3 or host 1.

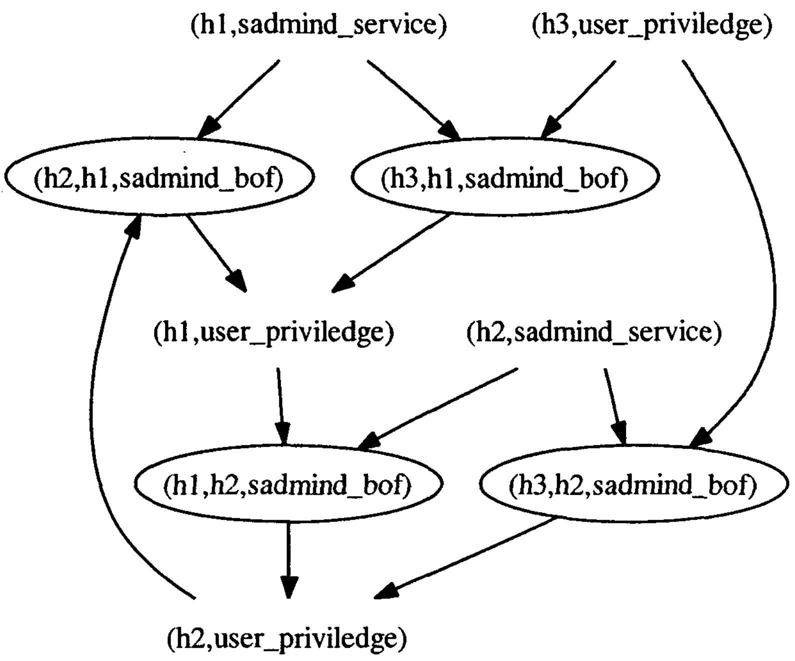

Figure 4: An Example of Attack Graph

Two important semantics of attack graphs are as follows. First, the require relation is always *conjunctive* whereas the imply relation is always *disjunctive*. More specifically, an exploit cannot be realized until *all* of its required conditions have been satisfied, whereas a condition can be satisfied by any *one* of the realized exploits. Second, the conditions are further classified as *initial* conditions (the conditions not implied by any exploit) and *intermediate* conditions. An initial condition can be independently disabled to harden a network, whereas an intermediate condition usually cannot be [45].

To generate an attack graph, two types of inputs are necessary, namely, *type graph* and *configuration graph*. Type graph represents expert knowledge about the dependency relationship between vulnerabilities. On the other hand, configuration graph represents hosts and their connectivity and vulnerability information. We assume the domain knowledge required for type graph is available from tools like the Topological Vulnerability Analysis (TVA) system, which covers more than 37,000 vulnerabilities taken from 24 information sources including X-Force, Bugtraq, CVE, CERT, Nessus, and Snort [28]. On the other hand, we assume the configuration information including vulnerabilities and connectivity can be obtained using available network scanning tools, such as the Nessus scanner [18]. We define type graph and configuration graph in Definition 2 and 3.

Definition 2 *A type graph TG is a directed graph $TG(N, E)$. The node set N is a subset of $(CT \times HT) \cup V$ where CT and V are sets of condition types and vulnerabilities, respectively, and $HT = \{source, destination\}$ indicating whether the condition is about the source or destination host. The edge set E can be partitioned into edges pointing from $CT \times HT$ to V and those from V to $CT \times HT$.*

Definition 3 *A configuration graph $CG(N, E)$ is a directed graph where $N \subseteq H \times 2^V$, H is a set of hosts, V a set of vulnerabilities, and $E \subseteq N \times N$.*

Figure 5 depicts the configuration graph and type graph required for generating the attack graph in above example (notice that we shall use lines without arrows for bidirectional connectivity between hosts hereafter). The left-hand side is a configuration graph showing the connectivity between three hosts where initially hosts 1 and 2 both have the vulnerability and the attacker has user privilege on host 3. The right-hand side is a type graph showing that an attacker who possesses user privilege on a source host may exploit the vulnerability on the destination host and thus obtain the user privilege on the latter.

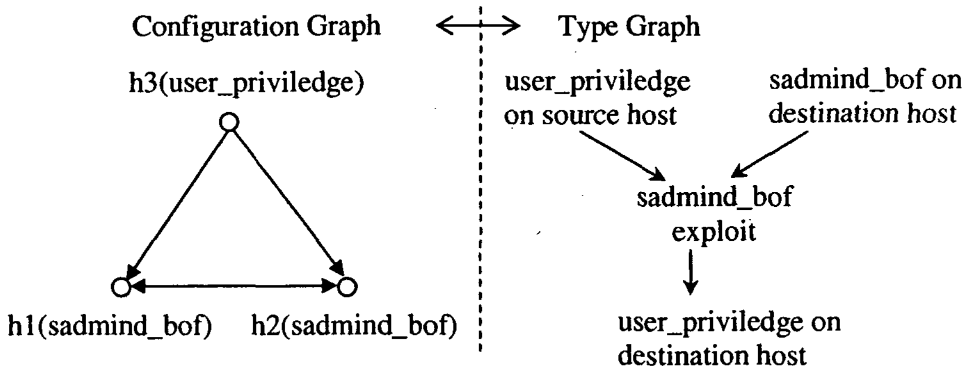

Figure 5: An Example of Type Graph and Configuration Graph

In this thesis, we shall refer to a simplified attack graph and type graph with conditions omitted, namely, *vulnerability-based* attack graph and type graph. Notice that in a vulnerability-based attack graph or type graph, we must annotate the edges to explicitly indicate the relationship between exploits to be either conjunctive or disjunctive, since such information is no longer available from the intermediate conditions.

3.2 The Complete Case

By a *complete case*, we mean all the hosts have exactly the same connectivity and vulnerabilities. In contrast, next section will consider the *incomplete case* where the connectivity or vulnerabilities (or both) may vary from one host to another. Clearly, the complete case is more straightforward, although unrealistic, and suitable for a starting point of our discussion.

3.2.1 One-Node Model

The left-hand side of Figure 6 shows a configuration graph. There are three hosts, $h1$, $h2$, and $h3$ each of which has exactly the same three vulnerabilities $v1$, $v2$, and $v3$. The three hosts are all connected to the same host ha which is assumed to be the attacker's host so no vulnerability is necessary (indicated by the symbol \$). We can interpret the graph as a collection of three independent machines each of which is separately connected to the external network. An attacker may attack each host from his/her own machine. The right-hand side of the figure shows a simple vulnerability-based type graph that indicates the dependency between the three vulnerabilities. Notice again in such a vulnerability-based type graph (or attack graph), we shall omit conditions.

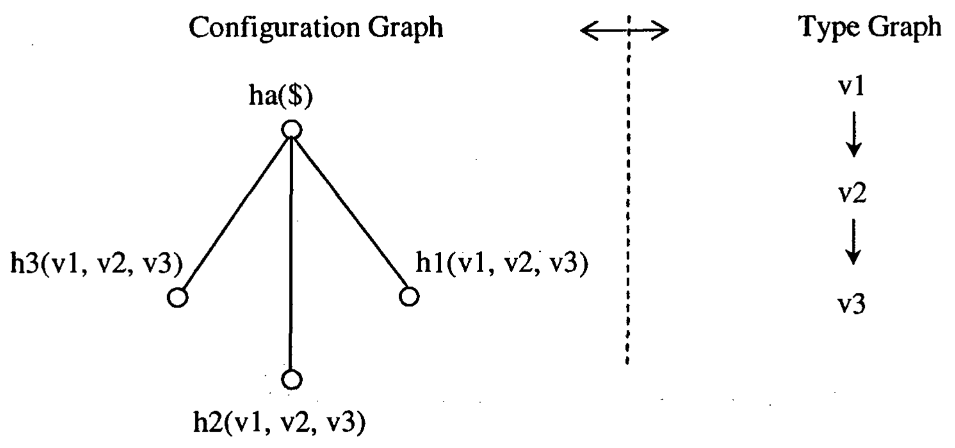

Figure 6: An Example of the Complete Case

Based on the configuration graph and type graph shown in Figure 6, it is easy to generate the attack graph. However, instead of studying the attack graph, we shall directly look at the possible attack sequences represented by such an attack graph, since the main objective of our compression model is to preserve such attack sequences. Notice that an attack graph and the collection of attack sequences the attack graph represents are equivalent in terms of revealing potential multi-step attacks. Table 1 lists all the three attack sequences that can be obtained based on the configuration graph and type graph in Figure 6. Inside each sequence we use the notation $hihjv$ to indicate an exploit of vulnerability v from source host hi to destination host hj.

Table 1: Attack Sequences

1	$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h1v3$
2	$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h2v3$
3	$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h3v3$

Clearly, in this simple case, we can use one of the three hosts as the reference of the others, namely, a *one-node compression model.* The compression model is shown in Figure 7. On the left-hand side is the compression model, which includes two components, namely, the *compressed configuration graph* and the *reference rules.* The compressed configuration graph is simply a configuration graph of smaller size. In this case, it has two hosts, the attacker's machine ha and a victim host $h1$. The reference rules indicate which host can act as the reference of which hosts. In this case, we have only one reference rule. The rule states that the host appearing before the word *as* can be replaced by any of the hosts appearing after. In the middle of Figure 7 is the original type graph. The right-hand side shows an attack graph generated from the type graph and the compressed configuration graph, namely, the *compressed attack graph.* Notice that the generation of the compressed attack graph does not employ the reference rules.

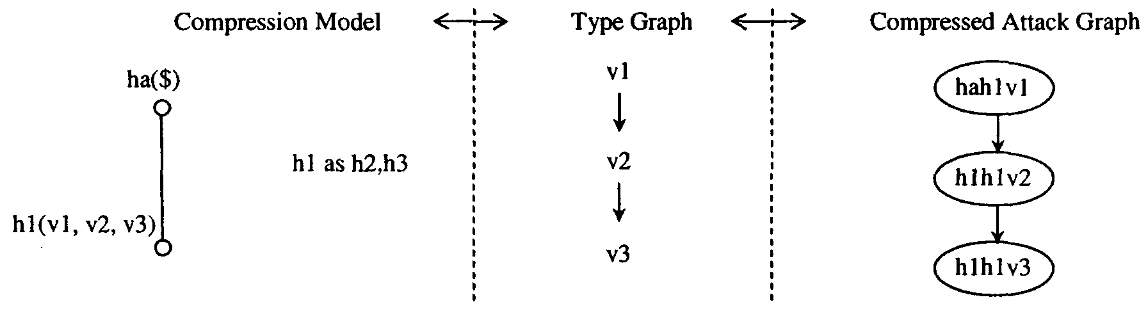

Figure 7: An Example of the Compression Model for the Complete Case

The compression model clearly satisfies the two requirements we have described in previous sections. First, we require the compression model to depict a clear and concise picture about the threat from potential multi-step attacks. In this case, the compressed attack graph only includes a single attack sequence, which reveals a path that may be followed by attackers in attacking the host $h1$. If the security administrator only wants to know what kind of attacks may happen to the network, then the compressed attack graph is sufficient for this purpose. Notice that although the compressed attack graph looks similar to the type graph in this special case, they are different concepts and will look very different in later cases. Second, we require the compression process to be lossless in the sense that the compression model should allow exactly the same set of attack sequences to be generated as in the original attack graph, which can be easily verified in this simple case.

