Two Formal Analyses of Attack Graphs*

S. Jha
Computer Sciences Department
University of Wisconsin Madison, WI 53706
E-mail: jha@cs.wisc.edu

O. Sheyner and J. Wing
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
E-mail: {sheyner,wing}@cs.cmu.edu

Abstract

An attack graph is a succinct representation of all paths through a system that end in a state where an intruder has successfully achieved his goal. Today Red Teams determine the vulnerability of networked systems by drawing gigantic attack graphs by hand. Constructing attack graphs by hand is tedious, error-prone, and impractical for large systems. By viewing an attack as a violation of a safety property, we can use off-the-shelf model checking technology to produce attack graphs automatically: a successful path from the intruder's viewpoint is a counterexample produced by the model checker. In this paper we present an algorithm for generating attack graphs using model checking as a subroutine.

Security analysts use attack graphs for detection, defense and forensics. In this paper we present a minimization analysis technique that allows analysts to decide which minimal set of security measures would guarantee the safety of the system. We provide a formal characterization of this problem: we prove that it is polynomially equivalent to the minimum hitting set problem and we present a greedy algorithm with provable bounds. We also present a reliability analysis technique that allows analysts to perform a simple cost-benefit trade-off depending on the likelihoods of attacks. By interpreting attack graphs as Markov Decision Processes we can use the value iteration algorithm to compute the probabilities of intruder success for each attack the graph.

Keywords: *Attack graph, model checking, minimization analysis, reliability analysis, Markov Decision Processes, network vulnerability, security.*

*S. Jha was supported by the Office of Naval Research under contracts N00014-01-1-0796 and N00014-01-1-0708. O. Shyener and J. Wing were supported by the Defense Advanced Research Projects Agency and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DOD, ARO, ONR or the U.S. Government.

1 Motivation

As networks of hosts continue to grow, evaluating their vulnerability to attacks becomes increasingly more important to automate. When evaluating the security of a network, it is not enough to consider the presence or absence of isolated vulnerabilities. A large network builds upon multiple platforms and diverse software packages and supports several modes of connectivity. Inevitably, such a network will contain security holes that have escaped notice of even the most diligent system administrator.

To evaluate the vulnerability of a network of hosts, a security analyst must take into account the effects of interactions of local vulnerabilities and find global vulnerabilities introduced by interconnections. A typical process for vulnerability analysis of a network is shown in Figure 1. First, scanning tools determine vulnerabilities of individual hosts. Using this local vulnerability information along with other information about the network, such as connectivity between hosts, the analyst produces an *attack graph*. Each path in an attack graph is a series of exploits, which we call *atomic attacks*, that leads to an undesirable state (e.g., a state where an intruder has obtained administrative access to a critical host).

1.1 Attack Graphs and Intrusion Detection

Attack graphs can serve as a basis for detection, defense, and forensic analysis. To motivate our study of attack graphs and attack graph generation algorithms, we discuss the potential applications of attack graphs to these areas of security.

Detection
System administrators are increasingly deploying intrusion detections systems (IDSs) to detect and combat attacks on their network. Such systems depend on software sensor modules that detect suspicious events and activity and issue alerts. Setting up the sensors usually involves a trade-off

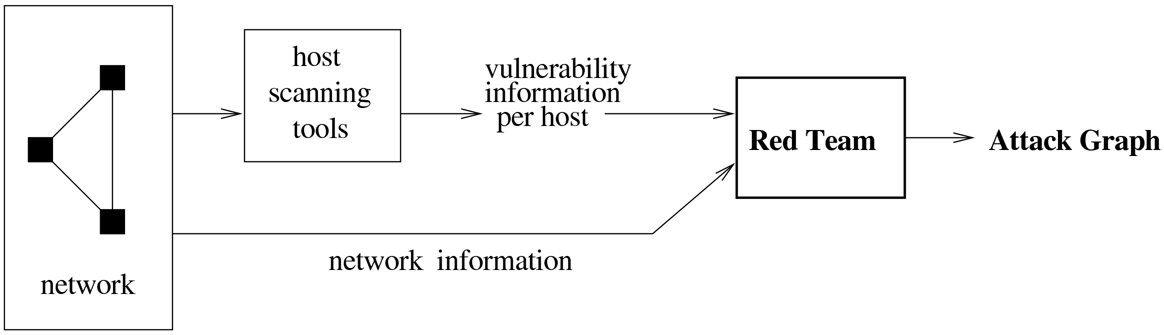

Figure 1. Vulnerability Analysis of a Network

between sensitivity to intrusions and the rate of false alarms in the alert stream. When the sensors are set to report all suspicious events, the sensors frequently issue alerts for benign background events. This often results in administrators turning off the IDS entirely. On the other hand, decreasing sensor sensitivity reduces their ability to detect real attacks.

To deal with this problem, intrusion detection systems usually employ heuristic algorithms to correlate alerts from a large pool of heterogeneous sensors. Valdes and Skinner [19] describe a probabilistic approach to alert correlation. Successful correlation of multiple alerts increases the chance that the suspicious activity indicated by the alerts is in fact malicious.

Attack graphs can enhance both heuristic and probabilistic correlation approaches. Given a graph describing all likely attacks (i.e., sequences of attacker actions), an IDS can match individual alerts to attack edges in the graph. Matching successive alerts to individual paths in the attack graphs dramatically increases the likelihood that the network is under attack. This on-line vigilance allows the IDS to predict attacker goals, aggregate alarms to reduce the volume of alert information to be analyzed, and reduce the false alarms rates. Knowledge of attacker goals and likely next steps helps guide defensive response.

Defense
A further benefit of attack graphs is that they can help analyze potential effectiveness of intrusion detections systems offline. In Section 3 we will show how attack graphs can be generated automatically from models of the network. We will also show that we can incorporate both the security policy and the intrusion detection system in the model and generate attack graphs for specific network configurations. Attack graphs enable an administrator to perform several kinds of analysis to assess their security needs: marking the paths in the attack graph that an IDS will detect; determining where to position new IDS components for best coverage; exploring trade-offs between different security policies and between different software/hardware configurations; and identifying the worst-case scenarios and prioritizing defense accordingly.

Forensics
After a break-in, forensic analysis is used to find probable attacker actions and to assess damage. If legal action is desired, analysts seek evidence that a sequence of sensor alerts comprises a coherent attack plan, and is not merely a series of isolated, benign events. This task becomes even harder when the intruders obfuscate attack steps by slowing down the pace of the attack and varying specific steps. It has been suggested that a convincing argument as to the malicious intent of intruder actions can be constructed by matching data extracted from IDS logs to a formal reference model based on attack graphs [17].

1.2 Our Contributions

We have seen that constructing attack graphs is a crucial part of performing vulnerability analysis of a network of hosts. Currently, *Red Teams* produce attack graphs by hand, often drawing gigantic diagrams on floor-to-ceiling whiteboards. Doing this by hand is tedious, error-prone, and impractical for attack graphs larger than a hundred nodes. We have demonstrated in earlier work that model checking can be applied to automatically generate attack graphs [15]. In this paper, we show that the attack graphs produced by our method are *exhaustive*, i.e., covering all possible attacks, and *succinct*, i.e., containing only relevant states (see Section 3.2).

We also provide a formal and detailed explanation of our model. Our definitions are based on a finite-state model of the network whose state transitions are described using standard pre- and post-conditions. Each state transition corresponds to a single atomic attack by the intruder. A state in the model represents the state of the system between atomic attacks. A typical transition from state s_1 to state s_2 corresponds to an atomic attack whose preconditions are satisfied in s_1 and whose effects hold in state s_2. An *attack* is a sequence of state transitions culminating in the intruder achieving his goal. The entire attack graph is thus a representation of all the possible ways in which the intruder can succeed.

