A Graph-Based System for Network-Vulnerability Analysis

Laura Painton Swiler and Cynthia Phillips
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

This paper presents a graph-based approach to network vulnerability analysis. The method is flexible, allowing analysis of attacks from both outside and inside the network. It can analyze risks to a specific network asset, or examine the universe of possible consequences following a successful attack. The graph-based tool can identify the set of attack paths that have a high probability of success (or a low "effort" cost) for the attacker. The system could be used to test the effectiveness of making configuration changes, implementing an intrusion detection system, etc.

The analysis system requires as input a database of common attacks, broken into atomic steps, specific network configuration and topology information, and an attacker profile. The attack information is "matched" with the network configuration information and an attacker profile to create a superset attack graph. Nodes identify a stage of attack, for example the class of machines the attacker has accessed and the user privilege level he or she has compromised. The arcs in the attack graph represent attacks or stages of attacks. By assigning probabilities of success on the arcs or costs representing level-of-effort for the attacker, various graph algorithms such as shortest-path algorithms can identify the attack paths with the highest probability of success.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

DISCLAIMER

1. Introduction

Military, government, commercial, and civilian operations all depend upon the security and availability of computer systems and networks. In October 1997, the Presidential Commission on Critical Infrastructure recommended increasing spending to a $1B level during the next seven years. The Commission recommended that this money be heavily focused on cyber-security research, including vulnerability assessment, risk management, intrusion detection, and information assurance technologies (Commission Report, Oct. 1997). In this paper, we describe a systematic analysis approach that can be used by persons with limited expertise in risk assessment, vulnerability analysis, and computer security to (1) examine how an adversary might be able to exploit identified weaknesses in order to perform undesirable activities, and (2) assess the universe of undesirable activities that an adversary could accomplish given that they were able to enter the network using an identified weakness.

Ideally, a network-vulnerability risk-analysis system should be able to model the dynamic aspects of the network (*e.g.*, virtual topology changing), multiple levels of attacker ability, dynamic behavior of a single attacker (*e.g.*, learning), multiple simultaneous events or multiple attacks, user access controls, and time-dependent, ordered sequences of attacks. Intrusion-detection systems have attempted to monitor abnormal patterns of system usage (such as suspicious configuration information changes) to detect security violations (Denning, 1985; Lunt, 1993). Our system would be complementary to an intrusion detection system. If an administrator does not want to pay the full cost (development cost or system-performance hit) of all possible intrusion-detection strategies, our system could suggest cost-effective subsets which focus on the most vulnerable system components.

Probabilistic Risk Assessment (PRA) techniques such as fault-tree and event-tree analysis provide systematic methods for examining how individual faults can either propagate into or be exploited to cause unwanted effects on systems. For example, in a *fault-tree* a negative consequence, such as the compromise of a file server, is the root of the tree. Each possible event that can lead *directly* to this compromise (e.g., an attacker gaining root privileges on the machine) becomes a child of the root. Similarly, each child is broken into a complete list of all events which can directly lead to it and so on. Wyss, Schriner, and Gaylor (Wyss et. al) have used PRA techniques to investigate network performance. Their fault tree modeled a loss of network connectivity, specifically the "all terminal connectivity" problem. Physical security and vital-area analyses have also successfully used PRA techniques (Stack and Hill, 1984). Since PRA methods can measure the importance of particular components to overall risk, it seems that they could provide insights for the design of networks more inherently resistant to known attack methods. These methods, however, have limited effectiveness in the analysis of computer networks because they cannot model multiple attacker attempts, time dependencies, or access controls. In addition, fault trees don't model cycles (such as an attacker starting at one machine, hopping to two others, returning to

the original host, and starting in another direction at a higher privilege level). Methods such as influence diagrams and event trees suffer from the same limitations as fault trees.

The major advance of our method over other computer-security-risk methods is that it considers the physical network topology in conjunction with the set of attacks. Thus, it goes beyond the scanning tools such as the SATAN (Security Administrator Tool for Analyzing Networks) tool that are currently available which check a "laundry list" of services or conditions that are enabled on a particular machine. For example, SATAN checks for the following vulnerabilities on UNIX based systems:

1. Are NFS file systems exported to unprivileged programs?
2. Are NFS file systems exported to arbitrary hosts?
3. Is X server access control disabled?
4. Is there a writable anonymous FTP home directory?
5. Is there an insecure version of sendmail in use?

...

but gives no indication of how these items lead to system compromise. All the vulnerabilities SATAN finds are well known and have either bulletins and/or patches from an incident response team or a vendor. SATAN is a useful network analysis tool and can provide a system administrator with a set of items to patch or fix. However, it cannot identify paths of attacks, alternative network configurations that would be more robust, or linked attacks such that a combined sequence of attacks would do more harm than an individual attack and it doesn't help the system administrator set security priorities.

Our approach to modeling network risks is based on an *attack graph*. Each node in the graph represents a possible attack state. A node will usually be some combination of physical machine(s), user access level, and effects of the attack so far, such as placement of trojan horses or modification of access control. Edges represent a change of state caused by a single action taken by the attacker (including normal user transitions if they have gained access to a normal user's account) or actions taken by an unwitting assistant (such as the execution of a trojan horse). Attack graphs will be presented in more detail in Sections 2 and 3.

The attack graph is automatically generated given three types of input: attack templates, a configuration file, and an attacker profile. *Attack templates* represent generic (known or hypothesized) attacks including conditions, such as operating system version, which must hold for the attack to be possible. The *configuration file* gives detailed information about the specific system to be analyzed including the topology of the network and configuration of particular network elements such as workstations, printers, or routers. The *attacker profile* contains information about the assumed attacker's capabilities, such as the possession of an automated toolkit or a sniffer as well as skill level. The attack graph is a customization of the generic attack templates to the attacker profile and the

network specified in the configuration file. Though attack templates represent pieces of known attacks or hypothesized methods of moving from one state to another, their combinations can lead to descriptions of new attacks. That is, any path in the attack graph represents an attack, though it could be cobbled together from many known attacks.

Each edge has a weight representing a success probability or a cost to an attacker (edges with zero probability are generally omitted). This weight is a function of configuration and attacker profile. Furthermore, each node can have local "overwrites" of these files representing effects of previous attacker actions on configuration (e.g. severed network connections, or changes to file-access privileges) or acquired attacker knowledge (learning). In Section 2 we discuss possible ways to estimate edge weights.

A short path in the attack graph represents a low-cost attack. Since edge weights will only be estimates, we consider the set of all near-optimal paths. If the edge weights are reasonably accurate, this set as a group represents the most vulnerable parts of the network. If one can assume independence of success probabilities, the same (shortest-path) algorithms can find paths with high success probability. By having multiple weights on each edge, one can represent potentially-conflicting criteria (e.g. the attacker wishes to minimize both cost and probability of detection).