3.2.2 Two-Node Model

Next we consider another situation of the complete case. left-hand side of Figure 8 shows a configuration graph. Again, there are three hosts, $h1$, $h2$, and $h3$ each of which has three vulnerabilities $v1$, $v2$, and $v3$. Different from the previous case, the three hosts are now fully connected to each other, and also to another host ha. We can interpret the graph as a small network of three identical machines connected to a switch and a router with no firewall, and an attacker may attack this network from his/her own machine ha. The right-hand side of the figure shows the same vulnerability-based type graph as before. Based on the configuration graph and type graph shown in Figure 8, the left column of Table 2 lists all the 27 attack sequences that can be obtained based on the configuration graph and type graph in Figure 8. We can ignore the crosslines for the time being, which will be explained shortly.

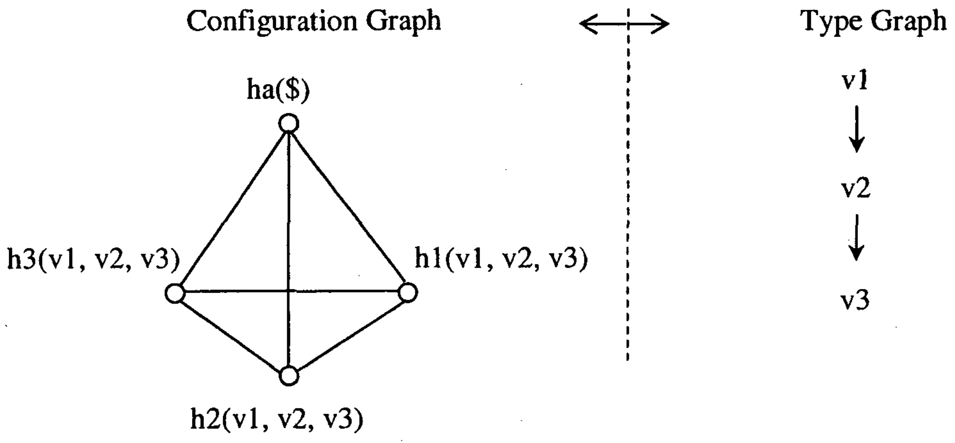

Figure 8: An Example of the Complete Case

Clearly, even for this simple network with three victim hosts, the number of attack sequences is now significant due to the connectivity between the victim hosts. This number will certainly increase very fast as the size of network increases. On the other hand, since the three victim hosts all have the same set of vulnerabilities and connectivity, the 27 attack

Table 2: Attack Sequences (Crosslines Indicate Sequences with Local Exploits)

1	~~$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h3v3$~~
2	$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h3v3$
3	~~$ha\$ \rightarrow hah1v1 \rightarrow h1h3v2 \rightarrow h3h3v3$~~
4	$ha\$ \rightarrow hah2v1 \rightarrow h2h1v2 \rightarrow h1h3v3$
5	~~$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h3v3$~~
6	~~$ha\$ \rightarrow hah2v1 \rightarrow h2h3v3 \rightarrow h3h3v3$~~
7	$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h3v3$
8	$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h3v3$
9	~~$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h3v3$~~
10	$ha\$ \rightarrow hah1v1 \rightarrow h1h3v2 \rightarrow h3h1v3$
11	$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h1v3$
12	~~$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h1v3$~~
13	~~$ha\$ \rightarrow hah2v1 \rightarrow h2h1v2 \rightarrow h1h1v3$~~
14	~~$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h1v3$~~
15	$ha\$ \rightarrow hah2v1 \rightarrow h2h3v3 \rightarrow h3h1v3$
16	~~$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h1v3$~~
17	$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h1v3$
18	~~$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h1v3$~~
19	$ha\$ \rightarrow hah1v1 \rightarrow h1h3v2 \rightarrow h3h2v3$
20	~~$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h2v3$~~
21	~~$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h2v3$~~
22	$ha\$ \rightarrow hah2v1 \rightarrow h2h1v2 \rightarrow h1h2v3$
23	~~$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h2v3$~~
24	$ha\$ \rightarrow hah2v1 \rightarrow h2h3v3 \rightarrow h3h2v3$
25	$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h2v3$
26	~~$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h2v3$~~
27	~~$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h2v3$~~

sequences apparently include much redundancy. We can certainly apply the same one-node compression model as in the previous case to use $h1$ as the reference of the other two hosts. However, recall that a unique requirement for compressing attack graph is the compressed result should itself be meaningful, and can reveal the same threat posed by multi-step attacks as the original attack graph does. This requirement makes our task slightly more complicated.

A more careful consideration will reveal the limitation of the one-node compression model in this case. More specifically, we need to distinguish between *local exploits*, which indicate the exploits originates from and ends at the same machine, and *remote exploits*, which are launched between different hosts. In Table 2, the cross lined entries represent attack sequences that involve at least one local exploit, while the others represent attack sequences involving only remote exploits. Clearly, with $h1$ as the reference to both $h2$ and $h3$, we will not be able to distinguish between those two types of attack sequences.

We thus use a two-node compression model for the complete case, as shown in Figure 9. On the left-hand side is the compression model, which includes the compressed configuration graph and the reference rules. The compressed configuration graph is simply a configuration graph of smaller size. In this case, it has three hosts, the attacker's machine ha and two victim hosts $h1$ and $h2$. The vulnerabilities on $h1$ and $h2$ have been specially designed to minimize redundancy while preserving possible attack sequences, as we shall show shortly. The reference rules indicate which host can act as the reference of which hosts. In this case, we have two reference rules. Each rule states the host appearing before the word *as* can be replaced by any of the hosts appearing after. In the middle of Figure 9 is the original type graph. The right-hand side shows an attack graph generated from the type graph and the compressed configuration graph, that is, the compressed attack graph.

The compression model clearly satisfies the first requirement. In this case, the compressed attack graph only includes a single attack sequence, but it is sufficient to reveal the difference between remote exploits and local exploits, if such a distinction is necessary. If the security administrator only wants to know what kind of attacks may happen

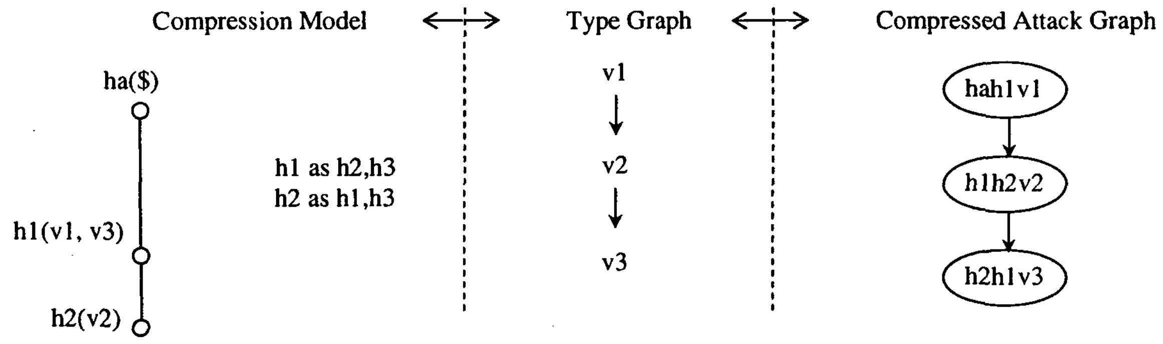

Figure 9: An Example of the Compression Model for the Complete Case

to the network, then the compressed attack graph is sufficient for this purpose. For the second requirement, the compression model should encode exactly the same information as the original attack graph. More precisely, both should allow the generation of the same collection of attack sequences.

We first verify the compression model against this requirement in Table 3 for attack sequences with only remote exploits. In the right column of each table, we list the reference rules applied to the generation of that attack sequence. We only list each rule once even if it is used multiple times. Notice that each application of a reference rule is only effective to one occurrence of an exploit in a sequence. For example, in Table 3, attack sequences number 2, 8, and 17 all apply the same reference rule, $h1$ as $h3$, but sequence 2 applies it to the last exploit, sequence 8 to the first and last exploits. Also, reference rules are not transitive. For example, applying two rules, $h1$ as $h2$ and $h2$ as $h3$, does not imply the rule $h1$ as $h3$ can be applied.

Table 4 shows the reference encoding for attack sequences containing local exploits. The reference rules are applied in a similar way. In contrast to using a single victim host,

Table 3: Generating Attack Sequences through Reference Encoding (Remote Exploits)

2	$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h3v3$	$h1$ as $h3$
4	$ha\$ \rightarrow hah2v1 \rightarrow h2h1v2 \rightarrow h1h3v3$	$h1$ as $h2$, $h2$ as $h1$, $h1$ as $h3$
7	$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h3v3$	$h1$ as $h3$, $h2$ as $h1$
8	$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h3v3$	$h1$ as $h3$
10	$ha\$ \rightarrow hah1v1 \rightarrow h1h3v2 \rightarrow h3h1v3$	$h2$ as $h3$
11	$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h1v3$	
15	$ha\$ \rightarrow hah2v1 \rightarrow h2h3v3 \rightarrow h3h1v3$	$h1$ as $h2$, $h2$ as $h3$
17	$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h1v3$	$h1$ as $h3$
19	$ha\$ \rightarrow hah1v1 \rightarrow h1h3v2 \rightarrow h3h2v3$	$h2$ as $h3$, $h1$ as $h2$
22	$ha\$ \rightarrow hah2v1 \rightarrow h2h1v2 \rightarrow h1h2v3$	$h1$ as $h2$, $h2$ as $h1$
24	$ha\$ \rightarrow hah2v1 \rightarrow h2h3v3 \rightarrow h3h2v3$	$h1$ as $h2$, $h2$ as $h3$
25	$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h2v3$	$h1$ as $h3$, $h2$ as $h1$, $h1$ as $h2$

the advantage of the two-node model is that the compression model can reveal the threat of a multi-step attack even if we do not consider any local exploit. For example, in Figure 9, if the vulnerabilities allow any attacker to gain user privilege on a remote host which the attacker initially has no control, then exploiting such vulnerabilities locally on a already compromised host will not elevate the attacker's privilege. Although such attacks are possible, we can safely disregard them. When local exploits need to be disregarded, the above two-node model does not need to be changed (on the other hand, if we had used a single victim host, then the model will collapse). We can make sure no sequence with local exploits is generated simply by carefully applying the reference rule.