As already discussed, attack graphs can be used to perform a variety of analysis. Specifically, attack graphs can be used to answer the following questions of interest to a system administrator:
Question 1: What successful attacks are undetected by the IDS?
Question 2: If all measures in a set M' are implemented, does the network become safe (more secure)?
Question 3: Given a set of measures M, what is the smallest subset of measures whose implementation makes the network safe?

Answers to these questions, can help an analyst or network administrator in choosing the best upgrade strategy. These questions are addressed in Section 5.

When we are modeling a system operating in an uncertain environment, certain transitions in the model represent the system's reaction to changes in the environment. We can think of such transitions as being outside of the system's control – they occur when triggered by the environment. When no empirical information is available about the relative likelihood of such environment-driven transitions, we can model them only as nondeterministic "choices" made by the environment. Moreover, for new vulnerabilities data for estimating probabilities might not be available. However, sometimes empirical data make it possible to assign probabilities to environment-driven transitions. We would like to take advantage of such information to quantify the probabilistic behavior of attack graphs. In this context, a system administrator might be interested in the following question:
Question 4: The deployment of which security measure(s) will increase the likelihood of thwarting an attacker?
The answer to this question is provided in Section 6.1. The system administrator can use the answer to question 4 to perform a quantitative evaluation of various security fixes (see example 1).

The main contributions of this paper over our earlier work [15] are:

- This paper explores the semantics of our network model more formally.

- In our earlier work [15] we proved that finding a *minimum* set of atomic attacks which must be removed to thwart an intruder is NP-complete. In this paper, we further explore the complexity of this problem. Section 5.1 proves that the problem is polynomially equivalent to the minimum hitting set problem where the collection of sets is represented as a labeled directed graph. This reduction provides us additional insight, which enabled us to find a greedy algorithm with provable bounds, which can be used to answer questions 1,2, and 3.

- This paper also presents an algorithm to compute the reliability (which is defined as the likelihood of an intruder not succeeding) for a network. A desirable feature of our algorithm is that it allows for *incomplete information*, i.e., probabilities of all transitions need not be provided. To our knowledge, previous metrics in the area of security require complete information.

Related work is provided in Section 2. Section 3 describes our model and our model checking based algorithm to generate attack graphs. An example network is presented in Section 4. Section 4 also describes our network model in detail. This network is used throughout the paper for illustrative purposes. In Section 5 we present analysis which helps us answers questions 1,2, and 3. Section 6 describes probabilistic attack graphs and our algorithm to compute reliability. Answer to question 4 is provided in that section. The proof of correctness of our algorithm to compute reliability are based on Markov Decision Processes and presented in the companion technical report [9]. Finally, directions for future work and concluding remarks are presented in Section 7.

2 Related Work

Phillips and Swiler [12] propose a concept of attack graphs that is very similar to the one described here. However, they take an "attack-centric" view of the system. Since we work with a general input language, in our model seemingly benign system events (such as failure of a link and user errors) and attacks occur simultaneously. Therefore, our attack graphs are more general than the one proposed by Phillips and Swiler. Based on these ideas a tool for generating attack graphs is presented in [18]. The tool constructs the attack graph by forward exploration starting from the initial state. A symbolic model checker (like NuSMV) works backward from the goal state to construct the attack graph. A major advantage of the backward algorithm is that vulnerabilities that are not relevant to the safety property (or the goal of the intruder) are never explored. This can result in significant savings in space. In fact, Swiler *et al.* [18] refer to the advantages of the backward search in their paper. Moreover, by using model checking we leverage off all the sophisticated reduction techniques developed in that area. The post-facto analysis suggested by Phillips and Swiler is also different from the one presented in this paper. We plan to incorporate their analysis into our tool.

Dacier [6] proposes the concept of privilege graphs, where each node represents a set of privileges owned by the user and arcs represent vulnerabilities. Privilege graphs are then explored to construct attack state graphs, which represents different ways in which an intruder can reach a certain goal, such as root access on a host. Based on the attack state graphs a metric, called the *mean effort to failure* or METF, is proposed. An experimental evaluation of their framework is described by Orlato *et al.* [11]. At the surface our notion of attack graphs seem similar to the one proposed by Dacier. However, as is the case with Phillips and Swiler, Dacier again takes an "attack-centric" view of the world. As pointed out earlier, our attack graphs are more general. From the experiments conducted by Orlato *et al.* it appears that even for small examples the space required to construct attack state graphs becomes prohibitive. Model checking has made significant advances in representing large state spaces. Therefore, by basing our algorithm on model checking we leverage off those advances and can hope to represent large attack graphs. However, the analytical analysis proposed by Dacier can also be performed on attack graphs constructed by our tool. We also plan to conduct an experimental evaluation similar to the one performed by Orlato et al.

Ritchey and Amman [14] also used model checking for vulnerability analysis of networks. They used the unmodified model checker SMV [16]. Therefore, they could only obtain one counter-example or one attack corresponding to a intruder's goal. In contrast, we modified the model checker NuSMV to produce complete attack graphs, which represents all possible attacks. We also described analysis that can be performed on these attack graphs. These analysis techniques cannot be meaningfully performed on single attacks.

3 Generating Attack Graphs using Model Checking

First, we formally define *attack graphs*, the data structure used to represent all possible attacks on our networked system. We restrict our attention to attack graphs representing violations of safety properties[1].

Definition 1 Let *AP* be a set of atomic propositions. An *attack graph* or *AG* is a tuple $G = (S, \tau, S_0, S_s, L)$, where S is a set of states, $\tau \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, $S_s \subseteq S$ is a set of success states, and $L : S \rightarrow 2^{AP}$ is a labeling of states with a set of propositions true in that state. Intuitively, S_s denotes intruder's goals, e.g., obtaining root access on a critical host.

Unless stated otherwise, we assume that the transition relation τ is total. We define an *execution fragment* as a finite sequence of states $s_0 s_1 ... s_n$ such that $(s_i, s_{i+1}) \in \tau$ for all $0 \leq i < n$. An execution fragment with $s_0 \in S_0$ is an *execution*, and an execution whose final state is in S_s is an *attack*, i.e., the execution corresponds to a sequence of atomic attacks leading to the intruder's goal state.

[1] We say more on liveness properties in Section 7.

Next we turn our attention to algorithms for automatic generation of attack graphs. Starting with a description of a network model M and a security property p, the task is to construct an attack graph representing all executions of M that violate p—these are the successful attacks. For the kinds of attack graph analysis suggested in Section 1, it is essential that the graphs produced by the algorithms be *exhaustive* and *succinct*. An attack graph is exhaustive with respect to a model M and correctness property p if it covers all possible attacks in M leading to a violation of p, and succinct if it only contains those states of M from which the system can get to a state violating p.

3.1 Reachability Analysis

If we restrict ourselves to safety properties, an attack graph may be constructed by performing a simple state-space search. Starting with the initial states of the model M, we use a graph traversal procedure (e.g., depth-first search) to find all reachable *success* states where the safety property p is violated. The attack graph is the union of all paths from initial states to success states.

While this algorithm has the advantage of simplicity, it handles only safety properties and may run into the state explosion problem for non-trivial models. Model checking has dealt with both of these issues with some success, so we will consider algorithms based on that technology.

3.2 Model Checking Algorithm

Model checking is a technique for checking whether a formal model M of a system satisfies a given property p. In our work, we use the model checker NuSMV [10], for which the model M is a finite labeled transition system and p is a property expressed in *Computation Tree Logic* (*CTL*). For now, we consider only safety properties, which in CTL have the form $\mathbf{AG}f$ (i.e., $p = \mathbf{AG}f$, where f is a formula in propositional logic). If the model M satisfies the property p, NuSMV reports "true." If M does not satisfy p, NuSMV produces a *counter-example*. A single counter-example shows an execution that leads to a violation of the property. In this section, we explain how to construct attack graphs for safety properties using model checking.

