

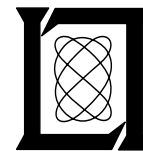
# **An Annotated Review of Past Papers on Attack Graphs**

**R.P. Lippmann  
K.W. Ingols**

**31 March 2005**

---

**Lincoln Laboratory**  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
*LEXINGTON, MASSACHUSETTS*



---

Prepared for the Department of the Air Force  
under Contract F19628-00-C-0002.

Approved for public release; distribution is unlimited.

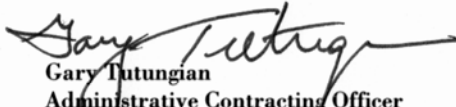
This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Department of the Air Force, Lighthouse Cyber Security Program, under Contract F19628-00-C-0002.

This report may be reproduced to satisfy needs of U.S. Government agencies.

The ESC Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

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

**FOR THE COMMANDER**



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

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission is given to destroy this document  
when it is no longer needed.

**Massachusetts Institute of Technology  
Lincoln Laboratory**

**An Annotated Review of Past Papers on Attack Graphs**

*R.P. Lippmann  
K.W. Ingols  
Group 62*

**Project Report IA-1**

**31 March 2005**

**Approved for public release; distribution is unlimited.**

**Lexington**

**Massachusetts**



## ABSTRACT

This report reviews past research papers that describe how to construct attack graphs, how to use them to improve security of computer networks, and how to use them to analyze alerts from intrusion detection systems. Two commercial systems are described [1, 2], and a summary table compares important characteristics of past research studies. For each study, information is provided on the number of attacker goals, how graphs are constructed, sizes of networks analyzed, how well the approach scales to larger networks, and the general approach. Although research has made significant progress in the past few years, no system has analyzed networks with more than 20 hosts, and computation for most approaches scales poorly and would be impractical for networks with more than even a few hundred hosts. Current approaches also are limited because many require extensive and difficult-to-obtain details on attacks, many assume that host-to-host reachability information between all hosts is already available, and many produce an attack graph but do not automatically generate recommendations from that graph. Researchers have suggested promising approaches to alleviate some of these limitations, including grouping hosts to improve scaling, using worst-case default values for unknown attack details, and symbolically analyzing attack graphs to generate recommendations that improve security for critical hosts. Future research should explore these and other approaches to develop attack graph construction and analysis algorithms that can be applied to large enterprise networks.



## **ACKNOWLEDGMENTS**

We would like to thank Robert LeBlanc for his support and encouragement under the Air Force Lighthouse Cyber Security program and Robert Cunningham for discussions concerning past papers.



## TABLE OF CONTENTS

|   |     |
|---|-----|
| Abstract  | iii |
| Acknowledgments                                   | v   |
| List of Tables                                    | ix  |
| 1. INTRODUCTION AND SUMMARY TABLE                 | 1   |
| 2. PROBLEMS AND POTENTIAL FUTURE DIRECTIONS       | 5   |
| 2.1 Scaling to Large Networks                     | 5   |
| 2.2 Obtaining Attack Details                      | 6   |
| 2.3 Computing Reachability                        | 7   |
| 2.4 Generating Recommendations from Attack Graphs | 7   |
| 3. DETAILED REVIEWS                               | 9   |
| 4. REFERENCES                                     | 23  |



## LIST OF TABLES

| <b>Table<br/>No.</b> |   | <b>Page<br/>No.</b> |
|----------------------|---|---------------------|
| 1                    | Research Papers That Describe Approaches<br>to Generate Attack Graphs | 2                   |



# 1. INTRODUCTION AND SUMMARY TABLE

Attack graphs are used to determine if designated goal states can be reached by attackers attempting to penetrate computer networks from initial starting states. For this use, they are graphs in which the starting node represents an attacker at a specified network location. Nodes and arcs represent actions the attacker takes and changes in the network state caused by these actions. Actions typically involve exploits or exploit steps that take advantage of vulnerabilities in software or protocols. The goal of these actions is for the attacker to obtain normally restricted privileges on one or more target hosts, where the target could be a user's computer, a router, a firewall, or some other network component. Many actions that compromise separate hosts and use them as stepping stones may be required in large attack graphs to reach the target host. A full attack graph will show all possible sequences of attacker actions that eventually lead to the desired level of privilege on the target. Some researchers use nodes to represent network states and arcs to represent attack actions, and some use other representations, including those in which both actions and network states are nodes and in which actions are nodes and network states are arcs. In addition, some attack graphs have one attacker starting location and one target host, some have multiple targets, and some have multiple attacker starting locations.

The papers included in this report focus on 3 goals related to attack graphs. First, many papers construct attack graphs to analyze network security (e.g., [3, 4]). In these papers, networks and their vulnerabilities are modeled, and these models are used to construct attack graphs. Attack graphs usually determine whether attackers starting from a specific location can gain normally restricted privilege levels on one or more important targets. A second goal of some papers is to present formal languages that can be used to describe actions and states in attack graphs (e.g., [5, 6]). These languages typically define the preconditions necessary for an attacker action or step to succeed and the post-conditions or changes in network state and attacker privilege levels that occur as a result of the attacker step. Since preconditions depend on characteristics of network hosts, these languages also describe network components (e.g., hosts, routers, switches, firewalls) and their vulnerabilities. Some languages rely on complex grammars and require many details (e.g., [4, 5, 16]), and others are simpler and more practical (e.g. [3, 7]). A third goal of some papers is to describe how attack graphs can be used to group the large numbers of alerts that are produced by intrusion detection systems (IDS) (e.g., [7, 8]). This is accomplished by first building attack graphs for a computer network that is being protected by the IDS. If IDS alerts can be associated with actions in the attack graph, then alerts that arrive in a sequence that is predicted when proceeding along a single attack graph path may indicate that an attacker is successfully performing the steps in that path. Grouping alerts generally requires the construction of attack graphs and a mechanism to match incoming IDS alerts to paths in the attack graph. When such matches are found, then a high-level alert is created for a security analyst. For papers that focus on grouping IDS alerts, this report focuses on the language used to represent actions and network characteristics and the approach used to construct attack graphs.

Table 1 presents a summary of the papers reviewed in this report. The first column provides the author, the date of the paper, and a cross-reference to the bibliography at the end of this report. The second column indicates if each attack graph has only one network host or network device as a target or if the attack graph simultaneously contains many targets. Constructing one attack graph for all hosts and network devices in a network as in [3] may result in a simpler representation that scales better for large networks than constructing an independent attack graph for each target. The third column in Table 1 describes the approach used to generate attack graphs. If examples of attack-graph generation are presented, this column also includes the maximum number of hosts and number of distinct vulnerabilities in the example. The fourth column in Table 1 describes how the approach used to generate attack graphs scales as the number of hosts in a network (N) increases. A scaling analysis is sometimes provided, but often scaling properties are inferred based on the approach or on timing measurements provided by the authors. The last column in Table 1 provides some summary comments on unique aspects of the paper.