The above compression models can be easily extended to the general case with any number of hosts or vulnerabilities, as long as all the hosts have the identical connectivity and vulnerabilities (that is, the complete case). More hosts will only need to be added to the reference rules. For vulnerabilities, we follow a simple procedure to assign them to hosts in the compression model, as detailed in Figure 10. The procedure first assigns all

Table 4: Generating Attack Sequences through Reference Encoding (Local Exploits)

1	$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h3v3$	$h2$ as $h1$, $h1$ as $h3$
3	$ha\$ \rightarrow hah1v1 \rightarrow h1h3v2 \rightarrow h3h3v3$	$h2$ as $h3$, $h1$ as $h3$
5	$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h3v3$	$h1$ as $h2$, $h1$ as $h3$
6	$ha\$ \rightarrow hah2v1 \rightarrow h2h3v3 \rightarrow h3h3v3$	$h1$ as $h2$, $h2$ as $h3$, $h1$ as $h3$
9	$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h3v3$	$h1$ as $h3$, $h2$ as $h3$
12	$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h1v3$	$h2$ as $h1$
13	$ha\$ \rightarrow hah2v1 \rightarrow h2h1v2 \rightarrow h1h1v3$	$h1$ as $h2$, $h2$ as $h1$
14	$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h1v3$	$h1$ as $h2$
16	$ha\$ \rightarrow hah3v1 \rightarrow h3h1v2 \rightarrow h1h1v3$	$h1$ as $h3$, $h2$ as $h1$
18	$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h1v3$	$h1$ as $h3$, $h2$ as $h3$
20	$ha\$ \rightarrow hah1v1 \rightarrow h1h2v2 \rightarrow h2h2v3$	$h1$ as $h2$
21	$ha\$ \rightarrow hah1v1 \rightarrow h1h1v2 \rightarrow h1h2v3$	$h2$ as $h1$, $h1$ as $h2$
23	$ha\$ \rightarrow hah2v1 \rightarrow h2h2v2 \rightarrow h2h2v3$	$h1$ as $h2$
26	$ha\$ \rightarrow hah3v1 \rightarrow h3h2v2 \rightarrow h2h2v3$	$h1$ as $h3$, $h1$ as $h2$
27	$ha\$ \rightarrow hah3v1 \rightarrow h3h3v2 \rightarrow h3h2v3$	$h1$ as $h3$, $h2$ as $h3$, $h1$ as $h2$

the vulnerabilities to each host, as in the original configuration graph. It then generates the compressed attack graph using the type graph and this temporary vulnerability assignment. This step is to ensure that all attack sequences that may appear in the compressed attack graph should remain so in the final compression model. Next, the procedure removes those assignments of vulnerabilities that do not appear in the compressed attack graph to avoid redundancy. If a reference host does not have any vulnerability, then we also remove that reference host from the compression model to avoid redundancy.

With the above procedure for assigning hosts, we can now show that the one-node compression model we described at the beginning of this section is actually a special case of the two-node model. As shown in the left-hand side of Figure 11, the configuration graph in Figure 6 can be represented by the two-node compression model with no connectivity between the two victim hosts. Once the above algorithm generates the compressed attack

Input: A configuration graph CG, a type graph TG, and a compressed configuration graph CCG with no vulnerability assigned to any host
Output: An updated CCG with vulnerabilities assigned using a function $f() : H \rightarrow V$ where H and V are the set of hosts and vulnerabilities in CCG, respectively
Method:

```
1.  For each $h \in H$
2.      Let $f(h) = V$
3.  Generate the compressed attack graph $CAG$ from $TG$ and $CCG$
4.  For each $h \in H$
5.      For each $v \in V$
6.          If $\langle h, v \rangle \notin CAG$
7.              Let $f(h) = f(h) - \{v\}$
8.      If $f(h) = \phi$
9.          Let $H = H - \{h\}$
10. Return $CCG$
```

Figure 10: An Algorithm for Assigning Vulnerabilities in the Complete Case

graph, it is clear that $h2$ will be assigned no vulnerability at all, and thus $h2$ itself will be removed from the compression model. That is, the two-node model reduces to the one-node model.

We formally describe the two-node compression model for the complete case in Definition 4.

Definition 4 *Given a vulnerability-based type graph $TG(N_t, E_t)$ and a configuration graph $CG(N_c, E_c)$ in the complete case and let H be the set of hosts appearing in TG, we define*

- *the compressed configuration graph as a directed graph $CCG(N_{cc}, E_{cc})$. The node set is $N_{cc} = \{ha(\$), hi(V_1), hj(V_2)\}$ where $ha, hi, hj \in H$, V_1 and V_2 are vulnerabilities assigned using the algorithm shown in Figure 10, and the edge set $E_{cc} = \{(ha, hi), (hi, ha), (hi, hj), (hj, hi)\}$,*

- *the reference rule $RR = \{hi \text{ as } hx \mid hx \in H \wedge x \neq i\} \cup \{hj \text{ as } hy \mid hy \in H \wedge y \neq j\}$,*

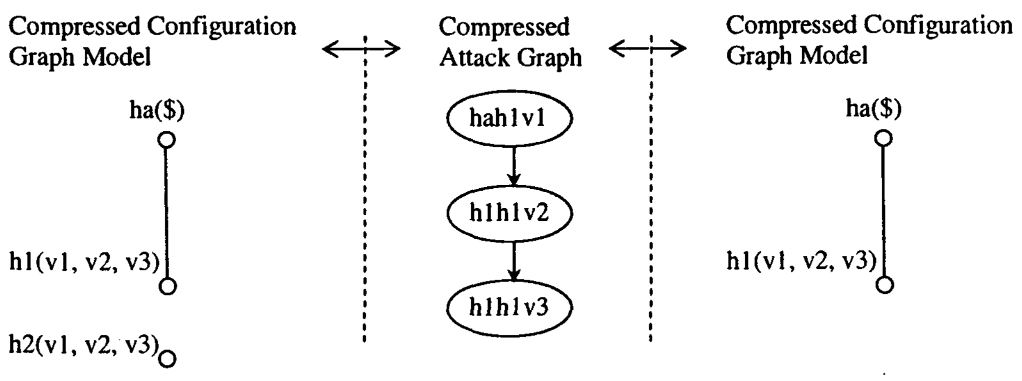

Figure 11: A Special Case of the Two-Node Compression Model

- *the compression model as the pair* $\langle CCG, RR \rangle$.

In the complete case, the above procedure will basically assign vulnerabilities to one of the two reference hosts in an alternating fashion (in later sections we shall see exceptions to this observation). For example, in Figure 9, if there are three most vulnerabilities $v4$, $v5$, and $v6$, then $v4$ and $v6$ will be assigned to $h2$ while $v5$ to $h1$. In general, a breadth-first search (BFS) in the type graph can be used to alternatively assign each encountered vulnerability to one of those two hosts based on its distance from the first vulnerability.

3.2.3 Implication of Type Graphs

We have so far only considered type graph as a total order. In reality a type graph will certainly include disjunctive and conjunctive relationships between vulnerabilities, as discussed in Section 3.1. Such relationships may introduce additional constraints for assigning vulnerabilities to hosts in our compression model. The BFS used by the compression model may not satisfy such constraints, and some possible attack sequences may thus be missing in the compressed attack graph. For example, if the type graph in Figure 9 requires $v1$ and $v2$ to be exploited on the same host, then the compressed attack graph would have been empty since the attack sequence shown in the right-hand side of Figure 9 is no longer possible. However, notice that the compression model is actually still correct since reference encoding will allow $h2$ to be replaced by $h1$ and thus enable the generation of that attack sequence.

Nonetheless, the above situation is not satisfactory since our goal is that the compressed attack graph can reveal the same threat as the original attack graph does, whereas here an empty attack graph certainly does not achieve this goal. A security administrator would have to apply the reference rule before he/she can identify any threat. We thus need to revise the compression model as follows. First of all, disjunctive relationships between vulnerabilities does not cause any complication since they do not introduce any constraints to assigning vulnerabilities to hosts. However, the complication does arise when type graph includes conjunction between vulnerabilities. Before we discuss the complication, recall in Section 3.1 we mentioned that in a vulnerability-based type graph the disjunctive conjunctive relationship between vulnerabilities must be explicitly represented due to the lack of conditions. We shall use a small circle to represent the conjunctive relationships between vulnerabilities, as depicted in Figure 12. For disjunctive relationships, we simply omit the small circle and directly connect interdependent vulnerabilities.

The left-hand side of Figure 12 indicates that both $v1$ and $v2$ must be exploited before $v3$ can be exploited, while all three vulnerabilities reside on different hosts. This case has no impact on the compression model, since there is no special requirement for assigning

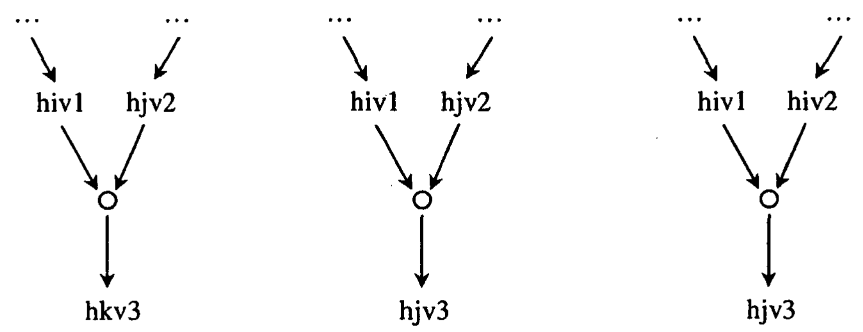

Figure 12: Different Cases of Conjunctive Relationships in Type Graphs

those three vulnerabilities to hosts. The three vulnerabilities may reside on the same or different hosts in the compression model, which makes no difference to the generation of attack sequences. The middle case indicates that both $v1$ and $v2$ must be exploited before $v3$ can be exploited on the same host as $v2$ is exploited on. This requirement can be satisfied by simply assigning $v2$ and $v3$ to the same host in the compression model although they clearly have different heights in the BFS tree.

The right-hand side of Figure 12 indicates that both $v1$ and $v2$ must be exploited on the same host (denoted by hi) before $v3$ can be exploited (on the same host or a different host). An complication arises here since $hiv1$ and $hiv2$ may have different height in the BFS tree, $v1$ and $v2$ may be assigned to different hosts in the compression model. This will result in the loss of attack sequences which should be present. Therefore, when the BFS first reaches one of the two vulnerabilities, it should assign both vulnerabilities to the same host (one of the vulnerabilities may thus be assigned to more than one host, which is fine).

A similar constraint may also be introduced by cycles in the type graph. By the monotonic assumption [3], an attacker would never need to exploit the same vulnerability more than once. Therefore, any cycle in a type graph can usually be safely ignored. However, in cases where such cycles are important, a cycle with an odd number of vulnerabilities will cause a complication to the compression model. For example, suppose in Figure 9 we add an edge pointing from $v3$ back to $v1$, and also suppose we do not consider local exploits. Clearly, now the edge from $v3$ to $v1$ will not lead to an exploit in the compressed attack graph because both $v3$ and $v1$ are on the same host $h1$ (while we choose not to consider local exploits). The solution is to add $v1$ to $h2$ such that the exploit $h1h2v1$ can be added to the end of the current compressed graph to reflect the cyclic dependency given in the type graph.

3.3 The Incomplete Case

The compression model discussed in the previous section only works in the complete case. For incomplete cases where hosts may have different vulnerabilities and connectivity, the compression model will introduce attack sequences that do no exist in the original attack graph. Such non-existent attack sequences should certainly be removed from the compression model. However, we need to distinguish two cases. First, if the difference between hosts in terms of connectivity and vulnerabilities is relatively small, then a conservative assumption can be made that such difference does not significantly reduce the threat of multi-step attacks. Therefore, we take a *negation approach* by regarding those hosts as in the complete case and apply the previous compression model; in addition, we indicate the missing connectivity or vulnerabilities using special notations in the reference rules. Second, when the difference between hosts is significant, we simply leave those hosts outside the compression model.