Attack graphs depict ways in which the system can reach an unsafe state (or, equivalently, a successful state for the intruder). We can express the property that an unsafe state cannot be reached as:

$$\mathbf{AG}(\neg \mathit{unsafe})$$

When this property is false, there are unsafe states that are reachable from the initial state. The precise meaning of

Input:

S – set of states
$R \subseteq S \times S$ – transition relation
$S_0 \subseteq S$ – set of initial states
$L : S \rightarrow 2^{AP}$ – labeling of states with propositional formulas
$p = \mathbf{AG}(\neg unsafe)$ (a safety property)

Output:

attack graph $G_p = (S_{unsafe}, R^p, S_0^p, S_s^p, L)$

Algorithm: $GenerateAttackGraph(S, R, S_0, L, p)$

(* Use model checking to find the set of states S_{unsafe} that violate the safety property $\mathbf{AG}(\neg unsafe)$. *)

$S_{unsafe} \;=\; modelCheck(S, R, S_0, L, p).$

(* Restrict the transition relation R to states in the set S_{unsafe} *)

$R^p \;=\; R \cap (S_{unsafe} \times S_{unsafe}).$
$S_0^p \;=\; S_0 \cap S_{unsafe}.$
$S_s^p \;=\; \{s | s \in S_{unsafe} \wedge s \models unsafe\}.$
$return(S_{unsafe}, R^p, S_0^p, S_s^p, L).$

Figure 2. Algorithm for Generating Attack Graphs

unsafe depends on the application. For example, in the network security example given in Section 4, the property given below is used to express that the privilege level of the intruder on the host with index 2 should always be less than the root (administrative) privilege.

$$\mathbf{AG}(\textit{network.adversary.privilege[2]} < \textit{network.priv.root})$$

We briefly describe the algorithm for constructing attack graphs for the property $\mathbf{AG}(\neg unsafe)$. The first step is to determine the set of states S_r that are reachable from the initial state. Next, the algorithm computes the set of reachable states S_{unsafe} that have a path to an unsafe state. The set of states S_{unsafe} is computed using an iterative algorithm derived from a fix-point characterization of the **AG** operator [4]. Let R be the transition relation of the model, i.e., $(s, s') \in R$ if and only if there is a transition from state s to s'. By restricting the domain and range of R to S_{unsafe} we obtain a transition relation R^p that represents the edges of the attack graph. Therefore, the attack graph is $(S_{unsafe}, R^p, S_0^p, S_s^p, L)$, where S_{unsafe} and R^p represent the set of nodes and edges of the graph, respectively, $S_0^p = S_0 \cap S_{unsafe}$ is the set of initial states, and $S_s^p = \{s | s \in S_{unsafe} \wedge s \models unsafe\}$ is the set of success states. This algorithm is given in Figure 2.

In symbolic model checkers, such as NuSMV, the transition relation and sets of states are represented using BDDs [3], a compact representation for boolean functions. There are efficient BDD algorithms for all operations used in the algorithm shown in Figure 2.

3.3 Attack Graph Properties

We can show that an attack graph G generated by the algorithm in Figure 2 is *exhaustive* (Lemma 1(a)) and *succinct* with respect to states and edges (Lemma 1(b) and 1(c)). Proof of the following lemma is straightforward and follows from the definitions.

Lemma 1 The following properties of the attack graph G are true:
(a) *exhaustive*
An execution e of the input model (S, R, S_0, L) violates the property $p = \mathbf{AG}(\neg unsafe)$ if and only if e is an attack in the attack graph $G = (S_{unsafe}, R^p, S_0^p, S_s^p, L)$.
(b) *succinct with respect to states*
A state s of the input model (S, R, S_0, L) is in the attack graph G if and only if there is an attack in G that contains s.
(c) *succinct with respect to edges*
An edge $t = (s_1, s_2)$ of the input model (S, R, S_0, L) is in the attack graph G if and only if there is an attack in G that includes t.

4 A Simple Intrusion Detection Example

Consider the example network shown in Figure 3. There are two target hosts, ip_1 and ip_2, and a firewall separating them from the rest of the Internet. As shown, each host is running two of three possible services (ftp, sshd, a database). An intrusion detection system (IDS) monitors the network traffic between the target hosts and the outside

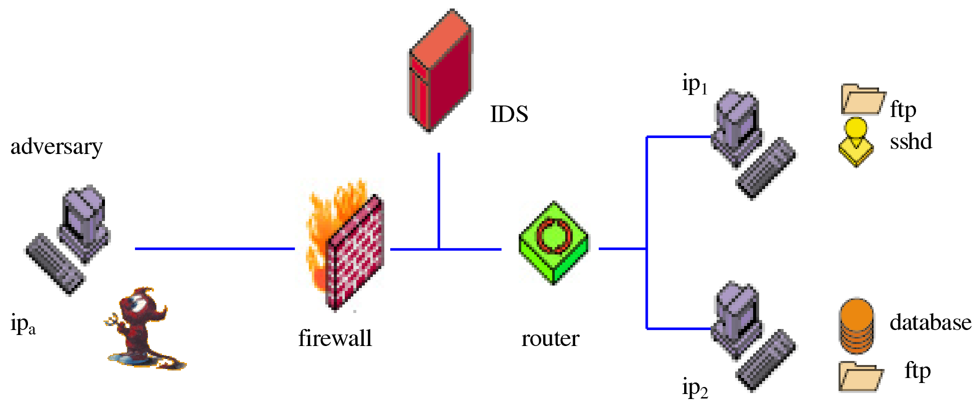

Figure 3. Example Network

world. There are four possible atomic attacks, identified numerically as follows: (0) sshd buffer overflow, (1) ftp .rhosts, (2) remote login, and (3) local buffer overflow. If an atomic attack is *detectable*, the intrusion detection system will trigger an alarm; if an attack is *stealthy*, the IDS misses it. The ftp .rhosts attack needs to find the target host with two vulnerabilities: a writable home directory and an executable command shell assigned to the ftp user name. The local buffer overflow exploits a vulnerable version of the xterm executable.

In this section, we construct a finite state model of the example network so that each state transition corresponds to a single atomic attack by the intruder. A state in the model represents the state of the system between atomic attacks. A typical transition from state s_1 to state s_2 corresponds to an atomic attack whose preconditions are satisfied in s_1 and whose effects hold in state s_2.

The intruder launches his attack starting from a single computer, ip_a, which lies outside the firewall. His eventual goal is to disrupt the functioning of the database. For which, the intruder needs root access on the database host ip_2.

4.1 States of the Finite State Machine Model

The Network

We model the network as a set of facts, each represented as a relational predicate. The state of the network specifies services, host vulnerabilities, connectivity, and a remote login trust relationship between hosts. There are six boolean variables for each host, specifying whether any of the three modeled services are running and whether any vulnerabilities are present on that host.

variable	meaning
ssh_h	ssh service is running on host h
ftp_h	ftp service is running on host h
data_h	database is running on host h
wdir_h	ftp home directory is writable on host h
fshell_h	ftp user has executable shell on host h
xterm_h	xterm executable is vulnerable to overflow on host h

Connectivity is expressed as a ternary relation $R \subseteq Host \times Host \times Port$, where $R(h_1, h_2, p)$ means that host h_2 is reachable from host h_1 on port p. The constants *sp* and *fp* will refer to the specific ports for the ssh and ftp services, respectively. Slightly abusing notation, we write $R(h_1, h_2)$ when there is a network route from h_1 to h_2. Similarly, we model trust as a binary relation *RshTrust* $\subseteq Host \times Host$, where *RshTrust*$(h_1, h_2)$ indicates that a user may log in from host h_2 to host h_1 without authentication (i.e., host h_1 "trusts" host h_2).