**Table 1**  
**Research Papers That Describe Approaches to Generate Attack Graphs**

| Paper            | Goals          | Graph Construction<br>Hosts/Vulnerabilities                                     | Scaling  | Comments  |
|------------------|----------------|---|--|---|
| Ammann, 2002 [9] | One final goal | Hand generated artificial network, hand analysis<br>3 Hosts/6 Vulns.            | Best research algorithm to date, base computation grows as $N^6$                   | Describes algorithms to find attack paths to specific goals. Finds the shortest path, all exploits that can be used to reach the goal, and paths of any length. Scales to only hundreds of nodes.   |
| Artz, 2002 [3]   | Multiple goals | Depth-first search in C++ program for a realistic network<br>17 Hosts/21 Vulns. | Poor, similar to full graph, reasonable run times (90 seconds) for 17-host network | Finds all paths to all possible goals and performs reachability computations. Search depth can be limited, determines visibility of attack paths to intrusion detection systems, imports hand-modified information from Nessus [10] scanner and the ICAT vulnerability database [11]. |
| Bilar, 2003 [12] | One final goal | Hand generated graph as in [13]<br>1 Host                                       | —  | Determines software to patch or update on a single host. Analyzed 6 different operating systems containing a total of 129 vulnerabilities. Found that all vulnerabilities need to be patched to reduce risk substantially.  |

**Table 1 continued**

| <b>Paper</b>             | <b>Goals</b>      | <b>Graph Construction<br/>Hosts/Vulnerabilities</b>                                     | <b>Scaling</b>                                | <b>Comments</b>  |
|--------------------------|-------------------|---|---|--|
| Cheung, 2003 [14]        | One final goal    | Proof-of-concept code for one known scenario  | Poor  | Describes the CAML attack language that can be used to correlate IDS alerts into scenarios.  |
| Cuppens, 2001–2 [6, 7]   | Multiple goals    | Description only  | —   | Describes the LAMBDA attack language and shows how it can be used to correlate intrusion detection system (IDS) alerts into scenarios using pair-wise rules.   |
| Dawkins, 2004 [15]       | One final goal    | Proof-of-concept with hand data entry, small artificial network<br>4 Hosts/ 4 Vulns.    | Poor, builds full graph first                 | Builds a full attack graph, extracts paths to desired final goal, and simplifies these into “a minimum cut set” where the goal cannot be reached if any single vulnerability is removed.   |
| Gorodetski, 2003 [16]    | One final goal    | Program generates individual random attack paths  | —   | Uses a formal grammar to specify allowable paths and generates individual paths from the grammar.  |
| Jajodia, 2003 [4, 17–19] | One final goal    | Automatic graph generation using algorithm from [9]<br>3–17 Hosts/4–? Vulns.            | Best to date, base computation grows as $N^6$ | Reads in vulnerability and reachability information from the Nessus [10] scanner, computes attack graphs using the approach described in [9], makes recommendations to prevent access to critical hosts, and simplifies the graph for visualization. |
| McDermott, 2001 [20]     | Goal and subgoals | By hand   | Poor  | Formalization of the approach described in [13].   |
| Moore, 2001 [21]         | One final goal    | By hand   | Poor  | Suggests using attack graphs described by [13] to describe attacks.  |
| Ning, 2003 [8]           | Multiple goals    | Automatic scenario generation from IDS alerts. Tested with a few small known scenarios. | Poor, NP complete                             | Presents a language that describes vulnerabilities and IDS alerts plus an approach to group alerts into graphs that display scenarios. Graphs have fewer than 10 nodes.  |

**Table 1 continued**

| <b>Paper</b>        | <b>Goals</b>   | <b>Graph Construction<br/>Hosts/Vulnerabilities</b>   | <b>Scaling</b>                                      | <b>Comments</b>  |
|---------------------|----------------|---|---|--|
| Ortalo, 1999 [22]   | One final goal | Automatic graph generation from UNIX host-based vulnerability checker over 21 months.<br>1 Host/13 Vulns. | Poor (sometimes couldn't build breadth-first graph) | Builds privilege graphs showing how an insider on a single UNIX host can increase privilege to admin or root using 13 vulnerabilities. Computes the number of steps in the shortest path and expected mean effort to failure but requires exploit success probability estimates. |
| Ritchey, 2000 [23]  | One final goal | Hand-generated artificial network input to model checker<br>4 Hosts                                       | Poor  | Uses model checking to determine if a final goal is reachable. Other papers (e.g., [4]) note that this approach does not scale well.   |
| Sheyner, 2002 [24]  | One final goal | Hand-generated artificial network input to model checker<br>3 Hosts/4 Vulns. to<br>5 Hosts/7 Vulns.       | Poor  | Uses model checking to determine if a final goal is reachable. It took 5 seconds for 4 hosts/4 vulns. but 2 hours for 5 hosts/8 vulnerabilities. Also analyzes the visibility of different attack paths to intrusion detection systems.  |
| Schneier, 1999 [13] | One final goal | By hand   | Poor  | Early discussion that presents a widely-used representation for attack graphs.   |
| Swiler, 2001 [25]   | One final goal | Proof-of-concept hand vulnerability entry and small artificial network<br>2 Hosts/5 Vulns.                | Poor (builds almost full graph first)               | Builds a semi-collapsed full attack graph that still scales combinatorically. Uses this large graph to compute shortest paths to specified goals with costs on links. A database holds information on the network, vulnerabilities, and attacker models.                         |
| Templeton, 2002 [5] | One final goal | Proof-of-concept code   | —   | One of the first papers to suggest modeling attack components using pre- and post-conditions and using forward chaining to build attack graphs. Recommends highly detailed attack component models.  |
| Tidwell, 2001 [26]  | One final goal | Description only, example uses 6 hosts.   | —   | Suggests using BNF grammars to describe network components and also attack pre- and post-conditions.   |

## 2. PROBLEMS AND POTENTIAL FUTURE DIRECTIONS

The detailed reviews in the following section and Table 1 demonstrate several problems or limitations of past attack-graph research. They also suggest future directions that might overcome many of these limitations.

### 2.1 SCALING TO LARGE NETWORKS

The first major limitation of prior studies is that most past algorithms have only been able to generate attack graphs on small networks with fewer than 20 hosts. The most capable systems have been developed at George Mason University [4, 9, 17–19] and MIT Lincoln Laboratory [3]. Both systems have been used to construct attack graphs for small 17-host networks that were simulations of actual networks, and scaling for both systems is poor. Computation required for the approach used at George Mason University described in [9] grows as  $N^6$ , where  $N$  represents the number of hosts in a network. Although this is not combinatorial and is the best upper bound reported to date, as stated in [9], this approach will only scale to networks with at most “tens or hundreds of hosts.”

Timing measurements and an algorithmic analysis demonstrate that scaling for the approach used at MIT Lincoln Laboratory [3] is also poor. This approach is more complete than that described in [9] because instead of assuming that reachability between all hosts is provided, it computes the reachability between all hosts before generating attack graphs. Reachability computations model the effect of gateways and firewalls to determine which other hosts in a network can be reached from any host. Reachability information is required to construct attack graphs because an attack can only proceed to new victims that can be reached from compromised hosts. Computing reachability using the simplest approach requires an additional  $N^2$  host-to-host reachability analysis prior to the attack-graph construction computation. The approach described in [3] also simultaneously builds an attack graph that shows how all targets in a network can be compromised by an attacker from a given starting location. A small 17-host network was analyzed in less than 90 seconds (including reachability computations), but scaling to larger networks is poor because this approach constructs a full attack graph that includes all possible paths to reach victims. In even a simple subnet with no filtering, the size of such a full graph grows combinatorially as the number of hosts increases.

Other approaches that construct full attack graphs [15, 22, 25, 26] will also exhibit poor scaling. For example, full attack graphs often could not be computed in [22] even when analyzing the effect of only 13 vulnerabilities on one file system. Approaches that use model checking to explore the entire space of allowable attack paths, as described in [24], also exhibit poor scaling. In these experiments the run time was only 5 seconds for a 4-host network with 5 vulnerabilities. This increased to 3 hours when the number of hosts was increased by one to 5 and the number of vulnerabilities was increased to 8.