In many cases, organizational boundaries or external knowledge about the network topology will provide a natural way for classifying subnets into the above two cases. For example, in a university network, each department or faculty usually has its own subnet inside which hosts are fully connected. Each lab or office usually is equipped with machines with similar or same configuration. On the other hand, the connectivity between different academic units and that with the outside world is usually limited by firewalls. An alternative approach when such information is not readily available is to apply clustering methods to the configuration and connectivity data such that clusters of machines with similar configuration and connectivity will be identified. The discussion of such clustering methods is out of the scope of this thesis.

3.3.1 Differences through Reference Rules

Figure 13 shows an example of the incomplete case where hosts have small differences in terms of connectivity and vulnerabilities. The left-hand side shows a configuration graph with three victim hosts whose connectivity and vulnerabilities are slightly different. More specifically, $h1$ and $h3$ both have three vulnerabilities $v1$, $v2$, and $v3$, while $h2$ does not have $v2$; the three hosts are fully connected to each other and to the attacker's host ha, except that there is no connection from $h3$ to $h1$ (for example, due to a personal firewall software running on $h1$). The right-hand side of the figure shows the same vulnerability-based type graph as before.

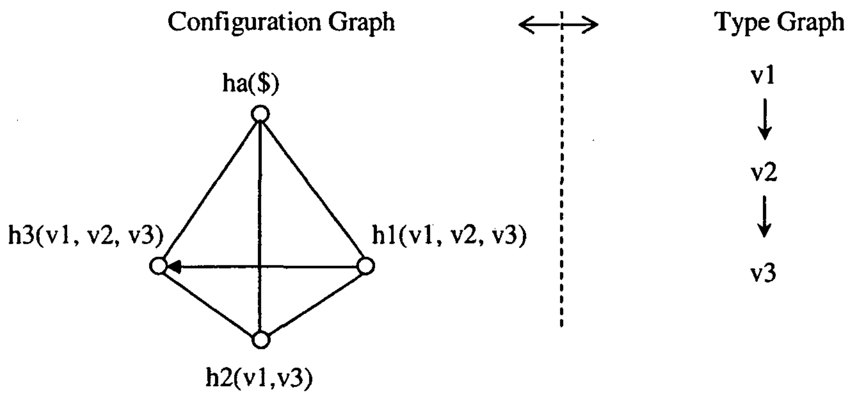

Figure 13: An Example of the Incomplete Case

Clearly, if we simply apply the compression model described in the previous section here, the resulted compressed configuration graph will allow non-existent attack sequences to be generated. The negation approach we take is to start with such an incorrect configuration graph, and fix it through the reference rules. That is, we represent non-existent connectivity and vulnerabilities in the reference rules. The modified compression model is shown in the left-hand side of Figure 14. The compressed configuration graph is identical to the one in Figure 9, because we assume all three hosts have the same connectivity and vulnerabilities as $h1$ does. The reference rules now include pairs of parentheses inside which the non-existent vulnerabilities or connectivity are shown.

The above compression model can satisfy both of the aforementioned requirements. First, the model reveals the same threat from potential multi-step attacks. The right-hand side of Figure 14 shows exactly the same compressed attack graph as before. A security

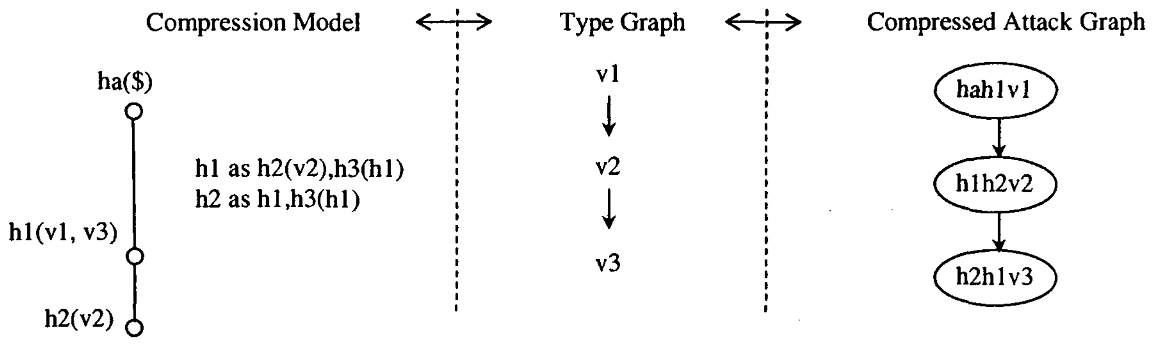

Figure 14: The Compression Model in the Incomplete Case

administrator will be able to tell from the compressed attack graph alone that attackers may exploit the three vulnerabilities in the given order across different hosts in the network. Second, because the reference rules now exactly capture the actual network configuration, the compression model will allow exactly the same set of attack sequences to be generated, as shown in Table 5 (the crosslines indicate sequences with local exploits).

Table 5: Generating Attack Sequences through Reference Encoding

1	~~$ha\$ \to hah1v1 \to h1h1v2 \to h1h1v3$~~	$h2$ as $h1$
2	~~$ha\$ \to hah1v1 \to h1h1v2 \to h1h3v3$~~	$h2$ as $h1$, $h1$ as $h3$
3	~~$ha\$ \to hah1v1 \to h1h1v2 \to h1h2v3$~~	$h2$ as $h1$, $h1$ as $h2$
4	$ha\$ \to hah1v1 \to h1h3v2 \to h3h2v3$	$h2$ as $h3$, $h1$ as $h2$
5	~~$ha\$ \to hah1v1 \to h1h3v2 \to h3h3v3$~~	$h2$ as $h3$, $h1$ as $h3$
6	~~$ha\$ \to hah3v1 \to h3h3v2 \to h3h3v3$~~	$h1$ as $h3$, $h2$ as $h3$

One subtlety is that the attack sequence represented by the compressed attack graph now does not correspond to the original network configuration (while it does in Figure 9). Indeed, we can observe that only $h1$ is *complete* in terms of connectivity and vulnerabilities, while $h2$ and $h3$ are both incomplete in this sense. Therefore, we cannot find any two hosts to exactly represent the compressed configuration graph in the complete case. However, we notice that the negation approach represents a conservative approach in which the worst case represented by the compressed attack graph is the most useful information to a security administrator. The attack non-existent sequences are secondary details that can be delayed until examining the reference rules. It is sufficient to advise users of the compression model that the compressed attack graph only reveals the potential attacking strategy. Actually attack sequences must be generated using the reference rules, if such generation is necessary.

Another issue lies in the choice of $h1$ as the *complete* host. We can easily imagine a case where none of the hosts is complete in terms of connectivity and vulnerabilities. That is, there does not exist any host whose connectivity and set of vulnerabilities are both the superset of those of other hosts. For example, suppose in Figure 13 we remove $v3$ from $h1$, then apparently no host is complete now. In such a case, the compression model must be based on *virtual* hosts which have the complete connectivity and set of vulnerabilities from the union of such sets of all the hosts. Again, the compressed attack graph generated from such virtual hosts may not correspond to any actual attack sequence in the original attack graph, but it reflects the worst case attack scenario, which we believe is the most pertinent information to a security administrator.

We are now ready to formally define the compression model in this incomplete case in Definition 5.

Definition 5 *Given a vulnerability-based type graph $TG(N_t, E_t)$ and a configuration graph $CG(N_c, E_c)$ in the complete case and let H be the set of hosts appearing in CG, we define*

- *the compressed configuration graph as a directed graph $CCG(N_{cc}, E_{cc})$. The node set is $N_{cc} = \{ha(\$), hi(V_1), hj(V_2)\}$ where $ha, hi, hj \in H$, V_1 and V_2 are vulnerabilities assigned using the algorithm shown in Figure 10, and the edge set $E_{cc} = \{(ha, hi), (hi, ha), (hi, hj), (hj, hi)\}$,*
- *the reference rule $RR = \{hi \textit{ as } hx(H_x, V_x) \mid hx \in H \wedge x \neq i\} \cup \{hj \textit{ as } hy(H_y, V_y) \mid hy \in H \wedge y \neq j\}$ where H_x and V_x represents the set of hosts that have no connectivity from hx, and the set of vulnerabilities absent in hx, respectively (similar for H_y and V_y),*
- *the compression model as the pair $\langle CCG, RR \rangle$.*

3.3.2 Hybrid Case

We now consider the other situation in the incomplete case, that is, when compression is applied to some hosts but not needed for the others, namely, a hybrid case. Figure 15 shows such an example. All the victim hosts have the same set of vulnerabilities, but they may differ in terms of connectivity. The two groups of hosts, $h1$, $h2$, $h3$ and $h4$, $h5$, $h6$, respectively correspond to the complete case. However, the two groups are only connected through $h7$ while there is no direct connection between other hosts. Clearly, the difference between the two groups in terms of connectivity is significant enough such that they should not be put into the same two-node compression model. Instead, two separate compression models should be created and then put together through hosts $h7$, which are not part of the compression models.

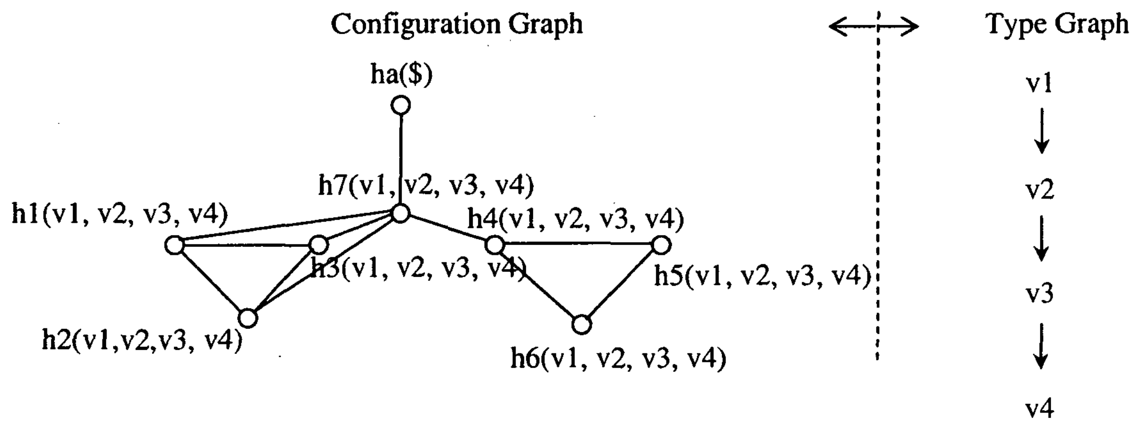

Figure 15: An Example of the Hybrid Case

The way of creating compression models for the two groups of hosts is slightly different. Since $h7$ is connected to all the three hosts in the first group, that is $h1$, $h2$, and $h3$, we can simply regard $h7$ as the attacker's host ha in previous cases. Therefore, we simply create a two-node compression model for those three hosts as usual. On the other hand, in the second group, only $h4$ is connected to the external node $h7$ while the other two are not. If we create a compression model for those three nodes, then the non-existent connectivity between $h5$, $h6$ and $h7$ will have to be explicitly represented in the reference rules. If the second group includes more hosts like $h5$ and $h6$ (that is, they are not connected to $h7$), then more redundancy will be introduced, and the fact that they are not connected to $h7$ would not be immediately available in the compressed attack graph. A better solution is not to include $h4$ in the compression model, but instead treat it the same as $h7$, that is, as an external host.