The Intruder

The function $plvl_A$: *Hosts* $\rightarrow$ {*none, user, root*} gives the level of privilege that intruder A has on each host. There is a total order on the privilege levels: *none* $<$ *user* $<$ *root*.

Several state variables specify which attack the intruder will attempt next:

variable	meaning
attack	attack type
source	source host
target	target host
strain	stealthy/detectable attack

Intrusion Detection System

Atomic attacks are classified as being either *detectable* or *stealthy* with respect to the Intrusion Detection System (IDS). If an attack is detectable, it will trigger an alarm when executed on a host or network segment monitored by the IDS; if an attack is *stealthy*, the IDS does not detect it.

We specify the IDS with a function *ids: Host* $\times$ *Host* $\times$ *Attack* $\rightarrow \{d, s, b\}$, where *ids*(h_1, h_2, a) = d if attack a is *d*etectable when executed with source host h_1 and target host h_2; *ids*(h_1, h_2, a) = s if attack a is *s*tealthy when executed with source host h_1 and target host h_2; and *ids*(h_1, h_2, a) = b if attack a has *b*oth detectable and stealthy strains, and success in detecting the attack depends on which strain is used. When h_1 and h_2 refer to the same host, *ids*(h_1, h_2, a) specifies the intrusion detection system component (if any) located on that host. When h_1 and h_2 refer to different hosts, *ids*(h_1, h_2, a) specifies the intrusion detection system component (if any) monitoring the network path between h_1 and h_2. In addition, a global boolean variable specifies whether the IDS alarm has been triggered by any previously executed atomic attack.

4.2 Initial States

Initially, there is no trust between any of the hosts; the trust relation Tr is empty. The connectivity relation R is shown in the following table. An entry in the table corresponds to a pair of hosts (h_1, h_2). Each entry is a triple of boolean values. The first value is 'y' if h_1 and h_2 are connected by a physical link, the second value is 'y' if h_1 can connect to h_2 on the ftp port, and the third value is 'y' if h_1 can connect to h_2 on the sshd port.

R	ip_a	ip_1	ip_2
ip_a	y,n,n	y,y,y	y,y,n
ip_1	y,n,n	y,y,y	y,y,n
ip_2	y,n,n	y,y,y	y,y,n

We use the connectivity relation to reflect the firewall rule sets as well as the existence of physical links. For the table above, the firewall is open and does not place any restrictions on the flow of network traffic.

Initially, the intruder has *root* privileges on his own machine ip_a and no privileges on the other hosts.

The paths between (ip_a, ip_1) and between (ip_a, ip_2) are monitored by a single network-based IDS. The path between (ip_1, ip_2) is not monitored. There are no other host-based intrusion detection components. The IDS detects the remote login attack, and the detectable strains of the sshd buffer overflow attack.

4.3 Transitions

Our model has nondeterministic state transitions. If the current state of the network satisfies the preconditions of more than one atomic attack rule, the intruder nondeterministically "chooses" one of those attacks. The state then changes according to the effects clause of the chosen attack rule. The intruder repeats this process until his goal is achieved.

We model four atomic attacks. Throughout the description, S is used to designate the source host and T the target host. $R(S, T, p)$ denotes that host T is reachable from host S on port p.

Sshd Buffer Overflow

This remote-to-root attack immediately gives a remote user a root shell on the target machine.

attack sshd-buffer-overflow **is**
  intruder preconditions
    [User-level privileges on host S]
    $plvl_A(S) \geq$ user
    [No root-level privileges on host T]
    $plvl_A(T) <$ root

  network preconditions
    [Host T is running sshd]
    ssh_T
    [Host T is reachable from S on port sp*]*
    $R(S, T, sp)$
  intruder effects
    [Root-level privileges on host T]
    $plvl_A(T) =$ root
  network effects
    [Host T is not running sshd]
    $\neg\text{ssh}_T$
end

Ftp .rhosts

Using an ftp vulnerability, the intruder creates an .rhosts file in the ftp home directory, creating a remote login trust relationship between his machine and the target machine.

attack ftp-rhosts **is**
  intruder preconditions
    [User-level privileges on host S]
    $plvl_A(S) \geq$ user

network preconditions

[Host T is running ftp]

ftp_T

[Host T is reachable from S on port fp*]*

$R(S, T, fp)$

[Ftp directory writable on host T]

wdir_T

[Ftp user has been assigned a valid shell on host T]

fshell_T

[No rsh trust for some host X and T]

$\exists X.\neg RshTrust(X, T)$

intruder effects

none

network effects

[Rsh trust between all hosts and T]

$\forall X.RshTrust(X, T)$

end

Remote Login

Using an existing remote login trust relationship between two machines, the intruder logs in from one machine to another, getting a user shell without supplying a password. This operation is usually a legitimate action performed by regular users, but from the intruder's viewpoint, it is an atomic attack.

attack rsh-login **is**

intruder preconditions

[User-level privileges on host S]

$plvl_A(S) = \text{user}$

[No privileges on host T]

$plvl_A(T) = \text{none}$

network preconditions

[Rsh trust between S and T]

$RshTrust(S, T)$

[Host T is reachable from S]

$R(S, T)$

intruder effects

[User-level privileges on host T]

$plvl_A(T) = \text{user}$

network effects

none

end

Local Buffer Overflow

If the intruder has acquired a user shell on the target machine, the next step is to exploit a buffer overflow vulnerability on a `setuid root` file to gain root access.

attack local-setuid-buffer-overflow **is**

intruder preconditions

[User-level privileges on host T]

$plvl_A(T) = \text{user}$

network preconditions

[There is a vulnerable xterm executable]

xterm_T

intruder effects

[Root-level privileges on host T]

$plvl_A(T) = \text{root}$

network effects

none

end

It is easy to see that each atomic attack strictly increases either the intruder's privilege level on the target host or remote login trust between hosts. This means that *the attack graph has no cycles*.

From our finite model we can now automatically construct attack graphs that demonstrate how the intruder can violate various security properties. Suppose we want to generate all attacks that demonstrate how the intruder can gain root privilege on host ip_2 and remain undetected by the IDS. The CTL formula that expresses the safety property that *the intruder on host with id* 2 *always has privilege level below root* or *is detected* is expressed using the following CTL property:

$$\mathbf{AG}(network.adversary.privilege[2] < network.priv.root \mid network.detected)$$

Figure 4 shows the attack graph produced by our tool for this property. Each node is labeled by an attack id number, which corresponds to the atomic attack *to be attempted next*; a flag S/D indicates whether the attack is stealthy or detectable by the intrusion detection system; and the numbers of the source and target hosts (ip_a corresponds to host number 0).

Any path in the graph from the root node to a leaf node shows a sequence of atomic attacks that the intruder can employ to achieve his goal while remaining undetected. For instance, the path highlighted by double-boxed nodes consists of the following sequence of four atomic attacks: overflow sshd buffer on host 1, overwrite .rhosts file on host 2 to establish rsh trust between hosts 1 and 2, log in using rsh from host 1 to host 2, and finally, overflow a local buffer on host 2 to obtain root privileges.