This system can answer "what-if" questions regarding security effects of configuration changes such as topology changes or installation of intrusion-detection systems. It can indicate which attacks are possible only from highly-skilled well-funded attackers, and which can be achieved with lower levels of effort. A business owner might decide it is acceptable to allow a relatively high probability of network penetration by a "national-scale" effort, but will tolerate only a small probability of attack from an "average" attacker. Government sites, which are attacked with much higher frequency[1], may need exceptionally low probability of success for a particular attacker level in order to expect few penetrations, and they may be more willing to pay the cost for that level of security.

Finally, this system can simulate dynamic attacks and use the results to test intrusion-detection systems. These analysis methods, as well as possible ways to calculate cost-effective defense strategies, are explained in more detail in Section 4.

The remainder of the paper is organized as follows. Section 2 gives a more detailed description of attack templates, the configuration file, and attacker profile. Section 3 discusses attack-graph generation. Section 4 presents analysis methods. Section 5 provides some concluding remarks. Appendix A lists some implementation details associated with generating the attack graph. Appendix B gives a detailed example applied to a test network we have built.

[1] The Defense Information Systems Agency reports that the Department of Defense is attacked 250,000 times a year. Los Alamos National Laboratories is attacked daily, with 22 proven outsider intrusions in the last five months. From "Security Measures," Albuquerque Journal, March 24, 1998, pp. B1-B2.

2. Configuration Files, Attacker Profiles, and Attack Templates

This section explains the inputs required for our method: configuration files, attacker profiles, and attack templates.

Configuration files

The configuration file contains information relevant to operating system, network type, router configuration, and network topology. More specifically, each **device** (i.e., workstation, printer, file server, etc.) should have the following information:

1. **Machine class**: workstation, printer, router, etc.
2. **Hardware type**: e.g., SUN SPARCstation™ 5
3. **Operating System**

a. O.S. patches that have been installed.

4. **Users** (Initially just the classes of users, i.e. root, normal, privileged.)
5. **Configuration**

a. Ports enabled

b. Services enabled

c. Any intrusion detection applications installed

4. **Type of network**(s) the device is on (Ethernet, FDDI, ATM, etc.)
5. **Physical link** information such as type of communications media

A configuration file also includes a graph of the topology of the network. Building and maintaining configuration files by hand will be a tedious, time-consuming and error-prone task which could seriously limit the utility of the system. Therefore, we envision an automated tool to generate and maintain this configuration file. For example, a root-level daemon on each network component can periodically send information to a central server. The configuration file could be based upon the information available from a tool like SATAN, augmented to match the conditions in the set of attack templates. We hope the system administrator will have reasonable defenses in place to protect this data when using the tool. For example, it may only be available online in one place while the administrator is running analyses.

Attacker Profiles

The attacker profile contains information about an assumed attacker's capabilities, such as the possession of an automated toolkit, a sniffer, etc. The attacker profile also contains an assumption about the skill level of an attacker, which is used to determine the probability of success for particular attack methods. The attacker profile represents the initial capabilities of the attacker in the same way that the configuration file represents the initial state of the network. To assist the analyst, default profiles for various attacker skill levels such as novice vs. expert could be provided. The network owner's security policies and strategies can be guided by the level of attacker they wish to strongly deter and their available budget.

Attack template

Attack templates represent generic steps in known attacks, including conditions which must hold for the attack to be possible. Each node in the attack template represents a state of an attack, as detailed below. The nodes are distinguishable, and therefore, each edge represents a change in state on one or more devices. Examples of state changes are: a file was changed, a configuration setting was altered, an executable was run, an attacker gains root privileges on a machine, etc. An example of attack templates using the following definitions and fields is shown in Figure 1. A more detailed attack graph of password guessing is presented in Appendix B.

Nodes have the following fields:

1. **User level:** Possible user levels include: none, guest (anonymous), normal user, privileged user, root, or system administrator.
2. **Machine(s):** This field could specify an individual machine or set of machines, all machines on a subnet, or all machines on multiple subnets. In the attack templates, this field contains placeholders (variables) that are instantiated in the attack graph.
3. **Vulnerabilities:** This field indicates changes to the original configuration caused by attacker actions. When building the attack graph, the vulnerabilities "overwrite" the relevant portions of the configuration file for a given node.
4. **Capabilities:** This field locally overwrites the attacker profile in the same way the vulnerabilities field overwrites the configuration file. Possible entries include physical access to part of the network, installation of a trojan horse, delivery of mail or an applet with executable content, or installation of a sniffer on an edge of the network. It can also indicate other programs that the attacker has successfully installed or has access to, such as crack programs, root kits, etc.
5. **State:** The state field breaks attacks into atomic pieces. An attack may require several steps, each of which could fail and none of which adds a new capability, vulnerability, etc. The states distinguish the nodes by indicating progress in the attack.

Edges in the attack template **represent actions** taken by the attacker or his/her victim/unwitting assistant. They can also indicate an event such as the detection of a particular type of packet on a network by some hardware and/or software under attacker control. To allow maximum detection of new attack sequences, these events should be atomic and nontrivial (probability of success is strictly above 0). Probability-one edges must change the environment (introduce a vulnerability, change user level, etc.). Each edge has conditions on the users and/or machines. If all the conditions are met, the attack succeeds with a given probability and/or cost. Our examples model this measure as static, but it can be a function of configuration and attacker capability. If a user is only interested in viewing the possible universe of attacks regardless of cost/success probability, then these functions could be extremely simple. The probability-of-success numbers can be obtained from polling experts (assessing the best subjective judgments), from information about the frequency of attacks on certain kinds of networks (Howard,

1997), and from experimentation. Computer-security personnel can test various attacks. Furthermore, one can make increasingly-automated testbeds accessible from the internet and advertise them as challenges to the computer-security community, then gather statistics about success probability.

A number of issues are not completely resolved. There is some flexibility in assigning conditions to the arcs (requirements for the attack) vs. the nodes (part of the state). For example, possession of a root kit may be required for a certain attack. It can be made a condition of the edge (hence the edge is not added to the attack graph unless the attacker possesses a root kit) or it can be made a state of the start node (thus the attacker must have a root kit in order for the node to be reached in the first place). In addition, one must carefully chose levels of machine aggregation. Generating nodes for all possible subsets of machines will be impossible even for small systems. However, we believe the design described above can model a wide variety of attacks. For example, we have developed a set of templates for several attacks in each of the following classes: sendmail, ftp, telnet, Windows NT, and Java. Furthermore, the system has sufficient flexibility to evolve smoothly as new, previously unanticipated modeling needs arise.