The compression model for this case is shown in the upper left corner of Figure 16.

The compression model includes the two-node model of each group together with external hosts. Notice the way of assigning vulnerabilities inside each two-node model is different from previous cases. This is due to the fact that the some external hosts, such as $h7$ and $h4$, are now in between the attacker's host ha and the two-node model. The algorithm shown in Figure 17 will lead to the current assignment of vulnerabilities, which is minimal in the sense that each appearance of a vulnerability is necessary for some attack sequences in the compressed attack graph.

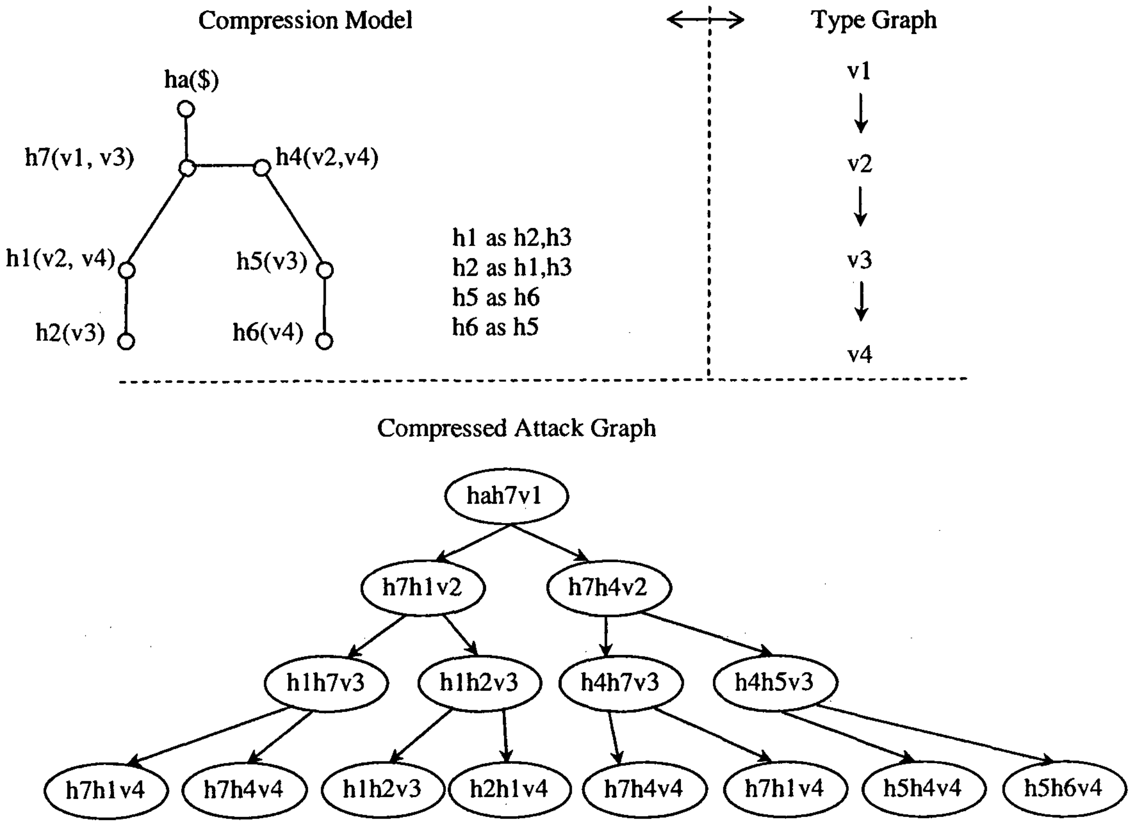

Figure 16: The Compression Model in the Hybrid Case

The compression model can satisfy our requirements. Clearly, the model can reveal the threat of potential multi-step attacks. At first glance, the compressed attack graph in Figure 16 seems to still have many repetitive attack sequences. However, those are necessary because they show different possible paths the attacker may follow, such as attacking one host inside a group then either continuing to other hosts inside that group, or moving back to the external host, or even moving between the two groups. What is being omitted in the compression model are the possible moves inside each group, such as those involving $h3$, which have little additional meaning and are thus hidden in the reference rules. The compressed attack graph shows to security administrators a complete and yet concise picture of what may happen to the network. Second, once the reference rules are applied, the compression model allows generating exactly the same set of attack sequences as in the original attack graph, although the result will include a large number of attack sequences and hence is omitted.

The above compression models can be easily extended to the general case with any number of hosts or vulnerabilities. More groups will only lead to more two-node models. For vulnerabilities, we follow a similar procedure to assign them to hosts in the compression model, as detailed in Figure 17. The main difference from the previous procedure for the complete case is that instead of generating the compressed attack graph for each group of hosts, the procedure generates a compressed attack graph for whole network using the type graph and temporary vulnerability assignment of each group. This step is to ensure that all attack sequences that may appear in the overall compressed attack graph should remain so in the final compression models.

Input: A configuration graph CG, a type graph TG, and compressed configuration graphs $CCG_i(i = 1, 2, \ldots, k)$ with no vulnerability assignment
Output: Updated $CCG_i(i = 1, 2, \ldots, k)$ with vulnerabilities assigned using $f_i() : H_i \rightarrow V_i$ where H_i and V_i are the set of hosts and vulnerabilities in CCG_i, respectively
Method:

```
1.  For $i = 1$ to $k$
2.      For each $h \in H_i$
3.          Let $f_i(h) = V_i$
4.  Generate the compressed attack graph $CAG$ from $TG$ and all the $CCG_i$'s
5.  For $i = 1$ to $k$
6.      For each $h \in H_i$
7.          For each $v \in V_i$
8.              If $\langle h, v \rangle \notin CAG$
9.                  Let $f_i(h) = f_i(h) - \{v\}$
10.         If $f_i(h) = \phi$
11.             Let $H = H - \{h\}$
12. Return $CCG_i(i = 1, 2, \ldots, k)$
```

Figure 17: An Algorithm for Assigning Vulnerabilities in the General Case

Chapter 4

Case Study and Experiments

In this chapter, we first give a case study to demonstrate the application of those methods in Section 4.1. We then show some experimental results on the performance of our methods in Section 4.2.

4.1 A Case Study of Analysis with Compression Model

To demonstrate how our compression model works during an analysis of attack graphs, we study a simple network illustrated in Figure 18. The network includes four densely connected subnets, $Li(i = 1, 2, \ldots, 5)$, $Ri(i = 1, 2, \ldots, 8)$, $Si(i = 1, 2, \ldots, 9)$, and $Di(i = 1, 2, \ldots, 6)$, which are interconnected with four routers $P1$ through $P4$. All hosts inside each subnet have the same set of vulnerabilities unless it is explicitly indicated otherwise in the figure. The host $R5$ has a wireless connection to the Internet. The dash line between $S9$ and $S8$ indicates there is no connection between them due to personal firewalls. Suppose the two hosts in red color, $S9$ and D_6, are compromised, and we suspect the intrusion origins from one of the mobile hosts Li's that is connected to the network through a wireless access point.

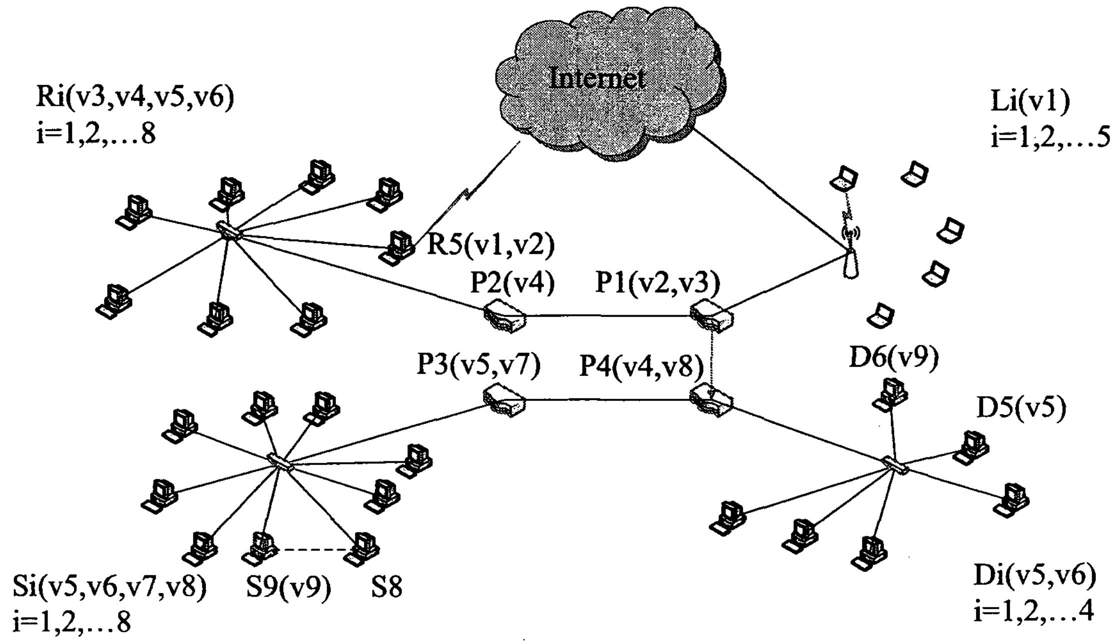

Figure 18: An Example Network

Assuming a simple type graph that forms a total order $v1 \rightarrow v2 \rightarrow \ldots v9$. Clearly, the traditional approach to attack graph generation would lead to a very large attack graph in this case (indeed, it includes around 500 attack sequences). It is thus desirable to apply our compression model here. The four subnets naturally map to four separate two-node compression models, while the four routers are left outside the compression models. In particular, hosts L_i's will reduce to a one-node model since in the infrastructure mode all the hosts communicate with the access point but there is no direction communication between those hosts.

For hosts R_i's, a two-node model suffices, except that host $R5$ will need to be left out of the compression model since it has a direct connection to the internet which are absent on other hosts in the subnet. The other two subnets can also be represented by two-node models, with the small difference in connectivity and vulnerabilities modeled in reference rules. The only exceptions are host $S9$ and $D6$, which are preferable left out of the compression model due to their special role as victim hosts, and also due to the unique vulnerability $v9$ that only they have. Figure 19 shows the complete compression model.