We have also expanded the example described above by adding two additional hosts, four additional atomic attacks, several new vulnerabilities, and flexible firewall configurations. *For the larger example the attack graph has* 5948 *nodes and* 68364 *edges.*

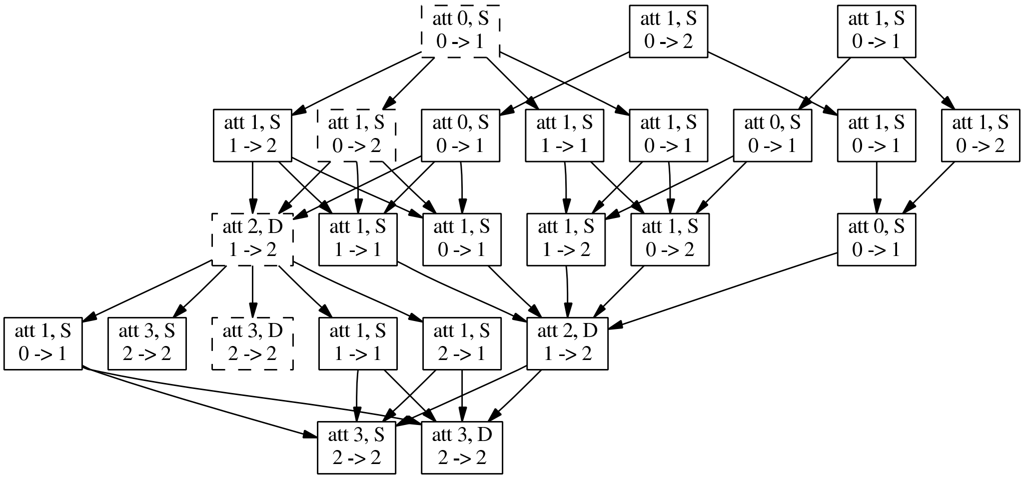

Figure 4. Attack Graph

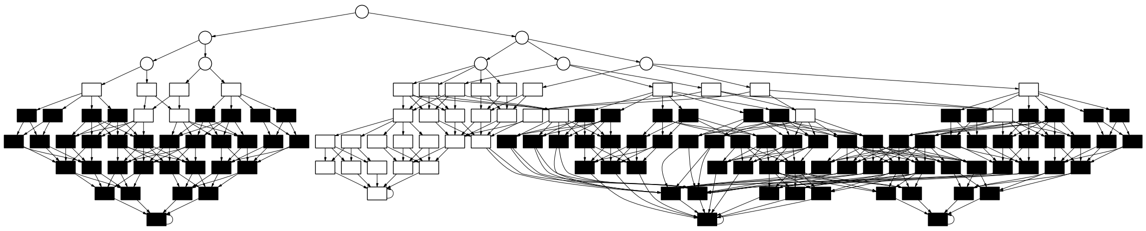

Figure 5. Attack Graph Analysis

5 Minimization Analysis

Once we have an attack graph generated for a specific network with respect to a given safety property, we can utilize it for further analysis. For example, an analyst may be faced with a choice between deploying additional network attack detection tools or prevention techniques. An analyst has a set of *measures*, such as deploying additional network detection tools or upgrading the software, available to him. This section provides answers to questions 1,2, and 3 posed in Section 1.2.

Question 1: What successful attacks are undetected by the IDS?

Answer:

To answer this question, we modify the model slightly, making only a subset of atomic attacks available to the intruder. For simplicity, we nondeterministically decide which subset to consider initially, before any attack begins; once the choice is made, the subset of available atomic attacks is fixed during any given attack. We ran the model checker on the modified model with the invariant property that says the intruder never obtains root privilege on host ip_2:

$$\mathbf{AG}(\textit{network.adversary.privilege[2]} < \textit{network.priv.root})$$

The post-processor marked the states where the intruder has been detected by the IDS. The result is shown in Figure 5. The white rectangles indicate states where the attacker had not yet been detected by the intrusion detection system. The black rectangles are states where the intrusion detection system has sounded an alarm. Thus, white leaf nodes are desirable for the attacker because his objective is achieved without detection. Black leaf nodes are less desirable—the attacker achieves his objective, but the alarm goes off.

The resolution of which atomic attacks are available to the intruder happens in the circular nodes near the root of the graph. The first transition out of the root (initial) state picks the subset of attacks that the intruder will use. Each child of the root node is itself the root of a disjoint subgraph where the subset of atomic attacks chosen for that child is used. Note that the number of such subgraphs descending from the root node corresponds to the number of subsets of atomic attacks with which the intruder can be successful—

the model checker determines that for any other possible subset, there is no possible successful sequence of atomic attacks.

The root of the graph in Figure 5 has two subgraphs, corresponding to the two subsets of atomic attacks that will allow the intruder to succeed. In the left subgraph the sshd buffer overflow attack is not available to the intruder; it can be readily seen that the intruder can still succeed, but cannot do so while remaining undetected by the IDS. In the right subgraph, all attacks are available. Thus, the entire attack graph implies that all atomic attacks other than the sshd attack are indispensable: the intruder cannot succeed without them. The analyst can use this information to guide decisions on which network defenses can be profitably upgraded.

The white cluster in the middle of the figure is isomorphic to the attack graph presented in Figure 4; it shows attacks in which the intruder can achieve his objective without detection (i.e., all paths by which the intruder reaches a white leaf in the graph).

We proceed to provide answers to questions 2 and 3. However, first we define an attack graph whose edges are labeled with atomic attacks, which is produced by our post-processor. Such an attack graph is used to perform further analysis to answer questions 2 and 3. Assume that we have produced an attack graph corresponding to the following safety property:

$$\mathbf{AG}(\neg \mathit{unsafe})$$

Let $\mathcal{A}$ be the set of atomic attacks, and $G = (S, E, s_0, s_s, L)$ be the attack graph, where S is the set of states, $E \subseteq S \times S$ is the set of edges, $s_0 \in S$ is the initial state, $s_s \in S$ is the success state for the intruder, and $L : E \rightarrow \mathcal{A} \cup \{\epsilon\}$ is a labeling function where $L(e) = a$ if an edge $e = (s \rightarrow s')$ corresponds to an atomic attack a, otherwise $L(e) = \epsilon$. Edges labeled with ϵ represent system transitions that do not correspond to an atomic attack. Moreover, as demonstrated below additional ϵ edges can be also introduced by our construction. Without loss of generality we can assume that there is a single initial and success state. For example, consider an attack graph with multiple initial states $s_0^1, \cdots, s_0^j$ and success states $s_s^1, \cdots, s_s^u$. We can add a new initial state s_0 and a new success state s_s with ϵ-labeled edges (s_0, s_0^m) $(1 \leq m \leq j)$ and (s_s, s_s^t) $(1 \leq t \leq u)$. Suppose we are given a finite set of measures $M = \{m_1, \cdots, m_k\}$ and a function $\mathit{covers} : M \rightarrow 2^{\mathcal{A}}$. An atomic attack $a \in \mathit{covers}(m_i)$ if adopting measure m_i removes the atomic attack a.

Question 2: If all measures in a set M' are implemented, does the network become safe (more secure)?
Answer:
A network administrator wants to find out whether adopting measures from a set $M' \subseteq M$ will make the network safe. This question can be answered in linear time using the attack graph G. First, we define $\mathit{covers}(M')$ as $\bigcup_{m \in M'} \mathit{covers}(m)$. Next, we remove all edges e from G such that $L(e) \in \mathit{covers}(M')$. The network is safe iff the success state s_s is not reachable from the initial state s_0. This simple reachability question can be answer in time that is linear in the size of the graph.

Question 3: Given a set of measures M, what is the smallest subset of measures whose implementation makes the network safe?
Answer:
A network administrator wishes to find a subset $M' \subseteq M$ of smallest size, such that adopting the measures in the set M' will make the network safe. Unfortunately, this problem is NP-complete, but we develop good approximation algorithms. We proceed in two steps:

- **Step 1 (Finding a small set of atomic attacks.)**
  In this step, we find a set of atomic attacks whose removal makes the network safe. Checking every possible subset of attacks is exponential in the number of attacks. In a related paper [15], we show that finding the *minimum* set of atomic attacks which must be removed to thwart an intruder is in fact *NP*-complete. We also demonstrated how a *minimal* set can be found in polynomial-time. In this paper, we further explore the complexity of this problem. Section 5.1 proves that the problem of finding a minimum set of attacks is polynomially equivalent to the minimum hitting set problem, where the collection of sets is represented as a labeled directed graph. This reduction provides us additional insight, which enabled us to find a greedy algorithm with provable bounds.