3. Generating the Attack Graph

In this section we describe how one might generate the attack graph from a configuration file, an attacker profile, and a database of attack templates. Appendix A discusses implementation issues. In general the nodes of the attack graph look like nodes of the attack templates instantiated with particular users and machines. Edges are labeled only by a probability-of-success (or cost) measure, and a documentation string for the user interface. For ease of exposition, for the remainder of this section, we will call the measure the *weight* of the edge. This weight is determined by an *instantiation function* associated with each edge of an attack template. This function accesses the configuration file and the attacker profile. If an edge goes from node u to node v, then we call node u the *tail* of the edge and node v the *head* of the edge.

We now describe how the attack graph could be generated by building backwards from a goal node. One could also build forward from a start node (to explore the universe of possibilities) or assume both a start and a goal node. We illustrate this description with the simple example in Figure 2. The attacker profile, which is not shown in Figure 2 for space reasons, assumes that the attacker has physical access to B and the boot CD. We maintain a queue of generated nodes which have not been processed. Initially this queue contains only the goal node and nodes are added as they are created.

Start with the goal node: achievement of user-level access on machine M. The graph generator checks the database of attack templates and identifies all edges whose heads match the goal node. Assuming this database contains only the two templates shown in Figure 2, we find two matches, namely the head of each attack template. Consider the first template for an rlogin attack. Machine M matches the variable M_2 in the template.

The instantiation function can then generate the tail node (node N_1) by generating all (user, machine) pairs that meet the constraints (the user has an account on this machine and M, and an appropriate rlogin file on M). Note that if machine M has rlogin disabled, then node N_1 would not be generated. On the assumption that machines A and B can communicate with M (given the rlogin file), the probability of the edge from node N_1 to the goal is 1. Node N_1 is an OR node, meaning that achievement of any (user, machine) pair suffices.

The goal node also matches the last node of the second template for physical access. Machine M matches the variable X and the instantiation function creates node N_4, which in turn generates N_5. However, the attacker does not have physical access to M. Thus, the nodes N_4 and N_5 are marked with a dotted line to show that under existing conditions, they would not be reachable from the start state. There could be other attack templates which would lead to physical access to M, and then these nodes would be enabled. In this case, the capability of physical access to M is an addition (or overwrite) to the attacker profile.

Since there are no more matches for the goal, node N_1 is removed from the queue and matched against the database against both heads and tails. In principle, it can again match with the head of the rlogin attack. However, assuming transitivity (i.e. that a user has rlogin set up symmetrically for all his accounts), applying this edge again will give no new information. Recognizing and preventing this in all cases is still a research issue. Node N_1 also matches with the last node of the second template on physical access, which generates node N_2.

Node N_2 matches the middle node of the second template. The attacker profile indicates that the attacker has physical access to machine B, but not to machine A. Since N_2 is an OR node, it can be satisfied by the attacker becoming root on B. In this example, node N_3 is created with a subset of the machines in node N_2. Alternatively, we could have generated an intermediate node for becoming root only on B rather than A or B. The advantage of this is that additional paths to the goal can pass through this intermediate node (i.e. a path unique to B cannot be built off a node which can be satisfied by either A or B). When both goal and start nodes are specified, either method is likely to work, since if this node is required for a path, it will be generated later. If only one of goal and start are specified, the more verbose method may be advantageous. We recognize node N_3 as a start node in this graph, and thus we do not try to match backwards from it. Although it is not shown, the attack graph would also contain a node for A similar to N_3 which, like nodes N_4 and N_5, is unreachable because the attacker has no physical access to A.

When a node is matched with a template in the database, the other endpoint could either be generated as in the example above, or be a node already generated. Thus the generator must be able to efficiently search the nodes generated so far. Edges created between two nodes already generated can lead to interactions between attack templates and the "discovery" of new attack sequences.

There are a number of implementation issues which must be resolved when the system is tested on large datasets. These issues are presented in Appendix A for interested readers. Appendix B presents a more detailed graph generation example, specifically a password guessing attack on a small network. This example is more comprehensive and more realistic, but is omitted from the main part of the paper for space reasons.

4. Analysis Methods

In this section we discuss analysis of the attack graph: determining a (set of) low-cost attack paths, finding a set of cost-effective defenses, and simulating dynamic attacks. Each edge in the attack graph has a weight, such as cost to the attacker. A path from a start node to a goal node has a weight equal to the sum of the weights of the edges in the path. In the case where weights represent success probabilities rather than costs, we can convert to a problem of this form. By replacing each weight by its logarithm, the weight of the path (sum) now represents the product of the probabilities, and we wish to find highest-cost paths. Because the probabilities are all between 0 and 1, the logs are all non-positive numbers. Therefore, if we negate all the probabilities (i.e., multiply by –1), all weights become non-negative and the problem is converted from maximization to a minimization problem, that of finding the low-cost paths. The structure of the weights is critical for this conversion, because in general finding the longest paths in a network is NP-complete (Garey and Johnson, 1979).

If one wishes to find only a single shortest path, representing the most likely or least-cost attack, from a start node to any number of goal nodes, then any standard shortest-path algorithm, such as Dijkstra's algorithm will suffice. Such codes are very efficient (near linear-time) and readily available.

However, the weights on the edges will almost surely not be sufficiently accurate to merit looking only at shortest paths. A better method is to use the technique of Naor and Brutlag (1993). Their algorithm computes a compact representation of all paths that are within δ of optimal for some given error parameter δ (the δ-optimal paths). For example, edges that are common to many δ-optimal paths are likely to represent vulnerable points.

If edges have two weights representing different optimization criteria, bicriteria shortest-path algorithms compute a set of paths that are (near) optimal with respect to one weight while obeying a bound (e.g. a budget) on the second weight. Current (near) exact solution methods involve shortest-path computations in significantly expanded graphs. However, scaling provides a graceful tradeoff between approximation quality and the time and space needed to compute the solution (Phillips 1993). Very recently, Tayi et al. (Tayi, Rosencrantz, Ravi, 1997) have shown how to compute all undominated (Pareto optimal) paths for multiple edge weights. Their algorithm runs in pseudo-polynomial time provided the number of criteria is bounded (i.e., the exponent in the running time depends on the number of criteria).

Given a set of possible defenses, each with a cost (financial, loss of service, etc.) and defense budget, we would like to compute a set of defenses to implement which will maximally decrease the probability of success (or increase attacker cost). Implementing a defense strategy on a particular machine could have a widespread effect on the attack graph, since it affects the weight on every edge involving that machine and an attack affected by the defense. In its most general form, this problem is NP-hard to approximate to within better than a logarithmic factor (by reduction from set cover). However, it is possible that attack graphs have special structure which makes the problem easier than this worst case.