In practice it is desirable to compute attack graphs for enterprise networks with 10,000 to 100,000 hosts. Many of the graph-building algorithms used in past research papers create a full attack graph and thus scale combinatorically with  $N$ , where  $N$  represents the number of hosts in a network. The best research algorithm [9] scales as  $N^6$ , and an approach described in a patent [2] claims to scale as  $N^4$ . None of these algorithms are acceptable for large enterprise networks. Only algorithms that scale linearly or quadratically in the number of hosts will be practical for large networks. Two approaches that may improve scaling are suggested by past studies. The first is to produce a restricted graph that is designed solely to answer questions required to analyze network security. One restricted set of questions is (1) what hosts can be compromised by an attacker starting at a given network location? and (2) what is the minimum set of exploits that enable successful attacks against the specified goals? It may be possible to develop a graph-building algorithm that answers only these questions but has complexity that is less than  $N^4$ .

A second approach that can be used to reduce the complexity of graph building is to group or aggregate similar hosts together. This reduces  $N$  by replacing many individual hosts with representative hosts. Aggregation was suggested in [25] for this purpose and in [18] to simplify the visual presentation of attack graphs. The simplest type of aggregation is to collapse hosts that are fully connected into protection domains. For example, all hosts on a single local area network (LAN) or subnet could be aggregated if they have the same reachability from outside the network and if they share the same vulnerabilities. It would also be possible to aggregate hosts across connected subnets when the gateway that connects subnets performs no filtering. A second type of aggregation is to group multiple exploits between two hosts when they have the same pre- and post-conditions.

## 2.2 OBTAINING ATTACK DETAILS

A second major weakness of many past approaches is that information used to describe pre- and post-conditions for exploits or attack components must be entered by hand. This is labor intensive and difficult, especially to model attacks using the amount of detail required by many studies (e.g., [4, 5]). For example, in [12] it is noted that it can take from 10 minutes to hours for an analyst to determine the pre- and post-conditions for a single attack component. Although some vulnerability databases exist (e.g., [11]), they do not contain the machine-readable details required to accurately produce many of the attack graphs shown in past papers. Such graphs require extensive human analysis of vulnerabilities and attack components or development of automated approaches to extract this information from text in attack descriptions. An alternative approach is to only require limited information concerning attack components and vulnerabilities as in [3] and to fill in unknown attack and vulnerability details with reasonable default values as suggested in [25]. If a worst-case assumption is made to fill in unknown details, in many cases the attack graph analysis will identify on a few hosts a small set of critical vulnerabilities that enable an outside attacker to progress to internal targets. This small set of vulnerabilities can be hand analyzed and details confirmed to verify the attack-graph analysis. This is much simpler than hand verifying the analysis of all vulnerabilities.

## **2.3 COMPUTING REACHABILITY**

As noted above, reachability computations determine which other hosts in a network can be reached from a given host by modeling the effect of gateways and firewalls. A weakness of many past approaches is the assumption that reachability information is available prior to computing attack graphs (e.g., [24]) or that it can be obtained using a vulnerability scanner from each subnet in the analysis (e.g., [4]). Determining reachability between all hosts in large networks with many firewalls is a computationally complex task. It is almost impossible to determine reachability using vulnerability scanners, and this approach can severely underestimate reachability. Firewalls can contain hundreds to thousands of access control rules, network address translation (NAT) rules, and network objects that represent groups of IP addresses (e.g., [27]). A single scan from one Internet protocol (IP) address will only exercise a few of these rules. Such scans will miss rules that apply to other source IP addresses or to destination IP addresses not included in the scan. It is difficult, in general, to know which source and destination addresses will be treated differently by a firewall due to NAT rules and the complexity of other rules. In addition, a single scan will not discover the effect of time-dependent firewall rules. Accurately determining the reachability between hosts in separate subnets requires downloading and analyzing configuration files for firewalls, routers, switches, virtual private networks, personal firewalls, and other network infrastructure devices that perform filtering. Future tools that use attack graphs to analyze network security on large enterprise networks must perform this type of analysis.

## **2.4 GENERATING RECOMMENDATIONS FROM ATTACK GRAPHS**

Past studies have demonstrated that attack graphs can become large and complex even for networks with fewer than 20 hosts (e.g., [3, 4]). Such large graphs, once generated, can be difficult to analyze and understand. An alternative approach is to not only generate attack graphs, but to also automatically analyze attack graphs to address security issues and make recommendations to improve security. Recommendations could, for example, suggest changes in the network architecture or patches for installed software that protect important hosts but result in few changes. One past study [17] developed approaches to make such recommendations. Future work should extend this work and explore approaches that can scale to large networks.



### 3. DETAILED REVIEWS

**Amenaza (2004), [1]** Secur/Graph Attack Graph Modeling, Amenaza Technologies Limited:  
<http://www.amenaza.com/company.html>.

This commercial company provides a software program that helps automate the attack-graph construction and analysis techniques presented in [13]. It makes it possible to construct AND/OR attack graphs using a graphical user interface and allows various types of costs or possibilities to be attached to attacker actions. It can also then compute the cost or probability of success of different attack paths and identify shortest or most-likely-to-succeed paths. Attack graphs must be constructed by hand, and then paths can be automatically analyzed. Examples illustrate attack graphs with 10's of nodes.

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

This paper describes an algorithm that can be used to create attack graphs. The complexity of this algorithm is polynomial and not combinatoric, and we are aware of no other research algorithm that had a better upper bound on complexity at the time this paper was published. In a network with  $N$  hosts, it requires reachability information between all hosts on all TCP/IP ports and protocols of interest, information concerning vulnerabilities on hosts, attacker privilege levels on all hosts, and information on exploits including pre- and post-conditions. Computation in the initial marking phase of the algorithm grows as  $N^6E$ , where  $N$  is the number of hosts and  $E$  is the number of exploits. This computation scales poorly to large networks and motivates the comment in the paper that this approach is useful for networks with at most hundreds of hosts. Algorithms are presented that use information obtained in the marking phase to compute paths to a final goal, to determine all exploits that can be used to reach a goal, and to find the path with the fewest attack steps to a goal. The marking phase could also be used to determine the hosts that can be compromised for a given attacker starting location. A simple example is presented using an artificial network with 3 hosts and 6 vulnerabilities. This network was analyzed by hand to demonstrate the algorithm. The algorithm that determines exploits that can be used to reach the final goal would include all exploits on a full graph pruned to contain only paths ending at the final goal.

The approach is restricted to monotonic attacks in which a component attack never changes the network state in a way that eliminates a precondition for another attack. This means that some attacks such as Denial of Service (DoS) attacks against hosts or services cannot be modeled and that other attacks that include DoS components against services need to be modeled by ignoring the DoS components. It also means that information along attack paths concerning change of network state (e.g., installing new passwords, a DoS attack against services, patching existing vulnerabilities) may not be preserved and will

not be available for forensic analysis that depends on detecting changes of state to network components. Finally, the need to precompute reachability before attack-graph generation means that this approach cannot model attacks in which firewalls or other network infrastructure components are compromised and firewall or filtering rules are changed, thus changing the underlying reachability between hosts.

**Artz, M. (2002), [3]** *NETspa, A Network Security Planning Architecture*, M.S. Thesis, Cambridge: Massachusetts Institute of Technology, May 2002.

This thesis describes the first version of the NetSPA (Network Security Planning Architecture) system that generates worst-case attack graphs. This is a C++ tool that inputs information from a custom database on host and software types and versions, intrusion detection system placement, gateways between subnets, firewall rules, and exploits. Other information, including critical network resources and the attacker starting locations, is provided at run time. Although information on network vulnerabilities is collected using the Nessus [10] vulnerability scanner, this information must be entered into the database by hand. Information on firewall rules and the network topology also must be entered into the database by hand. Exploits are described using a simple language that specifies the requirements for the exploit (usually connectivity to a vulnerable victim), the effect of the exploit (usually the privilege level on the victim and any side effects such as a DoS of the service used for a buffer overflow attack), and whether this exploit is visible to a network intrusion detection system. Some trust relationships are also provided to model sniffer attacks. Analysis first involves computing connectivity between all hosts using network topology information and firewall rules. Attack graphs are then built, using a depth-limited forward-chaining depth-first search. This tool produced attack graphs that were identical to those produced by the model checker described in [24] for the same small test network. It was also evaluated using a realistic network with 17 representative hosts from an actual network, 21 unique vulnerability types, internal and DMZ networks, and a firewall with 12 rules. Attack graphs were generated to determine hosts that can be compromised by novice and expert external attackers; the effect of combining DNS, SMTP, and web servers; the best placement for an IDS to detect external attacks; and the effect of new “zero-day” vulnerabilities on DMZ servers. Although the largest graphs for the 17-host network took less than 90 seconds to produce when the graph depth was limited to three, scaling is poor because the graph produced is similar to a full graph.