- **Step 2 (Finding a small set of measures)**
  Assume that we find a set of atomic attacks A' whose removal makes the network safe, or equivalently thwarts the intruder. Recall that $M = \{m_1, \cdots, m_k\}$ is the set of measures and $\mathit{covers} : M \rightarrow 2^{\mathcal{A}}$ is a function, where $\mathit{covers}(m_i)$ represents the set of atomic attacks that are removed by adopting the measure m_i. With each attack a in the set A', we associate a set of measures $M(a)$ which is $\{m_i \mid a \in \mathit{covers}(m_i)\}$. The set of attacks A' defines a collection $C_{A'}$ of subsets of M. We wish to find the smallest subset $M' \subseteq M$ such that for all $a \in A'$ there exists an $m_i \in M'$ such that $a \in \mathit{covers}(m_i)$, or equivalently $M' \cap M(a) \neq \emptyset$. This is known as the minimum hitting set problem, which is NP-complete, but good approximation algorithms exist to solve this problem (see Section 5.2)

5.1 The Minimum Critical Attack Sets and the Minimum Hitting Set Problem

This section addresses the first step in the answer to question 3. Assume that we are given an attack graph $G = (S, E, s_0, s_s, L)$, where S is the set of states, $E \subseteq S \times S$ is the set of edges, $s_0 \in S$ is the initial state, $s_s \in S$ is the success state for the intruder, and $L : E \rightarrow \mathcal{A} \cup \{\epsilon\}$ is a labeling function.

Given a state $s \in S$, a set of attacks C is *critical* with respect to s if and only if the intruder cannot reach his goal from s when the attacks in C are removed from his arsenal. Equivalently, C is critical with respect to s if and only if every path from s to the success state s_s has at least one edge labeled with an attack $a \in C$.

A critical set corresponding to a state s is *minimum* (denoted by $M(s)$) if there is no critical set $M'(s)$ such that $|M'(s)| < |M(s)|$. In general, there can be multiple minimum sets corresponding to a state s. Of course, all minimum critical sets must be of the same size.

A critical set of an attack graph $G = (S, E, s_0, s_s, L)$ is defined as a critical set corresponding to the initial state s_0. Therefore, the *Minimum Critical Set of Attacks (MCSA) problem* is the problem of finding a minimum critical set of attacks $M(s_0)$. The decision version of the problem is defined as follows: given an attack graph $G = (S, E, s_0, s_s, L)$ and a positive integer K, is there a critical set of attacks $A \subseteq \mathcal{A}$ such that $|A| \leq K$.

Assume that we are give an attack graph $G = (S, E, s_0, s_s, L)$. A *path* π is sequence of states $q_1, \cdots, q_n$, such that $q_i \in S$ and $(q_i, q_{i+1}) \in E$. A *complete path* starts from the initial state s_0 and ends in the success state s_s. The label of a path $\pi = q_1, \cdots, q_n$ (abusing notation, we will denote it also as $L(\pi)$) is a subset of a set of attacks $\mathcal{A}$

$$\bigcup_{i=1}^{n-1} \{L(q_i, q_{i+1})\} \ \setminus \ \{\epsilon\} \ .$$

$L(\pi)$ represents the set of atomic attacks used on the path π. A set of attacks $A \subseteq \mathcal{A}$ is called *realizable* in the attack graph G iff there exists a complete path π in G such that $L(\pi) = A$. In other words, an intruder can use the set of attacks A to start from the initial state and reach the success state. The set of all realizable sets in an attack graph G is denoted by $Rel(G)$. The following lemma is easy to prove and follows straight from the definitions.

Lemma 2 Assume that we are give an attack graph $G = (S, E, s_0, s_s, L)$. A set of attacks A is critical iff

$$\forall A' \in Rel(G). A' \cap A \neq \emptyset \ .$$

In other words, all realizable sets have a non-empty intersection with a critical set A.

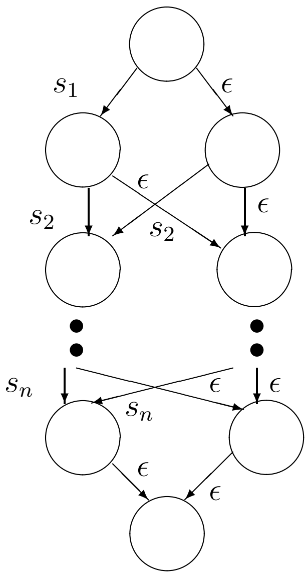

Figure 6. Attack graph representing an exponential number of realizable sets.

Definition 2 Hitting Set [8, Problem SP8]
Instance: Collection C of subsets of a finite set S, positive integer $K \leq |S|$.
Question: Is there a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from each subset in C?

The discussion given in appendix A proves that the problem of finding critical sets in an attack graph is *polynomially equivalent* to finding hitting sets for a collection, with one caveat–the collection of sets C is represented as an attack graph. *An attack graph can be an exponentially succinct representation of a collection of sets.* Figure 6 shows an attack graph of linear size whose set of realizable sets is the power set of $\{s_1, \cdots, s_n\}$. Therefore, the minimum critical set problem is polynomially equivalent to the hitting set problem where the collection of sets C is represented as a labeled directed graph.

5.2 A Greedy Algorithm

Next, we describe a greedy algorithm *GREEDY-HITTING-SET* for the minimum hitting set problem. Let (C, S, K) be an instance of the hitting set problem. Let S' and C' be initially the empty set. The greedy algorithm executes the following steps until $C' = C$.

- Pick an element s out of the set $S \setminus S'$ that covers the maximum number of sets in the collection $C \setminus C'$. An element s is said to cover a set $S_1 \subseteq S$ iff $s \in S_1$.

- Let s be the element picked in the previous step and $C(s)$ be the collection of sets in C covered by s. Update S' and C' as follows:

$$S' \leftarrow S' \cup \{s\}$$
$$C' \leftarrow C' \cup C(s)$$

Let H_d be the d-th harmonic number $\sum_{i=1}^{d} \frac{1}{i}$, and $C(s)$ be the number of sets in the collection C that are covered by the element s.

Lemma 3 *GREEDY-HITTING-SET* is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max_{s \in S}\{|C(s)|\})$.

The proof of the lemma follows from the equivalence between the minimum hitting set and the minimum cover problem [2] and the proof of the approximation factor $\rho(n)$ for the greedy algorithm for the minimum cover problem [5]. Using the equivalence between the problems of finding a minimum critical set and a minimum hitting, set we can construct a greedy procedure (called *GREEDY-CRITICAL-SET*) for finding a critical set for the attack graph. Assume that we are given an attack graph $G = (S, E, s_0, s_s, L)$, where S is the set of states, $E \subseteq S \times S$ is the set of edges, $s_0 \in S$ is the initial state, $s_s \in S$ is the success state for the intruder, and $L : E \rightarrow \mathcal{A} \cup \{\epsilon\}$ is a labeling function. Moreover, assume that we can compute in polynomial time the function $\mu_G : \mathcal{A} \rightarrow \aleph$, where $\mu_G(a)$ is the number of realizable sets in the attack graph G that contain the attack a. Formally, $\mu_G(a)$ is equal to

$$|\{A' | a \in A' \text{ and } A' \in Rel(G)\}| \; .$$

Initially, let A' be the empty set and $G' = G$. The greedy algorithm *GREEDY-CRITICAL-SET* executes the following steps until G' is empty.

- Pick and element a from the set $\mathcal{A} \setminus A'$ that maximizes $\mu_{G'}(a)$.

- Let a be the element picked in the previous step. Update A' and G' as follows:

  $A' \leftarrow A' \cup \{a\}$
  Remove all edges labeled with a from G'

Lemma 4 *GREEDY-CRITICAL-SET* is a polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) = H(\max_{a \in \mathcal{A}}\{\mu_G(a)\})$.