A reasonable first question is to take the set of paths computed by the Naor-and-Brutlag algorithm and find a set of defenses that increases the cost of each of those paths above some threshold such as the value of the data stored in the system. The Naor-and-Brutlag algorithm also gives the number of δ-optimal paths. Therefore, one can use the following greedy algorithm: for defense d_i, let p_i be the number of paths with length under threshold whose length meets the threshold when defense d_i is added. Let c_i be the cost of defense d_i. Choose the most cost-effective defense (the one which maximizes (p_i / c_i)). Iterate until all paths are over the threshold. Alternatively, one can modify exact set-cover algorithms for this problem. Because one can model airline crew scheduling as a set-cover problem, there has been extensive work in (near) exact methods for this problem.

The unweighted version of this defense problem can model the placement of monitors for intrusion detection. The question becomes: choose a minimum number of monitor placements such that all the near-optimal attack paths are monitored at least k times. That is, any attempt to execute any of the attacks will potentially be observed by k (possibly nondisjoint) monitors. If monitoring of each edge or node in the attack graph were independent (i.e. we must pay for each monitor placed on any edge), we have the k-hurdle problem, which can be solved efficiently (Burch et. Al, 1998). When sets of edges are affected by a single monitor placement, the problem is still theoretically as hard as set cover (assuming no special structure). However, it will be easier than the weighted version in practice.

Even in the absence of automated defense-selection tools, however, the system can serve as a defense-selection tool. A network administrator can change the configuration file to reflect the placement of a set of defenses, and then run the shortest-paths analysis to determine their effect. Using global search techniques, this iterative procedure could be automated as well.

Alternatively, a system administrator could use the attack graph as the foundation for a simulation tool. The simulation could start from the node where the attacker breaks in or begins. The attacker could pick an edge (representing an attack), have the simulation "flip a coin" to see if the path is successful according to the edge probability, and if successful, the attacker continues down the path, otherwise, she backtracks. This kind of a model

could represent the real behavior of attackers (going down one branch, figuring that it is too difficult to do something such as get root on a particular machine, so backing up and trying another method). Another strategy would be that the attacker chooses his next attack edge based on configuration knowledge of all outgoing links, plus an estimate of the shortest path from neighboring nodes. The success probabilities used in the simulation can change dynamically to reflect the success/failure the attacker has had so far (i.e. as the attacker learns more about the particular system). This simulation technique would be appropriate for a graphical user interface which could show a network designer the paths the attacker is most likely to take (for example, by lighting up nodes with a green light as the attacker is successful, and displaying a red light where the attacker gets blocked).

5. Conclusions

We have spoken with computer security experts, and general consensus is that an attack-graph analysis should work well for modeling enterprise-level (commercial or military) network risks. We would like to take this work further and develop a robust tool with a graphical interface which is easy to use and which links to a large list of vulnerabilities, such as the databases that commercial vendors (i.e., Internet Security Systems' X-force database) have created or that CERT has compiled.

This paper has presented a method for risk analysis of computer networks. The method is based on the idea of an attack graph which represents attack states and the transitions between them. The attack graph can be used to identify attack paths that are most likely to succeed, or to simulate various attacks. The attack graph could also be used to identify undesirable activities an attacker could perform once they entered the network. The major advance of this method over other computer security risk methods is that it considers the physical network topology in conjunction with the set of attacks. Thus, it goes beyond the scanning tools that are currently available which check a "laundry list" of services or conditions that are enabled on a particular machine.

The method we have presented addresses many of the modeling issues that a traditional PRA method such as fault trees do not. Specifically, our graph-based approach allows for modeling dynamic aspects of the network (this can be done by overwriting the configuration file as the attacker makes system changes). Our approach allows for several levels of attacker capability, and can capture the learning behavior of the attacker by adding capabilities to the attacker profile as the graph gets built. It allows for the modeling of user access levels and transitions between them, which are critical in network security. And it represents the time dependencies in sequences of attacks. We would like to examine the possibility of using the attack graph approach, especially the idea of attack templates, for testing intrusion detection systems. The attack graph could also be the basis for identifying the most cost-effective set and placement of defenses.

There are potential limitations with our method. We have not generated a realistic size attack graph based on 10 or 20 templates, and we have not resolved all of the issues associated with the matching of templates to configuration and attacker profile. Also, the existence of attack templates and of the configuration file could be another vulnerability in itself. If these got into the wrong hands, they would be very valuable tools for the attacker. However, we believe that the approach we have presented is an advance in network-vulnerability modeling and will ultimately help network security if implemented in a reasonable way.

Acknowledgments

Timothy Gaylor, formerly at Sandia National Laboratories and currently at 3M, was instrumental in the development of the approach in this paper. The basic notion of an attack graph is due to Fred Cohen of Sandia National Laboratories. The authors also thank Greg Wyss and John Howard of Sandia National Laboratories and Jean Camp at the Kennedy School of Government/Harvard University for helpful and insightful discussions.

References

Cherkassky, B.V., A.V. Goldberg, and T. Radzik. "Shortest Paths Algorithms: Theory and Experimental Evaluation," *Math Programming*, **73**, pp.129--174, 1996. Web site: http://www.neci.nj.nec.com/homepages/avg/soft/soft.html

Burch, C., Krumke, S., Marathe, M., Phillips C., and Sundberg, E. "Multicriteria Approximation Through Decomposition", submitted, 1998.

Denning, D. E. "An Intrusion-Detection Model." *IEEE Transactions on Software Engineering*, **13**(2), 1987.

Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, NY, 1979

Howard, J. D. "An Analysis of Security Incidents on the Internet, 1989-1995." Doctoral dissertation, Carnegie Mellon University, 1997.

Internet Security Systems, Inc. 41 Perimeter Center East, Suite 550, Atlanta, GA 30346. Creator of the X-force database, accessed via http://www.iss.net/xforce.

Lunt, T. F. "A Survey of Intrusion Detection Techniques." *Computers and Security* **12**, pp. 405-418, 1993.

Naor, D. and D. Brutlag, "On suboptimal alignment of biological sequences," *Proceedings of the 4th annual Symposium on Combinatorial Pattern Matching*, Springer Verlag, 1993, pp. 179-196.

Phillips, C. A., "The network inhibition problem," *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, May 16-18, 1993, pp. 776-785.