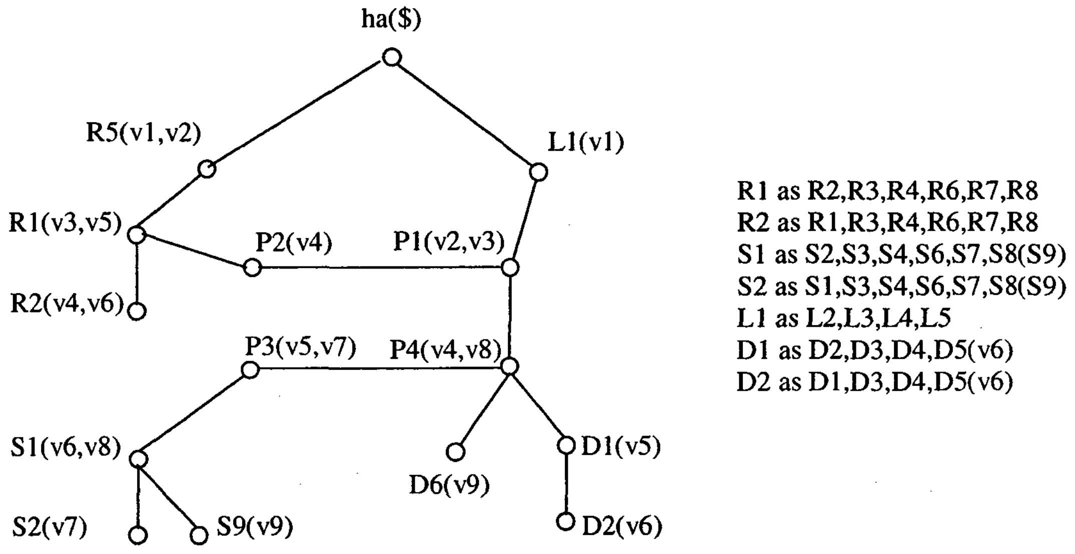

Figure 19: The Compression Model

Figure 20 shows the compressed attack graph that can be generated using the above compression model. This compressed attack graph clearly shows the possible paths that may be taken by attackers in compromising the two victim hosts $S9$ and $D6$. For example, he/she may start from the mobile host $L1$ and move to $P1$, $P4$, $P3$, and $S1$ where he/she can either continue to compromise $S9$, or move back to $P3$, $P4$, and finally compromise $D6$. Some other attack sequences also exist, although they do not directly lead to the compromise of the two hosts in question.

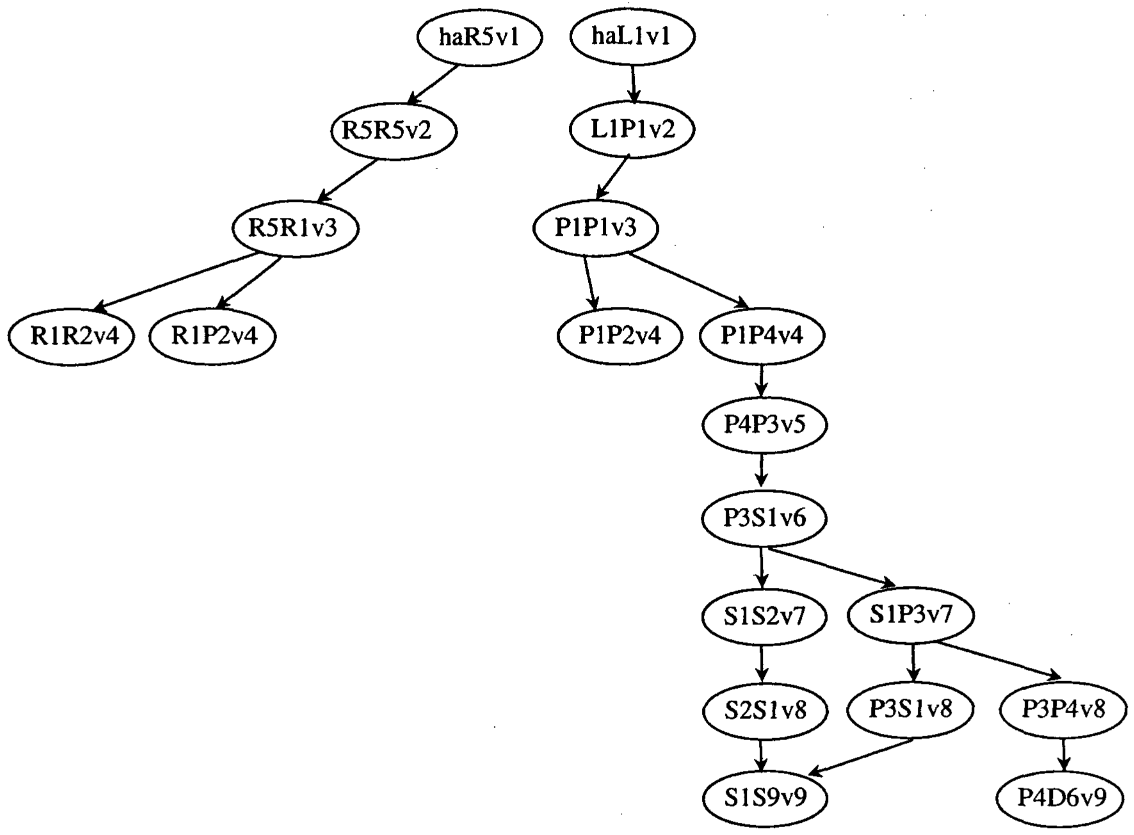

Figure 20: The Compressed Attack Graph

Once we obtain the analysis result, the two attack sequences in Table 6, we can easily generate the complete analysis result by applying the reference rules. We shall not show the result since it would include a large number of attack sequences. However, the result in Table 6 clearly includes enough information for security administrators to take actions such as network hardening. Moreover, we never need to first generate the complete attack graph, which would be prohibitive, in order to obtain such information. In summary, our compression model can enable useful analysis with less severe scalability issues.

Table 6: Attack Sequences Based on the Compression Model

1	$ha\$ \rightarrow haL1v1 \rightarrow L1P1v2 \rightarrow P1P1v3 \rightarrow P1P4v4 \rightarrow P4P3v5 \rightarrow P3S1v6 \rightarrow S1S2v7 \rightarrow S2S1v8 \rightarrow S1S9v9$
1	$ha\$ \rightarrow haL1v1 \rightarrow L1P1v2 \rightarrow P1P1v3 \rightarrow P1P4v4 \rightarrow P4P3v5 \rightarrow P3S1v6 \rightarrow S1P3v7 \rightarrow P3P4v8 \rightarrow P4D6v9$

4.2 Implementation and Experiments

We have implemented the compression model in Java. Our experiments also employ a synthetic topology generator, the Boston university Representative Internet Topology gEnerator (BRITE) [7], and a general-purpose network visualization tool OTTER [47] by the Cooperative Association for Internet Data Analysis (CAIDA). We tested the performance of our proposed compression model on machines equipped with Intel Pentium M 1.80GHz processor, 1024MB RAM, and Windows XP operating system.

Although there exist many topology generators, we choose BRITE due to its following advantages [7].

1. First, BRITE is integrated with CAIDA's visualization tool, Otter, allowing easy visualization of generated topologies.

2. Second, BRITE exports to simulation software ns, SSFNET, JavaSim, OmNet++, etc., which allows potential future work based on current results.

3. Third, BRITE is implemented in Java and C++, which can be easily understood and integrated with our codes.

4. Finally, BRITE provides a GUI and a configuration file for users to easily specify different topology generation parameters.

The Parameters and values, which are described in the table 7 [7], are instantiated as the following in our experiments.

```
public void set_config(int nodeAmount) {
```

```
strConf[0]="BriteConfig";

strConf[1]="BeginModel";

strConf[2]="Name = 3 #Router Waxman = 1,    ASWaxman = 3";

strConf[3]="N = "+nodeAmount+" #Number of nodes ingraph";

strConf[4]="HS = 100 #Size of main plane (number ofsquares)";

strConf[5]=" LS = 100 #Size of inner planes (number ofsquares)";

strConf[6]=" NodePlacement = 1 #Random = 1, Heavy Tailed= 2";

strConf[7]=" GrowthType = 1 #Incremental = 1, All = 2";

strConf[8]=" m = 2 #Number of neighboring node each new

                   nodeconnects to";

strConf[9]=" alpha = 0.15 #Waxman Parameter";

strConf[10]=" beta = 0.2 #Waxman Parameter";

strConf[11]="BWDist = 1 #Constant = 1, Uniform =2,

                       HeavyTailed = 3,Exp =4";

strConf[12]="BWMin = 10.0"; strConf[13]=" BWMax = 1024.0";

strConf[14]="EndModel"; strConf[15]="BeginOutput";

strConf[16]="BRITE = 1 #1/0=enable/disable output in BRITE format";

strConf[17]="OTTER = 1 #1/0=enable/disable visualization in otter";

strConf[18]="DML = 0 #1/0=enable/disable output to SSFNet's

                                                DML format ";

strConf[19]=" NS = 0 #1/0=enable/disable output to NS-2";

strConf[20]=" Javasim = 0 #1/0=enable/disable output to Javasim";

strConf[21]="EndOutput";
```

Table 7: BRITE's Parameters [7]

Parameter	Meaning	Values
HS	Size of one side of the plane	int
LS	Size of one side of a high-level square	int
N	Number of nodes	int
Model	model id	int
alpha	Waxman-specific	exponent
beta	Waxman-specific	exponent
Node	Placement how nodes are placed in the plane	1: Random, 2: HT
m	Number of links per new node	int
Growth Type	how nodes join the topology	1: Incremental,2:Random
BWdist	bandwidth assignment to links	1:Const, 2: Unif, 3: Exp,4:HT
MaxBW, MinBW	min, max link bandwidth values	float

}

The way BRITE generates a topology is described as the follows [7].

1. First, place the nodes in the plane.

2. Second, interconnect the nodes.

3. Third, assign attributes to topological components, such as delay and bandwidth for links, AS id for nodes, etc.

4. Finally, output the topology to a specific format.

We use the command line interface of BRITE to invoke the BRITE generation engine, since this can be implemented in a script for repetitive experiment runs. The command line receives as arguments a configuration file, a location for an export file, and a seed file. BRITE uses pseudo-random numbers at various places of the generation process, such as nodes placed randomly in the plane, nodes interconnected according to certain probability function, etc. The seed_file parameter contains seeds to initialize the pseudo-random number generator. More precisely, the command line used in the experiments for starting BRITE is the following.

```
$ Java main.brite my_config.conf my_export.odf seed-file.
```

BRITE's output file includes the information about the topology contained in the file, such as number of nodes and edges, and the node and edge information. For each node and edge in the graph, a line in a specific format is included in the output file. The following is an example of the output format.

```
    g 0 d 1 Node Values
    f 0 Degree
    g 1 d 1 Node Classification
    f 1 Corresponding AS
    g 2 d 2 Edge Values
    f 2 Bandwidth'Distance
    t 3
    T 3
    N 0 19 17 0
    v 0 0 3
    v 0 1 -1
    N 1 18 38 1
```

```
v 1 0 4

v 1 1 -1

N 2 31 4 2

v 2 0 4

v 2 1 -1

L 0 1 2

V 0 2 10.0'15.524174696260024

L 1 2 1

V 1 2 10.0'47.20169488482379

L 2 0 1

V 2 2 10.0'19.4164878389476
```

We inject additional information about node names and vulnerabilities into the output file of BRITE such that the result can be used as configuration graph of our compression model. Since the random topology does not include organizational information for forming the natural clusters of hosts, we use the k-means algorithm [32] to divide all nodes into clusters. Vulnerabilities are randomly assigned to all hosts. The following is an example of the processed file after necessary information is injected.

```
g 0 d 1 Node Values

f 0 Degree

g 1 d 1 Node Classification

f 1 Corresponding AS

g 2 d 2 Edge Values
```

```
f 2 Bandwidth'Distance

t 256

T 512

N 0 60 31 H(0,0):(V1,V5,V8,V11,V18)

v 0 0 5

v 0 1 -1

N 1 82 18 H(0,67):(V11,V18,V19)

v 1 0 5

v 1 1 -1

N 2 40 97 H(1,128):(V4,V5,V10)

v 2 0 2

v 2 1 -1

L 271 119 16

V 271 2 10.0'54.405882034941776

L 272 63 212

V 272 2 10.0'73.348483283569

L 273 63 99

V 273 2 10.0'63.12685640834652

L 274 226 95

V 274 2 10.0'15.264337522473747

L 275 226 150

V 275 2 10.0'35.34119409414458
```

To examine the generated topology, we use OTTER [47] to visualize the result. Figure 21 shows a screenshot of the displayed topology.