Next, we explore conditions when the function μ_G can be computed in polynomial time. Assume that the attack graph G is a DAG. An argument for this was given in Section 4.3. Moreover, assume that each atomic attack is *used only once* on a path from the initial state s_0 to the success state s_s. This is not a unreasonable assumption because the attack graph edges are labeled with instantiations of attack templates shown in Section 4.3, e.g., a local-setuid-buffer-overflow attack on two different hosts are distinct in the attack graph. Such attack graphs are called *use once* DAGs. The following lemma is easy to prove.

Lemma 5 For an attack graph that is a use once DAG, the function μ_G can be computed in time that is linear in size of the attack graph.

6 Probabilistic Analysis of Attack Graphs

One way to incorporate probabilities into attack graphs is to choose a subset of states and make transitions out of those states probabilistic. Suppose that the graph has a state s with only two outgoing transitions. In a regular attack graph, the choice of which transition to take when the system is in state s is nondeterministic. However, we may have some empirical data that enables us to estimate that whenever the system is in state s, on average it will take one of the transitions four times out of ten and the other transition six remaining times. We can place probabilities 0.4 and 0.6 on the corresponding edges in the attack graph. Intuitively, probability of the transition $s \rightarrow s'$ represents the likelihood that the atomic attack corresponding to the transition will succeed. We call a state with known probabilities for outgoing transitions *probabilistic*. When we have assigned all known probabilities in this way, we are left with an attack graph that has some probabilistic and some nondeterministic states in it. We call such mixed attack graphs *probabilistic attack graphs*. We use probabilistic attack graphs to evaluate the reliability of a network. Note that probabilities of all the transitions might not be available because of lack of data, e.g., a new type of atomic attack.

Since the attack graph includes only those states and transitions that can lead to success states, it excludes some transitions that exist in the complete model M. These excluded transitions can have non-zero probability, so that the sum of probabilities of transitions from a probabilistic state will be less than 1. To address this problem, we must model the rest of M in some way. We add a "catch-all" *escape* state s_e to the attack graph. A probabilistic state s in the attack graph will have a transition to s_e if and only if in M there is a transition from s to some state *not* in the attack graph. The probability of going from s to s_e will be 1 minus the sum of the probabilities of going to other states. There are no transitions out of s_e except a self-loop (which preserves the totality of the transition relation τ).

In an attack graph containing the escape state s_e attacks are allowed to terminate in s_e. We will call them *escape attacks*, or attacks that were pre-empted by the intruder before he reached his goal.

Definition 3 A *probabilistic attack graph* or PAG is a tuple $G = (S_n, S_q, s_e, S, \tau, \pi, S_0, S_s, L)$, where S_n is a set of nondeterministic states, S_q is a set of probabilistic states, $s_e \in S_n$ is a nondeterministic escape state ($s_e \notin S_s$), $S = S_n \cup S_q$ is the set of all states, $\tau \subseteq S \times S$ is a transition relation, $\pi : S_q \to S \to \Re$ are transition probabilities, $S_0 \subseteq S$ is a set of initial states, $S_s \subseteq S$ is a set of success states, and $L : S \to 2^{AP}$ is a labeling of states with a set of propositions true in that state.

A probabilistic attack graph (PAG) distinguishes between nondeterministic states (set S_n) and probabilistic states (set S_q). Moreover, the sets of nondeterministic and probabilistic states are disjoint ($S_n \cap S_q = \emptyset$). The function π specifies probabilities of transitions from probabilistic states, so that for all transitions $s_1 \to s_2 \in \tau$ such that $s_1 \in S_q$, we have $P(s_1 \to s_2) = \pi(s_1)(s_2) > 0$. Thus, $\pi(s)$ can be viewed as a probability distribution on next states. Intuitively, when the system is in a nondeterministic state s_n, we have no information about the relative probabilities of the possible next transitions. When the system is in a probabilistic state s_q, it will choose the next state according to probability distribution $\pi(s_q)$.

Complete probability case. Assume that each transition in the attack graph is assigned a probability, i.e., there are no nondeterministic states. Let $G = (S, \tau, S_0, S_s, L)$ be the attack graph and P a function that assigns probabilities to transitions. The probabilities can be loosely interpreted as the probability of the atomic attack corresponding to the transition succeeding. We are interested in finding the reliability of the attack graph, i.e., the probability that the intruder will not succeed. We can view G as a Markov chain with S as its state space and $P(s_1 \to s_2)$ as its transition probability. Let $U : S \to \Re_+$ be the steady state probability of the Markov chain (see [7] for definitions and technical conditions). In this case, the reliability of the attack graph G is given by the following expression:

$$1 - \sum_{s \in S_s} U(s)$$

In other words, the reliability is the probability that in the "long run" the Markov chain will not be in a state in the set S_s. We wish to perform similar analysis on probabilistic attack graphs in the presence nondeterministic states.

6.1 Reliability

Assume that we are given a PAG $G = (S_n, S_q, s_e, S, \tau, \pi, S_0, S_s, L)$. Intuitively, we are interested in finding out the probability that the intruder will reach a success state starting from one of the initial states. Recall that in the absence of nondeterministic states we can compute this metric by using the steady state probabilities of the Markov chain. In presence of nondeterministic states the intruder will choose transitions in order to maximize his probability of succeeding. For example, if an intruder reaches a nondeterministic state s with transitions to $s_1, \cdots, s_k$, he will choose to transition to state s_i ($1 \le i \le n$) which will maximize his probability of reaching a success state. This idea can be "formalized" using concepts from the theory of Markov Decision Processes [1, 13].

Given a state s, the set of successors of s is denoted by $succ(s)$. Formally, $succ(s)$ is equal to $\{s' | (s, s') \in \tau\}$. First, we define a *value function* $V : S \to \Re^+$. For all $s \in S_s$, $V(s) = 1.0$. For all states $s \in S \setminus S_s$ the value function is iterated according to the following equations until convergence.

$$V(s) = \begin{cases} \max_{s' \in succ(s)} V(s') & \text{if } s \in S_n \setminus S_s \\ \sum_{s' \in succ(s)} P(s \to s') V(s') & \text{if } s \in S_q \setminus S_s \end{cases}$$

Let $V^\star$ be the value function after convergence. Intuitively, $\sum_{s \in S_0} V^\star(s)$ is the probability for the intruder to reach a success state if he "resolves" the nondeterminism to maximize the probability of succeeding. Therefore, the worst case reliability of the network is $1 - \sum_{s \in S_0} V^\star(s)$. This algorithm is known as *value iteration*. The justification of the value iteration algorithm is presented in the companion technical report [9].

Example 1 We implemented the value iteration algorithm in our attack graph post-processor and ran it on a slightly modified version of the intrusion detection example from Section 4. In the modified example, each attack has both detectable and stealthy variants. The intruder chooses which atomic attack to try next, and he has a certain probability of picking a stealthy or a detectable variant. We assigned imaginary probabilities of picking a stealthy attack variant as follows: 0.2 for sshd buffer overflow, 0.5 for ftp .rhosts, 0.05 for the remote login, and 0.8 for local buffer overflow. The intruder's goal is to obtain root access on host ip_2 while remaining undetected. Accordingly, the states where this goal has been achieved were assigned benefit value 1.0.

In this setup, the computed probability of intruder success is 0.2, and his best strategy is to attempt sshd buffer overflow on host ip_1, and then conduct the rest of the attack from that host. The only possibility of detection is the sshd buffer overflow attack itself, since the IDS does not see the activity between hosts ip_1 and ip_2.

Given this context, a system administrator can ask the following question:

Question 4: The deployment of which security measure(s) will increase the likelihood of thwarting an attacker?