Presidential Commission on Critical Infrastructure Protection. Commission Report "Critical Foundations: Protecting America's Infrastructures," October 1997. Available at: http://www.pccip.gov/report_index.html

SATAN. (Security Administrator Tool for Analyzing Networks) tool. SATAN's creators, Mr. Dan Farmer and Mr. Wietse Venema, made SATAN widely available over the Internet without cost starting April 5, 1995. It can be obtained from the web site: http://142.3.223.54/~short/SECURITY/satan.html

Stack, D. W., and M. S. Hill. "A SETS User's Manual for Vital Area Analysis," SAND83-0074 and NUREG/CR-3134. Prepared by Sandia National Laboratories for the U.S. Regulatory Commission, Washington D.C., 1984.

Tayi, G., Rosencrantz, D. and S. Ravi. "Path Problems in Networks with Vector Valued Edge Weights." Submitted for publication, October 1997.

Wyss, G. D., Schriner, H. K., and T. R. Gaylor (1996). "Probabilistic Logic Modeling of for Hybrid Network Architectures." Published in the *Proceedings of the 21st IEEE Conference on Local Computer Networks.*

Appendix A: Implementation Issues of Attack Graph Generation

There are a number of implementation issues which must be resolved when the system is tested on large datasets. For example, it may be useful to allow some hierarchy in the attack graph generation. If there is a common set of attack paths that allow an attacker to become root from a normal user account on the same machine, this could be a useful building block. If multiple machines have identical parameters, this subgraph need only be built once. It can be collapsed to one edge, with the option of expanding the graph for the system administrator via the user interface.

For each piece of the configuration or attacker profile files, it would be useful to maintain a list of edges whose probability was influenced by that attribute. This will allow quick recomputation of edge weights if a configuration or attacker parameter is changed. However, it is more challenging to leave such a "trail" for pieces that were missing in the configuration file or lead to edges not existing.

Instantiation functions could become quite complicated. For example, suppose one is searching for the universe of possible consequences from a break-in. In "spam" attacks on networks, an attack is replicated on many machines. If one wants to predict the number of machines compromised, the instantiation function must have an inclusion/ exclusion calculation if the weights are probabilities.

The instantiation function may generate multiple nodes if reachability is a condition on an edge and there are multiple routers between a pair of machines (see the example in Appendix B). The steps necessary for routing a message, telnet session, etc., are explicitly included in the attack graph because this access is an important security parameter. If a worrisome attack path involves going through multiple routers, the system administrator has the option of modifying the access-control tables to forbid the access.

There are two possible ways to represent the users and/or machines in a node: as an explicit list, or as a list of conditions (from edge conditions). Since each condition is associated with an instantiation function, one can go from condition lists to explicit user lists. Both representations could be used in different parts of the attack graph during generation depending upon the ways the lists will be refined. For example, the list-of-conditions method may be better for matching.

Another issue is how to model attacks that require access to two different user accounts possibly on two different machines. This could be done as a 2-step process in the attack template. However, in the attack graph, getting access to two users' accounts is highly correlated within the various attacks, and this correlation must be incorporated into both instantiation functions. Therefore, obtaining access to two or more accounts should probably be combined as a single atomic event. Since we expect most attacks to require

access to only a small number of accounts simultaneously, this consolidation/duplication should not cause overwhelming graph expansion.

Matching methods will evolve experimentally. However, unification techniques used in logic programming languages are a natural starting place. It is possible that using lists of conditions, one can search the set of generated nodes efficiently using hashing techniques.

Appendix B: Password Guessing Example

This Appendix presents an example of the graph-based vulnerability assessment method, specifically a password-guessing attack on a small network. The network, shown in Figure B.1, is small but has a somewhat complex topology and also has many of the main technologies we are interested in modeling: an ATM-switched network, an Ethernet network, two routers of differing types, a firewall to the Internet, SGI workstations, and SUN workstations.

Figure B.2 is an attack template showing several possible ways to gain illegal access to a machine by password guessing. For example, an attacker can use anonymous ftp to plant a trojan horse which when executed mails him back the password file. He then can run a password cracking program on the password file. Or, if the attacker has a sniffer and sniffs the password, if the password is plaintext, the attacker can login as a normal user with that password. As shown in Figure B.2, attack templates are multigraphs. That is, there can be multiple edges between two nodes indicating different attack methods. For example, in Figure B.2, trojan horses can lead to attacker acquisition of the password file in three different ways. We chose password guessing because it is a common attack estimated to be used in approximately one-quarter of attacks, based on the analysis of incidents reported to the Computer Emergency Response Team (CERT), in the dissertation by John Howard, 1997. This example is not meant to be exhaustive even for password guessing. In general an assessment is only as complete as allowed by the coverage of the database.

Attack graphs assume a start and/or goal state. For this example, we assumed that the attacker had access to a normal user account on the Sun workstation SUN1. That is, the attacker could be an insider with an account on SUN1 or could have gained access to SUN1 from the Internet by getting through the firewall. The file server in this network is the Silicon Graphics workstation SGI1 on the Ethernet network. We assumed that the attacker's goal was to access protected data files on the file server SGI1. The starting and goal states are specified in the attacker profile. Only one of these is needed and the attack graph can be built from that point. In this example, however, we specify both.

Figure B.3 shows the attack graph generated from the password-guessing attack template and the network configuration information. This graph shows specific steps the attacker would take to get the protected files. We will not step through the graph generation in detail, but the overall idea is that the user on SUN1 is going to try to access an account on SGI2. From there, she sniffs the password of a user on the broadcast Ethernet network who is logging into SGI1.

This graph was generated as follows: the start node (the attacker having access to a normal user account on SUN1) matches the conditions of the lower start node on the password-guessing template (normal user on a machine M). From the template start node, there are two paths, one involving email and one involving anonymous ftp. The

graph-generation algorithm checks the configuration file to see if email is enabled between SUN1 and SGI1. It is not, because SGI1 is configured to be a protected server which only has privileged users who must logon for access. Likewise, anonymous ftp is turned off on SGI1. However, SGI2 has these services. Thus, the paths of planting a trojan horse via email or obtaining the password file via anonymous ftp are matched to the SGI2 where SGI2 is machine B on the attack template. To access SGI2 via ftp or email, the packets must go through both the NetEdge and Cisco routers. This is information that is in the configuration file. These show up as states in the attack graph because they represent stages necessary to perform the ftp or email actions. (Note: this approach can help show where it will be beneficial to prevent attack. For example, one could configure the routers to not allow any traffic from the ATM network to the Ethernet network).

Note that the start node did not match the upper start state in the password template based on the sniffing route. That is because SUN1 is on an ATM network, which is a switched packet network. It is very difficult to sniff packets on a switched network but relatively easy to do on a broadcast network.