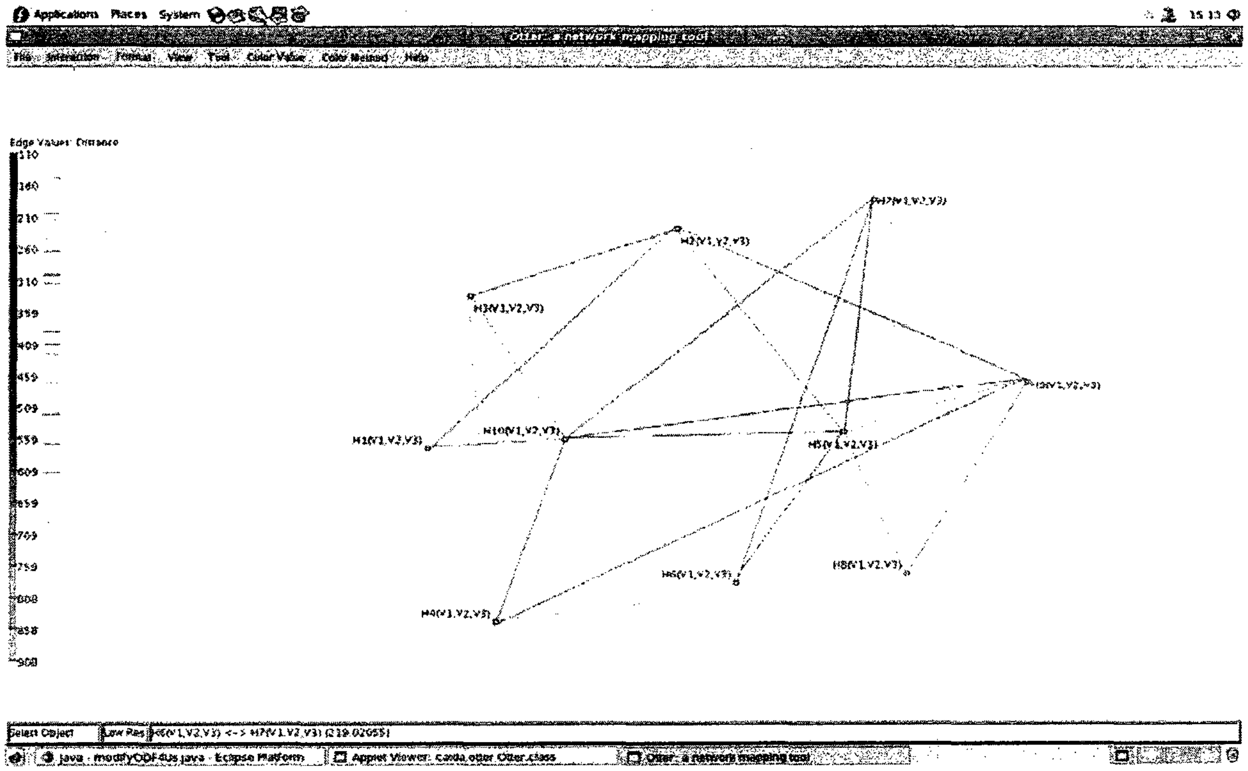

Figure 21: A Topology Shown in OTTER

Finally, we generate attack sequences using both the traditional method and our compression model. In the following we compare the two results to demonstrate the performance of our model. We generated around 2800 different topologies using the aforementioned tools. We vary the number of hosts from 60 to 300 and the number of sub-nets from 2 to 15.

Figure 22 shows the size of the full attack graph and the compressed attack graph, respectively, in the number of hosts. Clearly, the size of the full attack graph increases almost exponentially, with close to 10000 attack sequences for a network of 300 hosts. Such a result is certainly incomprehensible to human eyes. On the other hand, the size of the compressed attack graph increases significantly slower, and the result may still be meaningful to a security expert for networks with around 300 hosts.

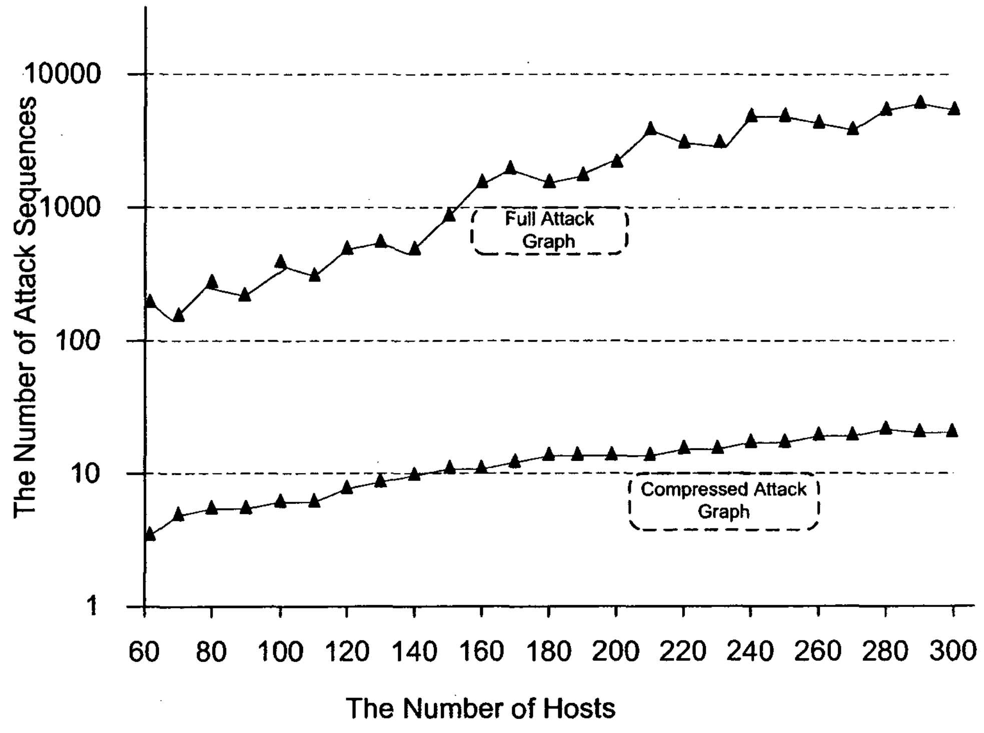

Figure 22: Attack Graph Sizes in the Number of Hosts

Figure 23 compares the number of hosts in the original attack graph and the compressed version. We can see that for a 300-node network, the compressed attack graph only includes around 30 hosts. This ten-fold difference clearly demonstrates the effectiveness of our approach. On the other hand, it is worth noting that the aforementioned reduction in the number of attack sequences is mainly due to a smaller number of hosts in the compressed attack graph, as evidenced here. This is because the compressed attack graph is similar to a full attack graph, so the number of attack sequences will still grow quickly in the number of hosts.

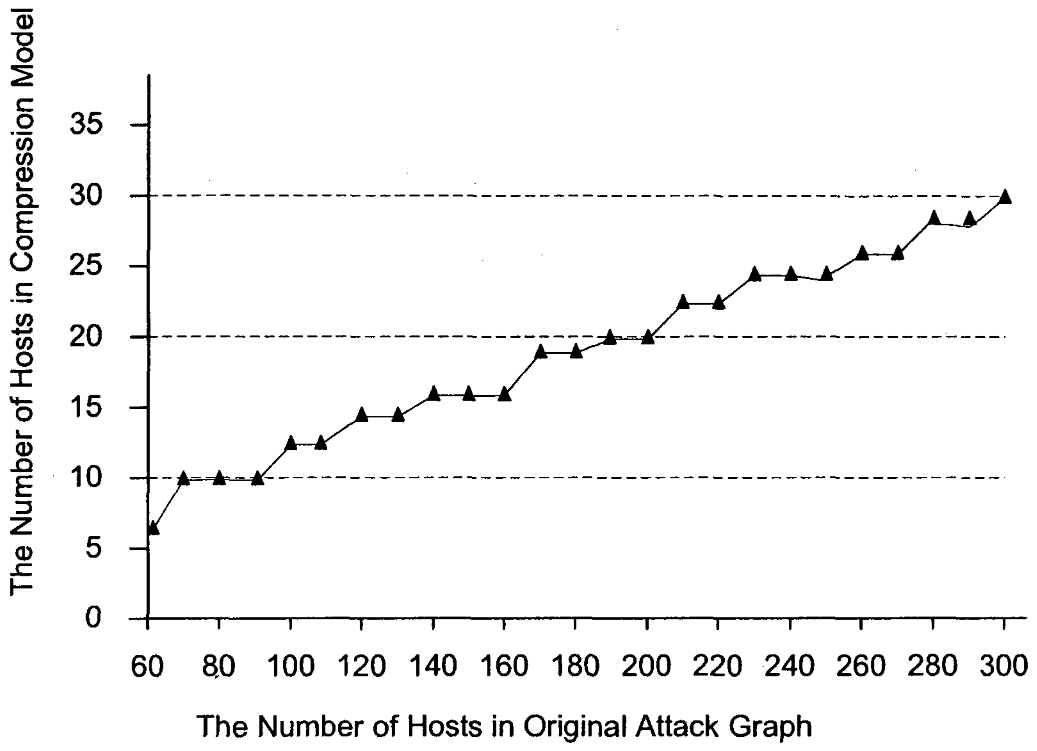

Figure 23: Comparison of the Number of Hosts

Figure 24 shows how the number of hosts in the full attack graph and compressed attack graph, respectively, grow in the number of clusters. Again, the size of the compression model increases much slower than the full attack graph due to the removal of redundant hosts in the former. On the other hand, for the compression model, the number of clusters is the main factor that causes the number of hosts to increase, as we have demonstrated in previous sections.