**Bilar, D. (2003), [12]** *Quantitative Risk Analysis of Computer Networks*, Ph.D. Thesis, Thayer School of Engineering, Hanover, New Hampshire: Dartmouth College, June 2003.

This thesis describes an optimization approach to determine what software running on an individual computer when patched provides the greatest reduction in risk where risk is defined as the probability of loss times the amount of loss. Data required for this approach include extensive details concerning versions of software and their vulnerabilities for each host modeled. The following details need to be

provided: (1) the consequences of exploits, (2) the loss associated with each consequence, (3) the probability of success for each exploit as a function of time, and (4) the cost of changing from one software version to another. This information was generated manually for 6 workstations running different operating systems containing a total of 129 unique vulnerabilities. Simple AND/OR attack graphs were built for each host by hand and used to combine attack success probabilities. These graphs were found to be unnecessary because the final conclusion after analyzing all the data was that “almost all if not all of the faults have to be eliminated to have an appreciable effect on the consequence risk probabilities.” This means that a host needs to be completely patched to ensure that it cannot be compromised. A second conclusion is that it is better to first patch the software that can be compromised by a remote user and thus has the highest risk. An interesting comment is that analyzing any single vulnerability to determine the pre- and post-conditions can take from 10 minutes to 2 hours.

**Cheung, S., U. Lindqvist, et al. (2003), [14]** “Modeling Multistep Cyber Attacks for Scenario Recognition,” *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, IEEE, 1, 284–292.

This paper describes a language called CAML (Correlated Attack Modeling Language) that can be used to model attack scenarios and recognize scenarios from intrusion-detection alerts. It is similar to LAMBDA [6] and JIGSAW [5] in that it defines preconditions and post-conditions for attack actions (called modules) and also describes the state of network components. Preconditions include information on hosts, services, file names and privileges, users, and information known by users such as the password of another user. The CAML language was tested by creating a model of an attack scenario that included 13 modules by hand and that shows that a manually created system could correctly chain attack modules and recognize simulated alerts generated using the known scenario. The weakness of this approach is that modules are labor intensive to create and there is no efficient tool to automatically create scenarios.

**Cuppens, F. and R. Ortalo (2001), [6]** “LAMBDA: A Language to Model a Database for Detection of Attacks,” *Recent Advances in Intrusion Detection (RAID) 2000, Lecture Notes in Computer Science, vol. 1907*, H. Debar, L. Me and F. Wu, Eds., Berlin: Springer Verlag.

This paper describes a language, LAMBDA, that can be used to describe attack scenarios as a combination of actions. As in the JIGSAW model [5], each action has conditions or requirements that must be satisfied for the action to succeed, and successful actions affect the network and may satisfy conditions for other actions. Actions can be combined using operators that specify sequencing, parallel unconstrained execution, absence of a condition, nondeterministic choice between multiple equivalent actions, and synchronized execution. As with JIGSAW, this language is labor intensive to use, only a few examples are provided, and an automated tool to create scenarios is not presented.

**Cuppens, F. (2002), [7]** “Alert Correlation in a Cooperative Intrusion Detection Framework,” *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC, IEEE Computer Society.

This paper describes how the LAMBDA language described in [6] can be used to link alerts from intrusion detection systems into scenarios. It includes a comprehensive description of the language and illustrates how it can be used to model a variety of attacks. Preconditions for actions are made explicit in this paper. One precondition specifies access levels of the intruder on the target system, including remote, local, user, root, and physical. Another specifies effects of attacks on the target system, including denial of service, alter, and (illegal) use. A final precondition specifies that a service is active on a target or source system. Post-conditions or attack effects include attacker knowledge about the target system that might be gained, for example, by a port scan. Intrusion-detection alerts are correlated by automatically generating rules from the LAMBDA action descriptions that link pairs of attacks together where success of the first attack may enable the second. These rules are generated offline and used to correlate intrusion detection alerts. The paper describes an approach that is not yet implemented. More actions need to be described in LAMBDA, and an approach needs to be developed to specify a global intrusion objective and determine if alerts represent actions that lead to that objective. This approach to correlating intrusion-detection alerts and developing attack scenarios is limited by the human labor required to describe actions and by the need to develop an efficient automated approach to create scenarios with global objectives.

**Dawkins, J. and J. Hale (2004), [15]** “A Systematic Approach to Multi-Stage Network Attack Analysis,” *Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04), 2004*. IEEE Computer Society:  
<http://csdl.computer.org/comp/proceedings/iwia/2004/2117/00/21170048abs.htm>.

This paper describes a framework that can be used to create and analyze attack graphs in computer networks. Models are built of the network (hosts and boundaries), of the privileges of the attacker on hosts and the reachability of hosts from the attacker, and of vulnerabilities. Vulnerabilities are modeled by describing pre- and post-conditions as in JIGSAW and LAMBDA. Analysis involves first producing a full breadth-first attack graph that includes all possible paths with no pruning for specific attack goals. The attack graph is limited by depth to stop after a given number of vulnerabilities have been exercised in sequence in each path. The full attack graph is then analyzed to identify attack paths that end in specific top-level goals. These paths are then analyzed to find the “minimum cut set” which is the smallest collection of paths such that if any one vulnerability is removed, they still correctly predict whether the final goal is reached. After a minimum-cut-set attack graph is generated, it can be used to determine the probability of achieving a top-level goal, but this requires knowledge of the probability of each component attack and the unlikely assumption that these probabilities are independent. A proof-of-concept tool is described that reads in network, vulnerability, and attacker models that are expressed in XML. It builds full attack graphs to a specified depth, allows a user to select a top-level goal, extracts paths that reach these goals, and simplifies these paths to produce a minimum-cut-set graph. Inputs to this tool are hand generated and the tool has only been applied to a small artificial network with 4 hosts and 4

vulnerabilities. Scaling results are not presented, but because a full graph is generated, scaling will be poor because the number of nodes in full graphs can grow combinatorically as the number of hosts in a network grows. The algorithm used to generate a minimum-cut-set graph from the full graph is also not specified.

One useful idea presented in this paper is to store network state changes differentially along attack paths. Any component attack may change the state of the network. An inefficient approach to storing these state changes would be to replicate the complete network state at each node in the attack graph and then use this state for the remainder of the graph. Instead, the paper suggests storing only the differences between the network state at each node in the attack graph. This makes it possible to use non-monotonic attacks that disable other attacks and to model DoS attacks.

**Gorodetski, V. and I. Kotenko (2002), [16]** “Attacks against Computer Network: Formal Grammar-based Framework and Simulation Tool,” *Lecture Notes in Computer Science, vol. 2516: Recent Advances in Intrusion Detection (RAID) 2002*, A. Wespi, G. Vigna, and L. Deri, Eds., Berlin: Springer Verlag: <http://space.iias.spb.su/ai/doc/RAID-2002.pdf>.

Instead of modeling attack graphs graphically, this paper proposes a model based on a formal stochastic context-free grammar where an attack hierarchy is created and the substitution operation is used to create specific attack paths. A proof-of-concept tool is developed for a small set of attacks and a test network where detailed information on the network and attacks is available. The tool allows component attacks to succeed or fail with given probabilities and generates individual random attack paths for each run. All information is entered manually, and the tool implements a finite state machine that generates attack paths from the formal grammar definition of component attacks and the network.