Follow the attack graph to the "normal user on SGI2" node. The intermediate nodes between SUN1 normal user and SGI2 normal user are an instantiation of the password template states, based on our actual test network. Now the graph generation algorithm examines what states on the attack template match "normal user on SGI2." The lower start node matches "normal user on SGI2" but it doesn't match the subsequent nodes because email and ftp are disabled on SGI1. We have assumed in the attacker profile that the attacker has access to a sniffer for broadcast Ethernet networks that requires root capability. These are publicly available; we downloaded one from the web. We have also assumed that the attacker can get root access on SGI2 once she is a normal user on SGI2 (there are a variety of attack templates which could outline how to get from normal user to root on a machine, including use of a toolkit, physical access, etc.). From root on SGI2, the attacker can install the sniffer to listen to the Ethernet traffic. So, the attacker can sniff the password of a privileged user or the system administrator logging into SGI1. With that, she will have access to the files on SGI1.

During the attack-graph generation, each edge is labeled with the probability that the attacker will successfully transition between the two adjoining nodes. Some of the probabilities are based on knowledge of the frequency of events. For example, the probability that a person will click on an email attachment and run it is fairly high. We estimated it at .9. Other probabilities will be based on configuration information and attacker skill level. An edge in the attack template could have several probabilities for different conditions and attacker skill level, and these will be generated by the instantiation function on the edge. For example, the function to generate the probability for successfully sniffing the packet containing the password could be a function of the number of users and the frequency of login for each user over the network. For another example, the configuration file will indicate whether traffic going to M is encrypted or

not. If the traffic is plaintext, then the probability of successfully guessing the password when it is sniffed is 1. If the password is encrypted, then the edge has probability 1 if the attacker possesses the key (as indicated in the attacker profile). Otherwise, it is set to some probability according to the instantiation function (either a probability based on attacker experience or financial ability, or 0 if it is assumed that the profile is complete in regard to key possession). The probabilities we used may not be very representative: more research is needed to obtain more accurate probability estimates. Alternatively, "level of effort" estimates could be used on the arcs.

Finally, we used a shortest-path algorithm to find the path that has the highest probability of success. This path is shown in Figure B.3 by the gray-colored nodes. To obtain this path, we modified a shortest-path code that was publicly available on the web. This code is called SPLIB, version 1.3, December 20, 1996, written by Cherkassky, Goldberg, and Radzik. SPLIB contains codes, generators, and generator inputs for shortest-path algorithms. We used one of the shortest-path algorithms based on the Dijkstra algorithm. The most successful path had a probability of success of 1*0.98*0.95*0.75*0.98*1*0.95 = 0.65.

We built the test network shown in Figure B.1. We found that implementing a test network is a useful tool for understanding attacks, identifying various paths, and getting a sense of the probability of success for various attacks by having different people attempt them.

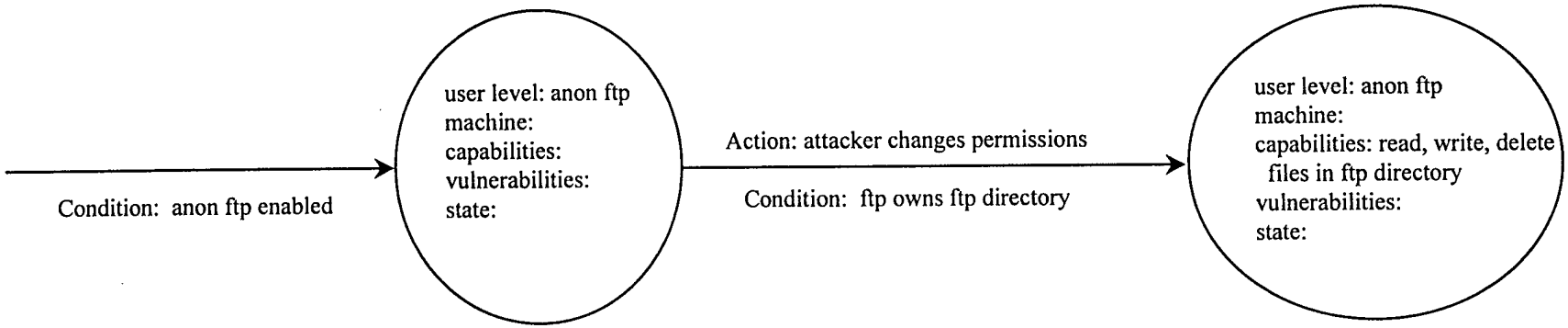
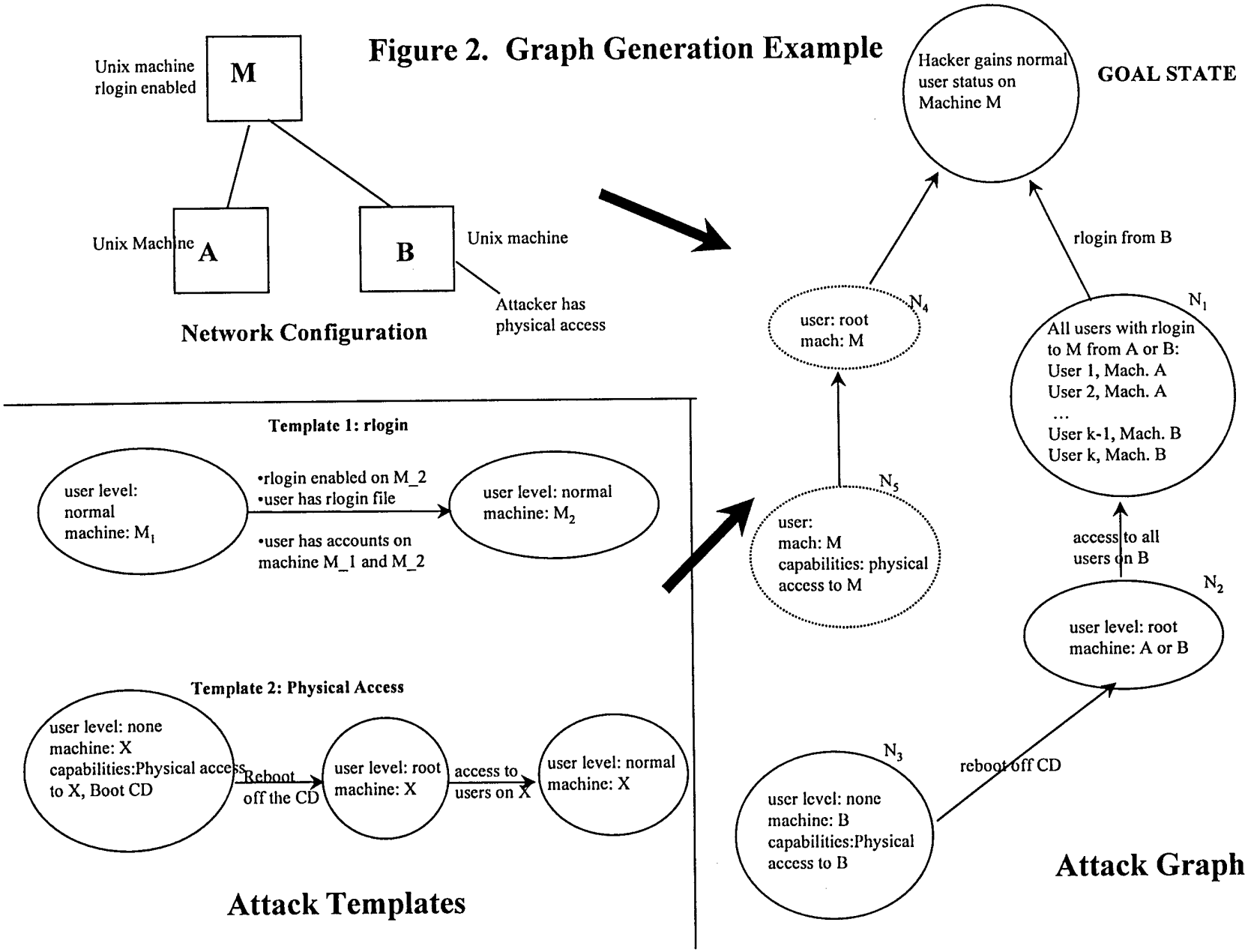