Answer:

Installing an additional IDS component to monitor the network traffic between hosts ip_1 and ip_2 reduces the probability of the intruder remaining undetected to 0.025; installing a host-based IDS on host ip_2 reduces the probability to 0.16. Other things being equal, this is an indication that the former remedy is more effective.

7 Summary of Contributions and Future Work

Our foremost contribution is the automatic generation of attack graphs. Our key insight is that an attack is equivalent to a counterexample produced by off-the-shelf model checkers; the attack/counterexample is a witness to a violation of a safety property. By a small, but critical enhancement to an existing model checker, we can easily produce attack graphs automatically; moreover, these graphs are succinct and exhaustive. A by-product of this part of our work is showing, by example, what level of abstraction is appropriate for modeling attacks. We use simple state machine specifications to model not just intruder behavior (by a set of atomic attacks), but also normal system behavior, system administrator recovery actions, and connectivity (communication) between subsystems.

Our second most important contribution is support for a range of formal analyses of attack graphs. Security analysts use attack graphs informally for attack detection, defense, and forensics. In this paper, we explain how they can now use our minimization analysis technique on attack graphs to more precisely answer questions like "Which security measure should I deploy in order to thwart this set of attacks?" and "Which set of security measures should I deploy to guarantee the safety of my system?" We also explain (briefly in text and in detail in the technical report [9]) how to use probabilities to perform reliability analysis. By annotating attack graphs with probabilities, we can interpret them as Markov Decision Processes (MDP). Then, by using MDP algorithms such as value iteration, security analysts can more precisely answer questions like "Which attack will incur the most damage to my system?" and "Will deploying this intrusion detection system increase or decrease the likelihood of thwarting this type of attack?"

On the theoretical front, we plan to exploit the full power of model checking by exploring how to handle liveness properties, not just safety properties. For example, a property that states a *user can always access a particular file server* would be violated if the server is disabled due to a denial-of-service attack. On the practical front, we plan to conduct larger case studies to illustrate the usefulness of automatically generating attack graphs. We also intend to build a tool that merges our work on attack graphs with existing intrusion detection technologies. The tool is intended help security analysts evaluate and enhance the security of a network.

References

[1] E. Altman. *Constrained Markov Decision Processes*. Chapman & Hall/CRC, 1999.

[2] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computational System Sciences (JCSS)*, 21:136–153, 1980.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, Aug. 1986.

[4] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1991.

[6] M. Dacier. *Towards Quantitative Evaluation of Computer Security*. PhD thesis, Institut National Polytechnique de Toulouse, December 1994.

[7] R. Durrett. *Probability: Theory and Examples*. Duxbury Press, 2nd edition, 1995.

[8] M. Garey and D. Johnson. *Computers and Intractibility*. W.H. Freeman and Company, 1979.

[9] S. Jha, O. Sheyener, and J. M. Wing. Two formal analyses of attack graphs. Technical Report CMU-CS-02-109, Carnegie Mellon University, February 2002.

[10] NuSMV: a new symbolic model checker. *http://afrodite.itc.it:1024/ nusmv/*.

[11] R. Ortalo, Y. Deswarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, 25/5:633–650, Sept/Oct 1999.

[12] C. Phillips and L. Swiler. A graph-based system for network vulnerability analysis. In *ACM New Security Paradigms Workshop*, pages 71–79, 1998.

[13] M. Puterman. *Markov Decision Processes-Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, 1994.

[14] R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 156 –165, May 2001.

[15] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2002.

[16] SMV: a symbolic model checker. *http://www-2.cs.cmu.edu/ modelcheck/*.

[17] P. Stephenson. Using formal methods for forensic analysis of intrusion events - a preliminary examination. White Paper, available at *http://www.imfgroup.com/Document Library.html*.

[18] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, June 12-14 2000.

[19] A. Valdes and K. Skinner. Probabilistic alert detection. In *Recent Advances in Intrusion Detection (RAID)*, 2001.

A Appendix to Section 5

Assume that we are given an attack graph $G = (S, E, s_0, s_s, L)$. Moreover, suppose one can compute the set of realizable sets $Rel(G)$. Lemma 2 proves that the problem of finding whether the attack graph G has a critical set of size $\leq K$ *is the* hitting set problem with $C = Rel(G)$, $S = \mathcal{A}$, and K.

Next suppose we have an instance (C, S, K) of the hitting set problem. We will construct and attack graph $G' = (S', E', s'_0, s'_s, L')$, where $L' : E' \rightarrow S \cup \{\epsilon\}$, i.e., the set of attacks used in the attack graph G' is S. Moreover, the set of realizable sets $Rel(G')$ of the graph G' is the collection C. A critical set of size $\leq K$ of the attack graph G' is a hitting set for the collection C. Next, we describe the construction of G'. Let $C = \{C_1, \cdots, C_m\}$ be the collection of sets and $S = \{s_1, \cdots, s_n\}$ be the set. We make m copies $S^1, \cdots, S^m$ of the set S. The set of elements in S^i will be denoted by $\{s^i_1, \cdots, s^i_n\}$. The set of states S' in the attack graph G' is

$$\{s'_0, s'_s\} \cup S^1 \cup \cdots \cup S^m .$$

The initial state is s'_0 and the final state is s'_s. The set of edges E' and the labeling function L' are defined as follows:

- There is an edge from s'_0 to every state in the set $\{s^1_1, s^2_1, \cdots, s^m_1\}$, and label of the edge (s'_0, s^1_i) is s_1 if $s_1 \in C_i$, otherwise it is ϵ.
- For all $1 \leq i \leq m$ and $1 \leq j \leq n-1$, there is an edge (s^i_j, s^i_{j+1}), and the label of edge (s^i_j, s^i_{j+1}) is s_{j+1} if $s_{j+1} \in C_i$, otherwise it is ϵ.
- There is an edge from every state in the set $\{s^1_n, s^2_n, \cdots, s^m_n\}$ to the state s'_s, and labels of all these edges is ϵ.

The sizes of the sets S' and E' in the attack graph G' are $mn + 2$ and $2m + mn$ respectively. It is easy to see that $Rel(G')$ is equal to C, and $S' \subseteq S$ is a critical set of the attack graph G' iff S' is a hitting set for the collection C. Since the size of G' is polynomial in the size of the instance of the hitting set problem and the hitting set problem is NP-complete, the MCSA problem is NP-hard. Lemma 1 in [15] proves that MCSA is in NP. Therefore, MCSA is NP-complete. The next example illustrates our construction.

Note: The discussion given above also proves that the problem of finding a minimum set of measures whose adoption will make the network safe is also NP-complete. One can simply take the set of measures M to be the set of attacks $\mathcal{A}$.

Example 2 We give a short example to illustrate the reduction. Consider a set $S = \{s_1, s_2, s_3\}$. Suppose that the collection C consists of the following subsets:

$$\begin{aligned} C_1 &= \{s_1, s_2\} \\ C_2 &= \{s_2, s_3\} \\ C_3 &= \{s_2\} \end{aligned}$$

The attack graph G' corresponding to this problem is shown in Figure 7. The set of attacks is $\{s_1, s_2, s_3\}$. The set of realizable sets $Rel(G')$ is exactly the collection C. The set of attacks $\{s_1, s_2\}$ is critical because every path from s'_0 to the success state s'_s uses at least one edge with the label in the set $\{s_1, s_2\}$. Moreover, $\{s_1, s_2\}$ is a hitting set for the collection $C = \{C_1, C_2, C_3\}$.

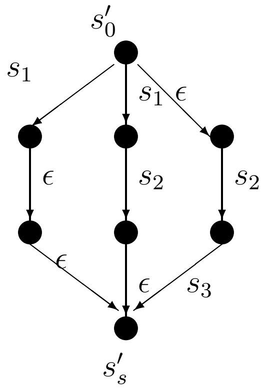

Figure 7. Attack graph corresponding to the collection C.