**Jajodia, S., S. Noel, and B. O’Berry (2003), [4]** “Topological Analysis of Network Attack Vulnerability,” *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds., Dordrecht, Netherlands: Kluwer Academic Publisher: <http://www.isse.gmu.edu/~snoel/Kluwer%20TVA%20chapter.pdf>.

This paper describes the Topological Vulnerability Analysis (TVA) tool. It is one of the most comprehensive tools developed to date for the purpose of building and analyzing attack graphs and has served as a framework at George Mason University for attack-graph research. This tool requires reachability and vulnerability information obtained by Nessus [10] scans, pre- and post-conditions for exploits entered by hand, and information on attacker goals and the network provided by hand. The polynomial time algorithm described in [9] is used to construct and analyze attack graphs. After the marking phase of this algorithm is complete, it is possible to construct attack paths to attack goals and also use a symbolic analysis technique to identify sets of actions (e.g., patch vulnerabilities, remove

network services, remove programs on hosts) that, if followed, prevent the attacker from reaching goal states. No information is presented about how this symbolic analysis scales for large networks. An example is presented for a small artificial network with 3 hosts, 4 exploits, and a firewall with a total of 6 rules and network objects. A simple attack graph shows how an outside attacker can obtain root-level privileges on an internal machine that cannot be directly compromised from the outside. The attacker uses a vulnerability in the IIS web server to compromise this host and then uses this web server as a stepping stone. This is accomplished by first using the Remote Copy (RCP) program to download a root kit to the web server. A port forward tool is then installed, and the attacker compromises the victim machine through the web server using an attack on the FTP server running on the victim machine. In this graph, the attacker uses the non-target internal host as a stepping stone. Attack steps include low-level attacker actions such as downloading programs from external hosts. The symbolic analysis demonstrates that many different changes to the network, including patching the IIS web server, removing the RCP program on the IIS server, and patching the FTP server on the victim, prevent the attacker from achieving root on the target machine. Firewall rules are not analyzed but determined implicitly by using Nessus to scan between subnets through firewalls to determine reachability through between subnets.

This is one of the most capable tools developed to date for the generation and analysis of attack graphs. Some information on vulnerabilities is automatically imported, the algorithm can scale to hundreds of nodes, and the goal is to make recommendations to improve security and not simply create attack graphs. The TVA tool, and others, are still limited. Major limitations with the TVA approach include the following:

- **Exploit information must be entered by hand**  
Information required to describe pre- and post-conditions for exploits must be entered by hand, and the exploits must be analyzed by hand because detailed information is required. This is labor intensive and difficult, especially to model attacks using the amount of detail shown in this paper. Although some vulnerability databases exist (e.g., [11]), they do not contain the amount of detail required to accurately produce the types of attack graphs shown in this paper.
- **Firewall and router rules are not imported and analyzed**  
A major limitation of the TVA tool is that firewall and routing rules are not imported directly and analyzed. Instead, the Nessus vulnerability scanner is used to determine whether it is possible to connect from one specific IP address in one subnet to IP addresses in another through the firewall. This approach can severely underestimate reachability between subnets. Firewalls can contain hundreds to thousands of access control rules, network address translation (NAT) rules, and network objects that represent groups of IP addresses (e.g., [27]). A single Nessus scan from one IP address will only exercise a few of these rules. Such scans will not exercise rules that apply to other source IP addresses or to destination IP addresses not included in the scan. It is difficult, in general, to know which source and destination addresses will be treated differently by a firewall due to NAT rules and the complexity of other rules. In addition, a single scan will not discover the effects of time-dependent firewall rules. Accurately

determining the reachability between hosts in separate subnets requires downloading and analyzing firewall and router rules. Depending on Nessus scans will lead to attack graphs that often ignore many avenues of attack. It also requires scanning the entire network from every subnet. This is impractical except for small enterprise networks.

- **Poor scaling to large networks**

The underlying algorithms used by TVA scale as  $N^6$ , where  $N$  is the number of hosts in the network as described in [9]. As noted in [9], the approach will probably not scale to more than hundreds of hosts. In addition, the approach is limited to monotonic attacks in which no component attack action makes a different attack impossible. It thus precludes inclusion of DoS attacks.

- **Requires low-level attack details**

This approach assumes that it is possible to obtain host, network, and attacker information necessary to support the detailed low-level attack modeling used and to model these details correctly. This is usually not possible because detailed host monitoring is frequently not allowed and is not practical on enterprise networks and because attacker modeling is extremely difficult. The problems faced when modeling detailed attack steps are demonstrated by the simple attack graph presented. A critical step was downloading a root kit program using the RCP program, and one of the network changes that the TVA analysis claimed could prevent compromising the victim is to remove the RCP program. In most Windows NT hosts, there are many alternative approaches to downloading files and RCP is not essential. Other approaches to downloading files to a compromised IIS server include sending the file along with the compromise, sending the file as an HTTP request after the compromise successfully responds to the attacker, sending HTTP cookies, using SSH, tunneling the file through HTTP, ICMP, or DNS, and other approaches that have been used in automated exploits and worms. The attack graph shown is incomplete because these other approaches are not modeled. It is thus incorrect to assume that removing the RCP program on the web server will protect the victim machine and block this attack. Since the attacker can run arbitrary code on the web server, the initial attack can include new programs that can automatically compromise the target machine and others, as seen in recent worms and “bot” networks [28]. Such low-level attack details are difficult to model correctly. An alternative worst-case approach is to assume that the attacker can download tools and proceed to launch another attack from any compromised host instead of trying to model the many approaches to download and run attack tools. This approach is used in [3].

**McDermott, J. P. (2001), [20]** “Attack Net Penetration Testing,” *Proceedings of the 2000 Workshop on New Security Paradigms*, New York: ACM Press, pp. 15–21.

This paper proposes a more formal definition of attack nets using a disjunctive Petri net. This explicitly differentiates intermediate states from actions that change state, and it models attack progress and concurrency using Petri net tokens. This approach has all the limitations of [13] but is a more formal approach that eliminates the lack of definitions in [13]. Some examples demonstrate how it can model low-level concurrency and provide detailed attack models.

**Moore, A., R. Ellison, et al. (2001), [21]** “Attack Modeling for Information Security and Survivability,” Software Engineering Institute: <http://www.cert.org/archive/pdf/01tn001.pdf>.

This paper suggests using attack graphs as described by [13] as an approach to document computer attacks in a structured and reusable form. Attack graphs are hand generated, and no standards for storing or sharing attack graphs are proposed other than the text format used by [13].

**Ning, P. and D. Xu (2003), [8]** “Learning attack strategies from intrusion alerts,” *Proceedings of the 10th ACM Conference on Computer and Communications Security*, New York: ACM Press, pp. 200–209.

This paper describes an attack description language similar to LAMBDA [6] and JIGSAW [5] that includes preconditions and post-conditions for intrusions (called hyperalerts) and is designed to correlate alerts from intrusion detection systems. The language was used to describe alert components from 5 different small attack scenarios that were run on isolated test bed networks. Alerts from intrusion detection systems on these networks were analyzed and used to produce scenario graphs that were similar to the original underlying attack scenarios except for missing steps caused by intrusion detection systems that couldn't observe all attack steps. This approach uses generic graph matching approaches to simplify the generated graphs. The paper states that this analysis is NP complete but that the graphs generated typically have fewer than 10 nodes and can be generated with reasonable response times. This approach would not scale to large scenario graphs. In addition, all models again need to be generated by hand and future work is proposed to deal with false alarms and attack steps missed by intrusion detection systems.