Figure 1. Example template for anonymous ftp attack

Figure 2. Graph Generation Example



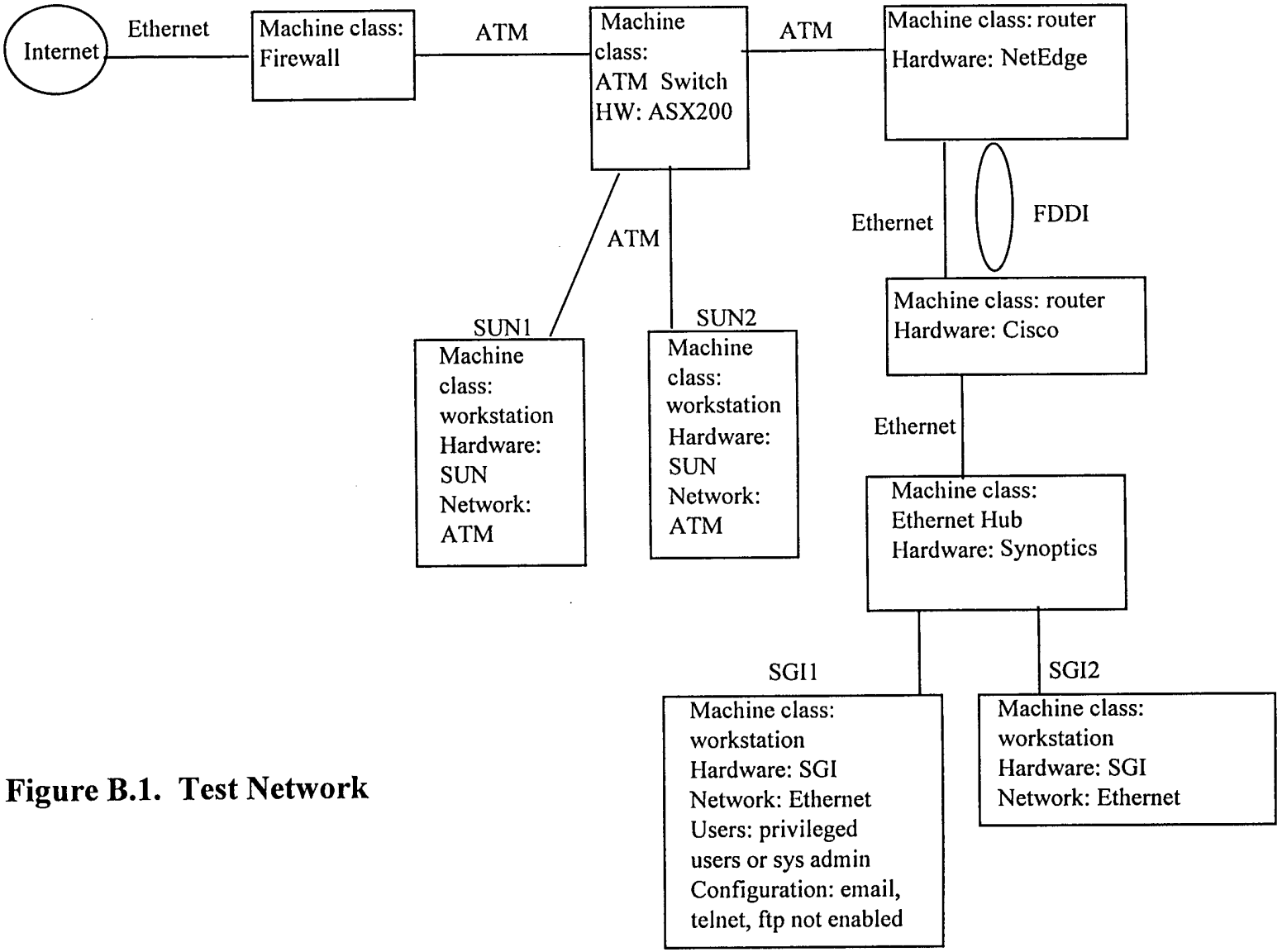


Figure B.1. Test Network

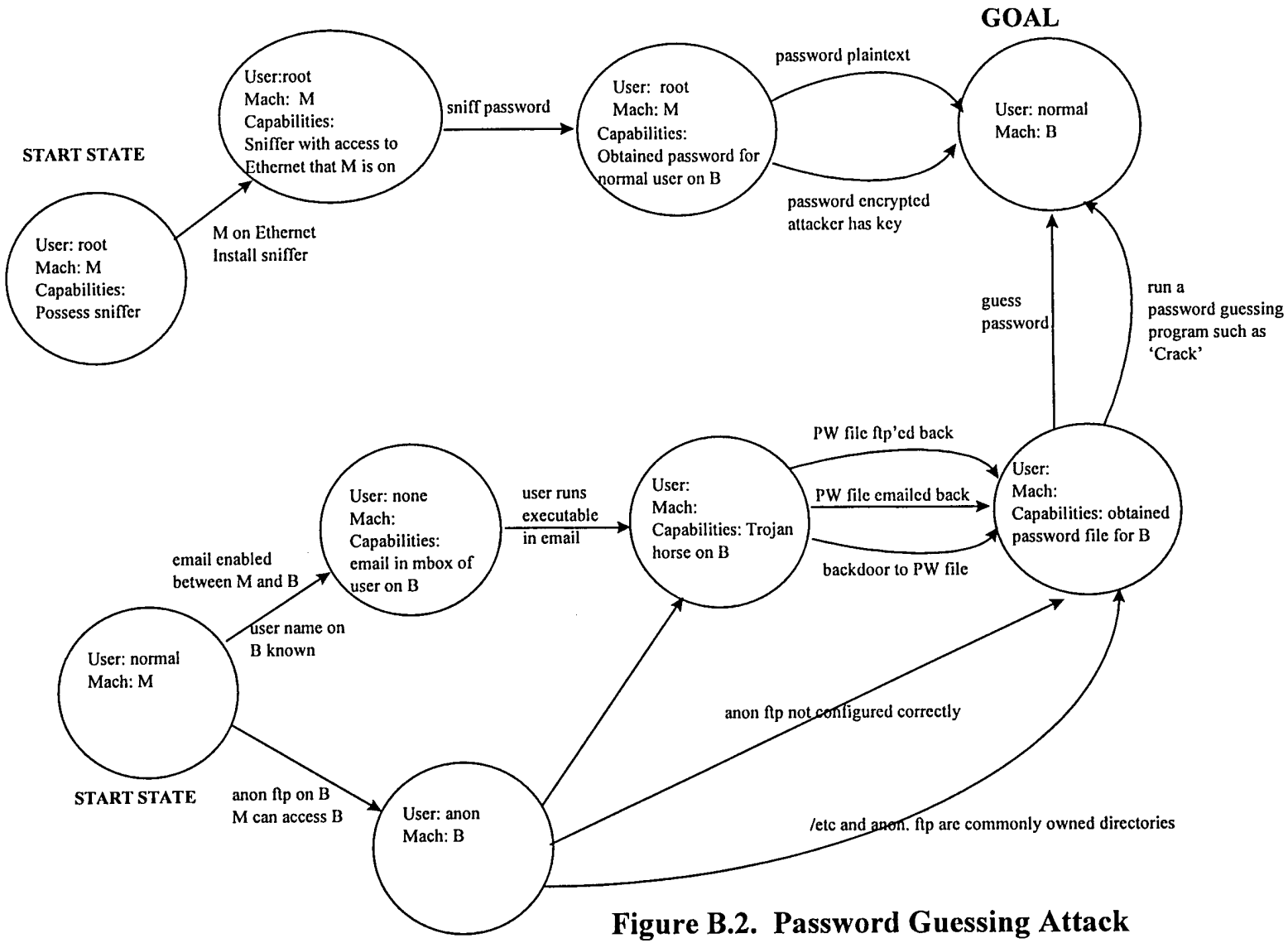
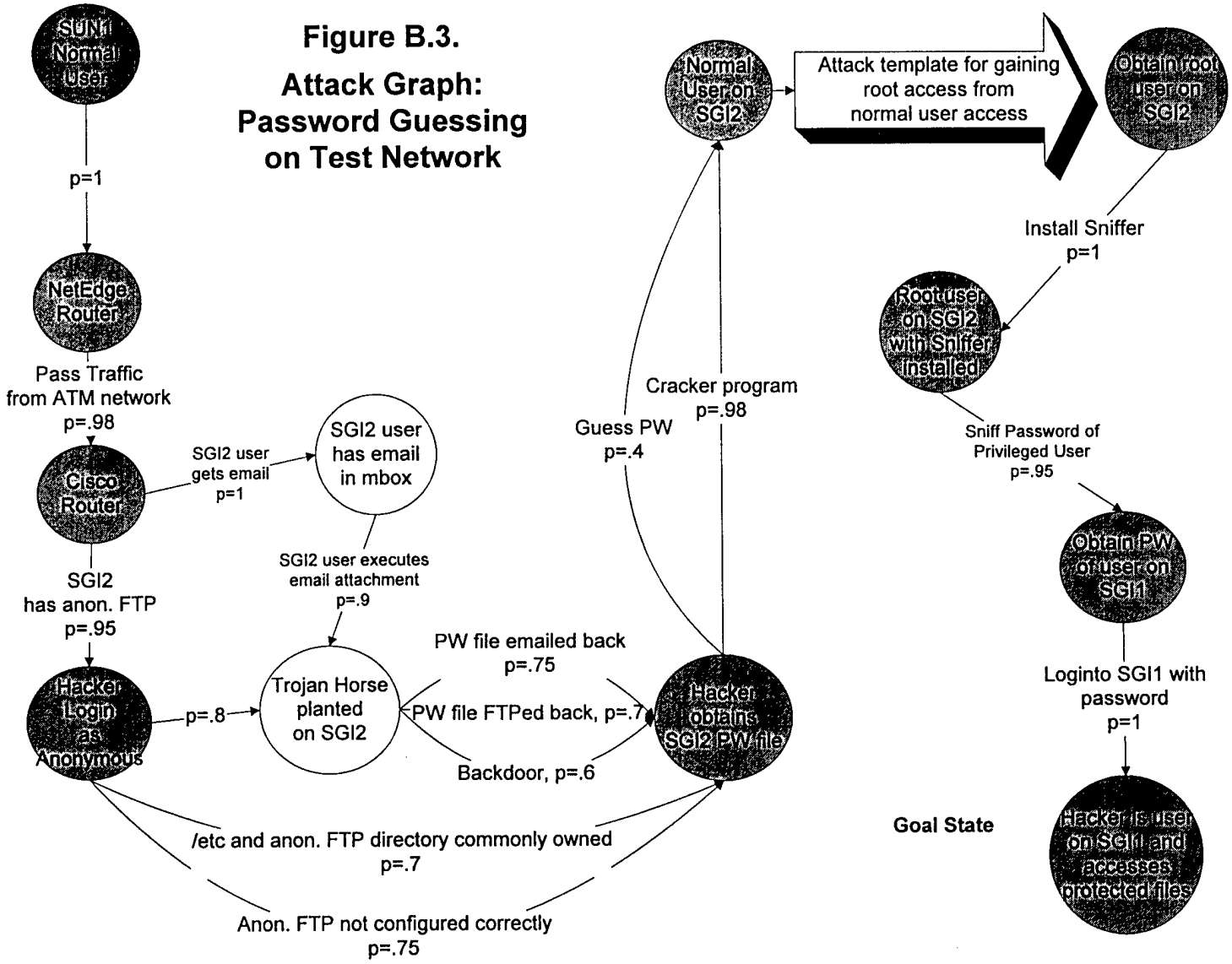


Figure B.2. Password Guessing Attack Template

Figure B.3.

Attack Graph: Password Guessing on Test Network



Report Number (14) SAND--98-1127C
CONF-980914--

Publ. Date (11) 199806
Sponsor Code (18) DOE/MA, XF
UC Category (19) UC-900, DOE/ER