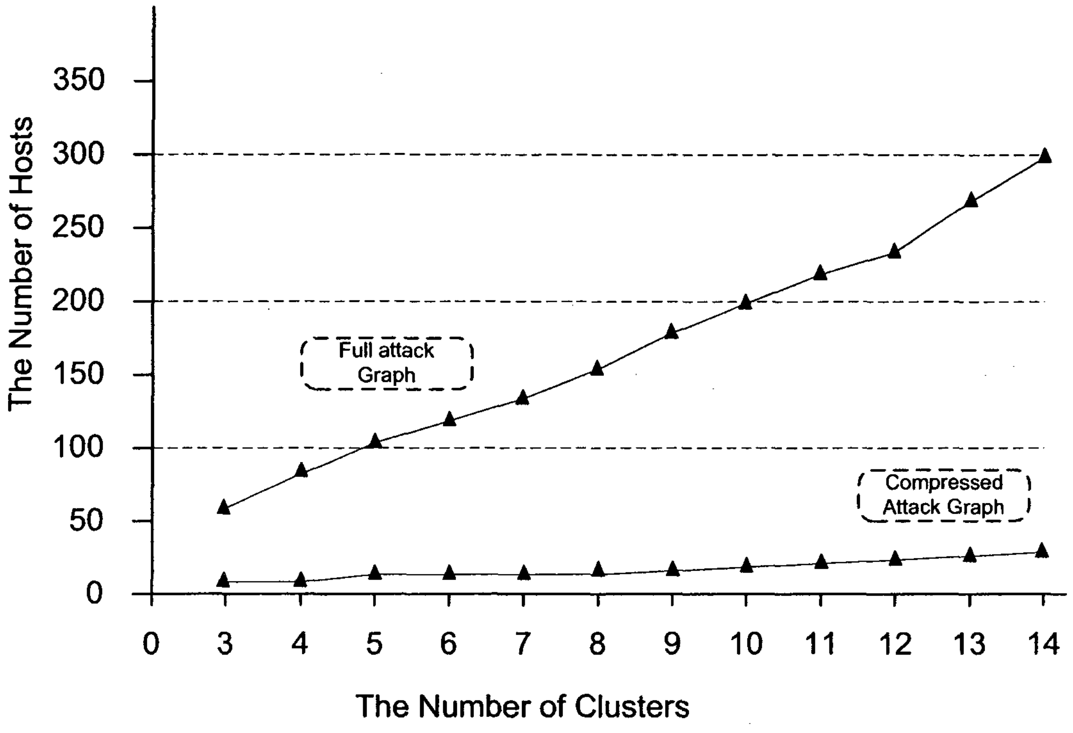

Figure 24: The Number of Hosts in the Number of Clusters

Chapter 5

Conclusion

This thesis tackled the scalability issue of attack graphs. As a useful tool for analyzing multi-step attacks, attack graph suffers from poor scalability. Even for reasonably large networks, the attack graphs are typically incomprehensible to human eyes. For large networks, even the generation of attack graphs would be prohibitive. Existing solutions can at best hide the scalability issues through hierarchical visualization of attack graphs, which does not really remove redundant information from those graphs.

In this thesis, we have adapted the well known compression technique, reference encoding, to the context of attack graphs. We proposed a compression model for attack graphs in several steps. First, we studied the complete case where hosts with identical connectivity and vulnerabilities can be represented by a one-host model. Second, we showed that such a one-host model has limitations in representing remote exploits across different machines. We then introduced a two-node model to address this limitation. We also showed that the one-host model is actually a special case of the two-node model.

Third, we studied the incomplete case where the connectivity and vulnerabilities may vary among hosts. We analyzed two cases, one with small variations which can be handled by reference rules and the other with significant differences that are better handled by leaving hosts outside the compression model. To evaluate the proposed compression model, we have conducted a case study of a small network compromised through mobile nodes that are connected to the main network through wireless links. The result shows that our compression model provides a clear picture about the multi-step attacks.

We have also implemented and tested the compression model on the basis of synthetic network topologies generated by existing tools. The results also confirm that our model can significantly reduce the size of attack graphs. Our future work includes the experimental study of the performance of our methods under real world network topologies and the integration of the compression model into existing attack graph generation engines.

Bibliography

[1] Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *DCC '01: Proceedings of the Data Compression Conference (DCC '01)*, page 203, Washington, DC, USA, 2001. IEEE Computer Society.

[2] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical report, Carnegie Mellon University, 2000.

[3] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, 2002.

[4] Applied Computer Security Associates. Workshop on. In *Information Security System Scoring and Ranking*, 2001.

[5] D. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 1st Workshop on Quality of Protection*, 2005.

[6] P. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 2nd ACM workshop on Quality of protection*, 2005.

[7] Boston university representative internet topology generator. Available at http://www.cs.bu.edu/brite/.

[8] CERT Coordination Center. CERT/CC overview incident and vulnerability trends. http:// www.cert.org/ present/ cert-overview-trends/, 2003.

[9] R. Chinchani, A. Iyer, H. Ngo, and S. Upadhyay. Towards a theory of insider threat assessment. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN'05)*, 2005.

[10] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'01)*, 2001.

[11] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02)*, pages 187–200, 2002.

[12] F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Proceedings of the 3rd International Symposium on Recent Advances in Intrusion Detection (RAID'01)*, pages 197–216, 2001.

[13] M. Dacier. Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.

[14] M. Dacier, Y. Deswarte, and M. Kaaniche. Quantitative assessment of operational security: Models and tools. Technical Report 96493, 1996.

[15] O. Dain and R.K. Cunningham. Building scenarios from a heterogeneous alert system. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, 2001.

[16] O. Dain and R.K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the ACM Workshop on Data Mining for Security Applications*, pages 1–13, 2001.

[17] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the 3rd International Symposium on Recent Advances in Intrusion Detection (RAID'01)*, pages 85–103, 2001.

[18] R. Deraison. Nessus scanner, 1999. Available at http://www.nessus.org.

[19] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

[20] D. Farmer and E.H. Spafford. The COPS security checker system. In *USENIX Summer*, pages 165–170, 1990.

[21] M. Frigault and L. Wang. Measuring network security using bayesian network-based attack graphs. In *Proceedings of The 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*, 2008.

[22] M. Frigault, L. Wang, A. Singhal, and S. Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of QoP (QoP'08)*, 2008.

[23] N. Habra, Charlier B.L., A. Mounji, and I. Mathieu. ASAX: software architechture and rule-based language for universal audit trail analysis. In *Proceedings of the 2nd European Symposium on Research in Computer Security (ESORICS 1992)*, pages 430–450, 2004.

[24] J. Homer, A. Varikuti, X. Ou, and M.A. Mcqueen. Improving attack graph visualization through data reduction and attack grouping. In *Proceedings of the 5th international workshop on Visualization for Computer Security (VizSec'08)*, pages 68–79, 2008.

[25] K.S. Hoo. Metrics of network security. White Paper, 2004.

[26] IBM. IBM tivoli risk manager. Available at http://www.ibm.com/software/tivoli/products/risk-mgr/.

[27] SRI International. Event monitoring enabling responses to anomalous live disturbances (EMERALD). Available at http:// www.sdl.sri.com/projects/emerald/.

[28] S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer Academic Publisher, 2003.

[29] A. Jaquith. *Security Merics: Replacing Fear Uncertainity and Doubt.* Addison Wesley, 2007.

[30] S. Jha, O. Sheyner, and J.M. Wing. Two formal analysis of attack graph. In *Proceedings of the 15th Computer Security Foundation Workshop (CSFW'02)*, 2002.

[31] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375, 2002.

[32] Available at http://www.rob.cs.tu-bs.de/teaching/interactive/.

[33] S. Jajodia L. Wang T. Islam T. Long A. Singhal. An attack graph-based probabilistic security metric. In *IFIP WG 11.3 Conference on Data and Application Security*, 2008.

[34] W. Lee, J.B.D. Cabrera, A. Thomas, N. Balwalli, S. Saluja, and Y. Zhang. Performance adaptation in real-time intrusion detection systems. In *Proceedings of The 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, 2002.

[35] R. Lippmann and K. Ingols. An annotated review of past papers on attack graphs. Technical Report PR-IA-1, 2005.

[36] K. Manadhata, J.M. Wing, M.A. Flynn, and M.A. McQueen. Measuring the attack surfaces of two ftp daemons. In *Quality of Protection Workshop*, 2006.

[37] P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy Magazine*, 4(6):85–89, 2006.

[38] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID'02)*, pages 115–137, 2002.

[39] National Institute of Standards and Technology. Technology assessment: Methods for measuring the level of computer security. NIST Special Publication 500-133, 1985.

[40] P. Ning, Y. Cui, and D.S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, pages 245–254, 2002.

[41] P. Ning and D. Xu. Learning attack strategies from intrusion alerts. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, 2003.

[42] P. Ning, D. Xu, C.G. Healey, and R.S. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, pages 97–111, 2004.

[43] S. Noel and S. Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *CCS Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC'04)*, 2004.

[44] S. Noel and S. Jajodia. Understanding complex network attack graphs through clustered adjacency matrices. In *Proceedings of the 21st Annual Computer Security Applications Conference*, 2005.

[45] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency grpahs. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, 2003.

[46] R. Ortalo, Y. Deswarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Trans. Software Eng.*, 25(5):633–650, 1999.

[47] Available at http://www.caida.org/tools/visualization/otter/paper/.

[48] X. Ou, W. F. Govindavajhala, and A.W. Appel. Mulval: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[49] X. Ou, W. F. Govindavajhala, and M.A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, 2006.

[50] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38, New York, NY, USA, 2006. ACM Press.

[51] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 12 1999.

[52] C. Phillips and L. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the New Security Paradigms Workshop (NSPW'98)*, 1998.

[53] X. Qin and W. Lee. Statistical causality analysis of INFOSEC alert data. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, pages 591–627, 2003.

[54] X. Qin and W. Lee. Discovering novel attack strategies from INFOSEC alerts. In *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS 2004)*, pages 439–456, 2004.

[55] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Proceedings of ICDE*, 2003.

[56] I. Ray and N. Poolsappasit. Using attack trees to identify malicious attacks from authorized insiders. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS'05)*, 2005.

[57] R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Research on Security and Privacy (S&P'00)*, pages 156–165, 2000.

[58] R. Ritchey, 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 (ACSAC'02)*, page 25, 2002.

[59] B. Schneier. Attack trees. *Dr. Dobbs Journal*, 24(12):21–29, 1999.

[60] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02)*, 2002.

[61] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[62] M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo. Security metrics guide for information technology systems. NIST Special Publication 800-55, 2003.

[63] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX'01)*, 2001.

[64] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 New Security Paradigms Workshop (NSPW'00)*, pages 31–38, 2000.

[65] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 54–68, 2001.

[66] L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 11 2006.

[67] L. Wang, A. Singhal, and S. Jajodia. Measuring network security using attack graphs. In *Proceedings of the 3rd ACM workshop on Quality of protection (QoP'07)*, New York, NY, USA, 2007. ACM Press.

[68] L. Wang, A. Singhal, and S. Jajodia. Measuring the overall security of network configurations using attack graphs. In *Proceedings of 21th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2007)*, 2007.

[69] D. Xu and P. Ning. Alert correlation through triggering events and common resources. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 360–369, 2004.

[70] D. Xu and P. Ning. Privacy-preserving alert correlation: A concept hierarchy based approach. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)*, 2005.

[71] D. Zerkle and K. Levitt. Netkuang - a multi-host configuration vulnerability checker. In *Proceedings of the 6th USENIX Unix Security Symposium (USENIX'96)*, 1996.

[72] Y. Zhai, P. Ning, P. Iyer, and D. Reeves. Reasoning about complementary intrusion evidence. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 39–48, 2004.