**Noel, S., S. Jajodia, et al. (2003), [17]** “Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs,” *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada: <http://www.isse.gmu.edu/~snoel/2003%20ACSAC.pdf>.

This paper provides further details that explain how network hardening recommendations are made in the TVA system described in [4]. Exploit dependency graphs are first constructed, and a symbolic description of exploits that lead to a given goal is created. This is then simplified into a symbolic equation that specifies the network preconditions necessary for the goal to be reached. Analysis of this equation, given the cost of making each network change, makes it possible to recommend a least-cost change that prevents the goal from being reached. No information is provided on how this symbolic analysis is performed or how it scales other than a comment that the number of terms in the equation can grow exponentially in the number of network preconditions. In a large enterprise network there can be many preconditions because these include vulnerabilities on each host, reachability between all host pairs, and trust relationships between hosts.

**Noel, S. and S. Jajodia (2004), [18]** “Managing Attack Graph Complexity Through Visual Hierarchical Aggregation,” *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, New York: ACM Press: <http://www.isse.gmu.edu/~sis/faculty/jajodia-vizsec-2004.pdf>.

This paper describes various approaches to collapse parts of attack graphs generated by the TVA system [4] to make visual understanding easier. The display uses exploit-dependency graphs. In these graphs, exploits between different hosts are treated as separate and unique and appear only once as nodes. There are thus at most  $N^2V$  nodes where  $N$  is the number of hosts and  $V$  is the number of unique vulnerabilities in the network. This representation often requires fewer nodes than graphs that include a node for each host privilege level and arcs for exploits used to raise privilege levels on hosts. The paper states that “perhaps the greatest challenge in making network attack graphs practical ... is managing their visual complexity in user interaction.” Three basic approaches to grouping exploit dependency graphs are presented. First, multiple exploits between the same two hosts can be aggregated. Second, hosts that are fully connected in protection domains (e.g., on a single LAN with no filtering devices) can be aggregated. Finally, precondition and post-condition nodes can be collapsed if they are for the same exploit. A user interface is presented that makes it easy to apply these different types of aggregation and “drill down” to examine attack graph details. Examples use networks with at most 16 hosts.

This is new and interesting work, but future analyses should extend this research to determine the role visualization plays in improving network security using attack graphs. Different types of security-related information can be presented using visualization, and this information can be displayed in many ways. Visualization can have many goals, including displaying and justifying automatically generated recommendations, allowing manual correction of vulnerability scanner analyses, verifying network topology data, mapping specific attack graphs onto a network diagram, and changing firewall rules or patching hosts to examine the effect on attack graphs. These different goals may require alternative

approaches to displaying attack graphs and the network under analysis. Future work should explore alternate visualization approaches and determine if any one is easier to interpret by system administrators. Exploit-dependency graphs are one approach, but many others have been suggested.

**Ortalo, R., Y. Deswarte, et al. (1999), [22]** “Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security,” *IEEE Transactions on Software Engineering*. **25**(5): 633–650.

This paper describes a thorough study that illustrates how to use privilege graphs to describe the security of a single UNIX host. Nodes on these graphs represent a set of privileges for a user or group of users, and arcs represent vulnerabilities that can transition between nodes. For example, a vulnerability could represent an easily guessed root password that a user can use to obtain root privileges. Each vulnerability is assigned a measure of effort required. In the described experiment, 4 values were used (0.1, 0.01, 0.001, 0.0001). Procedures to compute the mean effort to failure (METF) were also presented that average over paths in the privilege graph. A software tool was developed that automatically probes the UNIX file system to determine which of 13 known vulnerabilities are present. This software is similar to many existing host-based vulnerability scanners. Other software attempted to build privilege graphs that start with user privilege and attempt to reach either root privilege or the privilege of the administration group using the vulnerabilities found. Different graph-building approaches were explored, including a breadth-first approach (denoted TM), a depth-first approach (denoted ML), and a shortest-path approach (denoted SP). Results from one UNIX system monitored daily over 21 months are presented. It was found that breadth-first attack graphs could sometimes not be computed because they were too large, even with only 13 vulnerabilities. This graph is similar to a “full” attack graph that finds all possible paths to a goal. Of the 3 measures, the SP graph changed the least over time. It represents a worst-case analysis in which the attacker takes the shortest path that is most likely to succeed. The METF measures on breadth-first and depth-first graphs varied over time even when the SP graph remained constant. The authors state that these analyses “provide useful feedback to the security administrators.” It is recommended that all the different measures should be computed and used because they represent different attacker models. This approach uses automated tools to find a small set of vulnerabilities but only analyzes a single host and uses graph-building tools and measures that may not scale to large networks.

**Ritchey, R. and P. Amman (2000), [23]** “Using Model Checking to Analyze Network Vulnerabilities,” *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pp. 156–165: <http://portal.acm.org/citation.cfm?id=884423&dl=ACM&coll=GUIDE>.

This paper provides a thorough and explicit example that shows how model checking can be used to determine if a final goal state is reachable for an attacker starting with limited privileges on a network. If the goal state is reachable, the model checker produces one example attack path that shows how the state is reached. The model checker is provided information on network hosts and their vulnerabilities,

reachability between all hosts, the current state of the attacker, and exploits that can be used by the attacker. Exploits are defined by preconditions (source machine access level, target access level, connectivity, and vulnerability required) and post-conditions (effect on target machine). An example is provided for a 4-node network. It demonstrates how modeling information is encoded for the model checker by hand and how it produces an example attack path.

**Ritchey, R., B. O’Berry, et al. (2002), [19]** “Representing TCP/IP Connectivity for Topological Analysis of Network Security,” *Proceedings of the 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada: <http://www.isse.gmu.edu/~csis/publications/acsac02.pdf>.

This paper provides details on how connectivity is modeled in the TVA system described in [4]. The model includes details on different layers of the OSI stack. At the lowest level the model identifies hosts that are on each LAN segment and whether they are connected by switches or hubs to determine the information that an attacker can gather by sniffing all traffic impinging on a compromised host and to determine if ARP spoofing is possible. For a higher network/transport level, the authors suggest modeling each connection to a remote service separately using the program name and version to identify connection endpoints. This approach assumes that port numbers can be ignored after scanning with Nessus [10] to obtain reachability. In practice, port numbers are important because they make it possible to interpret how firewall and routers affect reachability in ways that are not discovered by a Nessus scan. Connections are also modeled to applications. These model, for example, the need to provide a password for user authentication to a particular application. The paper doesn’t describe how all the required details can be obtained accurately. For example, we and others have found that it is difficult to match software versions to vulnerabilities because this information is often inaccurate and missing in vulnerability databases and because software version numbers are often too coarse and not updated after patches are applied that eliminate vulnerabilities.

**Sheyner, O., S. Jha, J. M. Wing, R. P. Lippmann, and J. Haines (2002), [24]** “Automated Generation and Analysis of Attack Graphs,” *2002 IEEE Symposium on Security and Privacy*, Oakland, California: <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/calder/www/sp02.html>.

This paper provides a thorough and detailed example of using a model checker to analyze the security of a small artificial network. A small artificial network with a few vulnerabilities is used to hand-create a finite-state machine that can be analyzed by a model checker. The model checker can determine if a goal state (e.g., administrator on a machine or administrator using stealthy attacks not visible to an IDS) can be reached and it can provide the paths used to reach the goal state. This was demonstrated using an artificial 3-host network with 5 vulnerabilities. The run time was 5 seconds for this network, but it increased dramatically to 3 hours when the number of hosts was increased to only 5 and the number of vulnerabilities was increased to 8. This approach scales poorly, and it is difficult to create inputs for the

model checker and interpret the outputs. It is not clear whether model checkers are practical for networks with thousands of hosts and hundreds of unique vulnerabilities where the state space is orders of magnitude larger than in the artificial example.

**Schneier, B. (1999), [13]** “Attack Trees,” *Dr. Dobbs Journal*, December, 1999.

This often-cited paper contains one of the first public descriptions of a manual approach to generating attack graphs. Each graph has one goal node, and nodes below this represent actions that can reach this goal. Actions combine using either OR (disjunctive) or AND (conjunctive) logic. Values can be assigned to action nodes that indicate if they are possible, if they require special equipment, the cost of the action, the likelihood of the action, and the probability of success. These values can be propagated to the goal state using the OR and AND nodes to compute the characteristics of paths from different starting actions to the goal state. Graphs can be applied in many fields, and experts in each field must generate them by hand.

**Skybox (2004), [2]** Skybox Security: <http://www.skyboxsecurity.com>.

Skybox Security has developed a software tool called Skybox View that generates attack graphs. Attack graphs are used to identify critical vulnerabilities that should be patched first to reduce risk. Input information required includes the attacker source location and target and the loss associated with compromising the target. Risk is calculated as the probability of success of an attack path times the loss associated with the compromised target. This requires information on the probability of success-of-attack components. Skybox creates its own database of vulnerabilities, and users must install special monitoring and aggregation hosts at their sites to collect information required to generate attack graphs. Although no technical details are provided concerning attack graph construction, a related patent [29] suggests that the computation to compute the attack graph between a single source and destination grows roughly as  $N^3$ , where  $N$  is the number of hosts in the network. Examining paths from the attacker to all hosts in the network would thus grow as  $N^4$ . This approach may thus have difficulty scaling to large networks.

**Swiler, L. P., C. Phillips, D. Ellis, and S. Chakerian (2001), [25]** “Computer-Attack Graph Generation Tool,” *Proceedings of the Second DARPA Information Survivability Conference and Exposition (DISCEX II) 2001*, Los Alamitos, California, pp. 307–321, IEEE Computer Society:  
[http://ieeexplore.ieee.org/xpl/abs\\_free.jsp?arNumber=932182](http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=932182).

This paper describes a proof-of-concept tool that was the most capable attack graph generation tool when it was first published. It builds attack graphs to determine the shortest path(s) to a specified goal. Inputs required include hand-generated vulnerability information (pre- and post-conditions), network

information, and attacker capabilities. Vulnerabilities are assigned costs that are used to find the shortest path and also paths that are epsilon longer than the shortest path. Attack paths are built forward from a start node. A trimmed full graph is built that stops adding extra nodes from any nodes that reach the goal state. Further analyses, including the shortest path analysis, are performed on the attack graph with a specified start and end node or goal state. The final result is a graph that can be presented to a system administrator. Data is entered by hand into files read by the tool and stored in an intermediate database. Missing values in network configuration information can be set to default values or to “unknown.” If an “unknown” value appears on a critical path (it was set to a default value to create the graph), this can be indicated, and the user can gather more information on this value. The graph is collapsed to eliminate paths that represent the same sequence of vulnerabilities but exploited in different orders. This helps reduce the number of nodes, but the number of nodes can still grow combinatorically as the number of vulnerabilities and hosts. A simple example generated by the tool includes only 5 vulnerabilities and 2 hosts. This paper has many limitations, including poor scaling and the need to manually enter information required to generate attack graphs. It also introduces a few new ideas. The following are two of these new ideas that can simplify attack-graph generation:

- **Group similar hosts**

It is suggested that hosts on the same LAN that have the same configuration and vulnerabilities should be grouped into single representative aggregate hosts. This reduces the number of hosts that need to be analyzed in an attack graph. This grouping, however, would be incorrect unless the hosts also have the same reachability to and from other hosts in the network.

- **Default values**

It is also suggested that default values should be provided for unknown details about the network. It is difficult to obtain all network and host information, and default values enable an analysis when complete information is not available. It is also suggested that a default value can be used but tagged as being originally “unknown.” If this value enables an attack of a critical network resource, then this can be flagged so the user can gather more information and verify the missing value.

**Templeton, S. and K. Levitt (2001), [5] “A Requires/Provides Model for Computer Attacks,”** *Proceedings of the 2000 Workshop on New Security Paradigms*, New York: ACM Press.

This is one of the first papers to outline how attack scenarios can be generated automatically by linking multiple attacker actions and subgoals. Subgoals in this approach are called concepts. Concepts have requirements, and when these are satisfied, the concept provides capabilities that other concepts can use. The linkage between requirements and provided capabilities can form multistage attack scenarios from multiple concepts. For example, requirements for a telnet connection, might include (1) network access is available to a specific host and port, (2) the host is active, (3) the telnet service is running on the host and (4) a valid username and password are available for that host. Requirements also include details of

software and hardware versions required to support attack steps. An example of a concept description for remote shell connection spoofing is presented written in a language called JIGSAW. It is labor intensive to describe concepts in JIGSAW because detailed low-level information is required to describe both requirements and capabilities. A proof-of-concept program was tried on a subset of the JIGSAW model that requires as input a final goal, but no complex scenarios produced by this program are presented. It is suggested that this approach could be used to discover new attack scenarios and correlate alerts from intrusion detection systems. The main weakness of this work is that excessive hand labor is required to generate detailed requirement and capability models, and the problems of scalable automation were not addressed.

**Tidwell, T., R. Larson, K. Fitch, and J. Hale (2001), [26]** “Modeling Internet Attacks,” *Proceedings of the Second Annual IEEE SMC Information Assurance Workshop*, United States Military Academy, West Point, New York, June 2001: IEEE Press, pp. 54–59.

This paper describes a system that could generate attack graphs to assess network security but does not state whether such a system has been constructed. Attack graphs are of the type described by [13] with one attacker goal. Most of the paper focuses on outlining how Backus-Naur Form (BNF) grammars can be used to specify pre- and post-conditions for attack components and also how BNF grammars can be used to describe characteristics of network components. A simple artificial example with 6 hosts is presented to illustrate how BNF grammars can describe network components and attack steps and be used by hand to produce an attack graph. The paper states that “the attack graph is constructed in a top-down fashion by chaining attack templates that match vulnerabilities found within the active network specification.” Although no timing or scaling results are presented and the authors do not state that any system to construct attack graphs was actually developed, this simple approach will not scale well with large networks because it will create a full attack graph that shows all possible paths to reach a final goal.

## 4. REFERENCES

- [1] Amenaza, "Secur/Tree Attack Tree Modeling," Amenaza Technologies Limited, 2004.
- [2] Skybox, "Skybox Security," 2004.
- [3] M. Artz, NETspa, *A Network Security Planning Architecture*, M.S. Thesis, Cambridge: Massachusetts Institute of Technology, May 2002.
- [4] S. Jajodia, S. Noel, and B. O'Berry, "Topological Analysis of Network Attack Vulnerability," *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds., Dordrecht, Netherlands: Kluwer Academic Publisher, 2003.
- [5] S. Templeton and K. Levitt, "A Requires/Provides Model for Computer Attacks," *Proceedings of the 2000 Workshop on New Security Paradigms*, New York: ACM Press, 2001.
- [6] F. Cuppens and R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks," *Recent Advances in Intrusion Detection (RAID) 2000, Lecture Notes in Computer Science 1907*, H. Debar, L. Me, and F. Wu, Eds., Berlin: Springer Verlag, 2001.
- [7] F. Cuppens "Alert Correlation in a Cooperative Intrusion Detection Framework," *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC, IEEE Computer Society, 2002.
- [8] P. Ning and D. Xu, "Learning attack strategies from intrusion alerts," *Proceedings of the 10<sup>th</sup> ACM Conference on Computer and Communications Security*, New York: ACM Press, 2003, 200–209.
- [9] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," *Proceedings of the 9<sup>th</sup> ACM Conference on Computer and Communications Security*, New York: ACM Press, 2002, 217–224.
- [10] Nessus, "Nessus Security Scanner," 2004.
- [11] P. Mell and T. Grance, "ICAT Metabase CVE Vulnerability Search Engine," National Institute of Standards and Technology, 2002.
- [12] D. Bilar, "Quantitative Risk Analysis of Computer Networks," *Thayer School of Engineering*, Hanover, New Hampshire: Dartmouth College, 2003.
- [13] B. Schneier, "Attack Trees," *Dr. Dobbs Journal*, December, 1999.

- [14] S. Cheung, U. Lindqvist, and M. Fong, "Modeling Multistep Cyber Attacks for Scenario Recognition," *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, vol. 1, IEEE, 2003, 284-292.
- [15] J. Dawkins and J. Hale, "A Systematic Approach to Multi-Stage Network Attack Analysis," *Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)*, IEEE Computer Society, 2004.
- [16] V. Gorodetski and I. Kottenko, "Attacks against Computer Network: Formal Grammar-based Framework and Simulation Tool," *Lecture Notes in Computer Science, vol. 2516.:Recent Advances in Intrusion Detection (RAID) 2002*, A. Wespi, G. Vigna, and L. Deri, Eds., Berlin: Springer Verlag, 2002.
- [17] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs," *Proceedings of the 19<sup>th</sup> Annual Computer Security Applications Conference*, Las Vegas, Nevada, 2003.
- [18] S. Noel and S. Jajodia, "Managing Attack Graph Complexity Through Visual Hierarchical Aggregation," *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, New York: ACM Press, 2004.
- [19] R. Ritchey, B. O'Berry, and S. Noel, "Representing TCP/IP Connectivity for Topological Analysis of Network Security," *Proceedings of the 18<sup>th</sup> Annual Computer Security Applications Conference*, Las Vegas, Nevada, 2002.
- [20] J. P. McDermott, "Attack Net Penetration Testing," *Proceedings of the 2000 Workshop on New Security Paradigms*. New York: ACM Press, 2001, pp.15–21.
- [21] A. Moore, R. Ellison, and R. Linger, "Attack Modeling for Information Security and Survivability," Software Engineering Institute, Technical Note CMU/SEI-2001-TN-01, March 2001.
- [22] R. Ortalo, Y. Deswarte, and M. Kaaniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security," *IEEE Transactions on Software Engineering*, vol. 25, pp. 633–650, 1999.
- [23] R. Ritchey and P. Amman, "Using Model Checking to Analyze Network Vulnerabilities," *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pp. 156-165, 2000.
- [24] O. Sheyner, S. Jha, J. M. Wing, R. P. Lippmann, and J. Haines, "Automated Generation and Analysis of Attack Graphs," in *2002 IEEE Symposium on Security and Privacy*. Oakland, California, 2002.

- [25] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-Attack Graph Generation Tool," *Proceedings of the Second DARPA Information Survivability Conference & Exposition (DISCEX II)*, Los Alamitos, California, vol. II, pp. 307-321, IEEE Computer Society, 2001.
- [26] T. Tidwell, R. Larson, K. Fitch, and J. Hale, "Modeling Internet Attacks," *Proceedings of the Second Annual IEEE SMC Information Assurance Workshop*, United States Military Academy, West Point, New York, June 2001: IEEE Press, 2001, pp. 54–59.
- [27] A. Wool, "A Quantitative Study of Firewall Configuration Errors," *IEEE Computer*, vol. 37, pp. 62–67, 2004.
- [28] D. Turner, S. Entwisle, O. Friedrichs, D. Hanson, M. Fossi, D. Ahmad, S. Gordon, P. Szor, E. Chien, F. Perriot, and P. Ferrie, "Symantec Internet Security Threat Report, Trends for January 1, 2004–June 30, 2004," vol. VI, September 2004.
- [29] G. Cohen, M. Meiseles, and E. Reshef, "System and Method for Risk Detection and Analysis in a Computer Network," USA: Skybox Security Ltd., 2004.



# REPORT DOCUMENTATION PAGE

*Form Approved*  
*OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

|   |                                    |   |   |  |  |
|---|------------------------------------|---|---|--|--|
| <b>1. REPORT DATE (DD-MM-YYYY)</b><br>31 March 2005   |                                    | <b>2. REPORT TYPE</b><br>Project Report |   | <b>3. DATES COVERED (From - To)</b>                        |  |
| <b>4. TITLE AND SUBTITLE</b><br>An Annotated Review of Past Papers on Attack Graphs   |                                    |   |   | <b>5a. CONTRACT NUMBER</b><br>F19628-00-C-0002             |  |
|   |                                    |   |   | <b>5b. GRANT NUMBER</b>                                    |  |
|   |                                    |   |   | <b>5c. PROGRAM ELEMENT NUMBER</b>                          |  |
| <b>6. AUTHOR(S)</b><br>R.P. Lippmann<br>K.W. Ingols   |                                    |   |   | <b>5d. PROJECT NUMBER</b>                                  |  |
|   |                                    |   |   | <b>5e. TASK NUMBER</b>                                     |  |
|   |                                    |   |   | <b>5f. WORK UNIT NUMBER</b>                                |  |
| <b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b><br><br>MIT Lincoln Laboratory<br>244 Wood Street<br>Lexington, MA 02420-9108  |                                    |   |   | <b>8. PERFORMING ORGANIZATION REPORT NUMBER</b><br>PR-IA-1 |  |
| <b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b><br>Department of the Air Force,<br>Robert LeBlanc, Lighthouse Cyber Security Program<br>US Air Force HQ CPSG/NIS<br>230 Hall Blvd., Suite 218<br>Lackland AFB, TX 78243-7056   |                                    |   |   | <b>10. SPONSOR/MONITOR'S ACRONYM(S)</b>                    |  |
|   |                                    |   |   | <b>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</b>              |  |
| <b>12. DISTRIBUTION / AVAILABILITY STATEMENT</b><br><br>Approved for public release; distribution is unlimited.   |                                    |   |   |  |  |
| <b>13. SUPPLEMENTARY NOTES</b>  |                                    |   |   |  |  |
| <b>14. ABSTRACT</b><br>This report reviews past research papers that describe how to construct attack graphs, how to use them to improve security of computer networks, and how to use them to analyze alerts from intrusion detection systems. Two commercial systems are described [1, 2], and a summary table compares important characteristics of past research studies. For each study, information is provided on the number of attacker goals, how graphs are constructed, sizes of networks analyzed, how well the approach scales to larger networks, and the general approach. Although research has made significant progress in the past few years, no system has analyzed networks with more than 20 hosts, and computation for most approaches scales poorly and would be impractical for networks with more than even a few hundred hosts. Current approaches also are limited because many require extensive and difficult-to-obtain details on attacks, many assume that host-to-host reachability information between all hosts is already available, and many produce an attack graph but do not automatically generate recommendations from that graph. Researchers have suggested promising approaches to alleviate some of these limitations, including grouping hosts to improve scaling, using worst-case default values for unknown attack details, and symbolically analyzing attack graphs to generate recommendations that improve security for critical hosts. Future research should explore these and other approaches to develop attack graph construction and analysis algorithms that can be applied to large enterprise networks. |                                    |   |   |  |  |
| <b>15. SUBJECT TERMS</b>  |                                    |   |   |  |  |
| <b>16. SECURITY CLASSIFICATION OF:</b>  |                                    |   | <b>17. LIMITATION OF ABSTRACT</b><br><br>Same as report | <b>18. NUMBER OF PAGES</b><br><br>35                       | <b>19a. NAME OF RESPONSIBLE PERSON</b>           |
| <b>a. REPORT</b><br>Unclassified  | <b>b. ABSTRACT</b><br>Unclassified | <b>c. THIS PAGE</b><br>Unclassified     |   |  | <b>19b. TELEPHONE NUMBER (include area code)</b> |