Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

FAST MODEL-BASED PENETRATION TESTING

Sankalp Singh
James Lyons
David M. Nicol

Coordinated Science Laboratory,
Department of Computer Science, and
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

ABSTRACT

Traditional approaches to security evaluation have been based on penetration testing of real systems, or analysis of formal models of such systems. The former suffer from the problem that the security metrics are based on only a few of the possible paths through the system. The latter suffer from the inability to analyze detailed system descriptions due to the rapid explosion of state space sizes, which render the models intractable for tools such as model checkers. We propose an approach to obtain statistically valid estimates of security metrics by performing repeated penetration testing of detailed system models. We make use of importance sampling techniques to help reduce the variance of our estimates, and achieve relative error bounds quickly. We validate our approach by estimating security metrics of a large model with more than 2^{1700} possible states.

1 INTRODUCTION

Security threats are an ever increasing problem to modern computing infrastructures. Attempts to characterize the security of a large networked system are the focus of several ongoing research efforts. On one end of the spectrum are approaches that perform penetration testing of an actual system implementation manually by Red Teams. Such an approach, while useful, generates only one of what may be many attack paths through a system. At the other end of the spectrum are approaches that try to build a model of the system, and then obtain comprehensive security metrics by analyzing the models. However, those approaches, which include formal model checking and attack languages, suffer from the degree to which state space explosion limits the amount of detail that the models can support. We propose an approach that we call *Model-Based Penetration Testing* (MBPT), which takes detailed information of the host configurations, attacks, vulnerabilities, critical assets, and connectivity to build complex models of the system that allow for automatic generation of repeated, independent attack paths.

Formal models for system security have been proposed to try to capture such properties with respect to networked system security. Model-checking tools then try to match the capabilities of the attacker with the current state to determine what transitions to apply to generate next states. The transitions usually result in increased access for the attacker in some portion of the system (Sheyner et al. 2002, Phillips and Swiller 1998).

One of the major difficulties of such model-checking approaches is that the state spaces of such models explode quite rapidly as the number of hosts, exploits available to the attacker, and vulnerabilities present on the hosts increase. For example, Sheyner et al. (2002) report handling a model with 229 bits of state. While a very large model from a state space perspective, the system that it represents is still quite small.

Such very large state spaces lend to themselves analysis via simulation techniques to estimate security metrics of the corresponding large systems. By generating paths through the state space, we are able to estimate metrics on the basis of observed paths.

An interesting metric associated with a path through such a state space is the number of steps used in reaching the goal. This is a measure of the level of hardness of an attack. Similarly, the total number of paths that exist could correlate with how easy or hard it is for an attacker to find one such path.

In this paper, we present a mechanism for building repeated random paths through the state space from which we can estimate metrics such as the total number of paths or the total number of paths of a particular length or less. The mechanism involves application of importance sampling techniques similar to those presented in (Heidelberger 1995, Shahabuddin 1994) to rapidly estimate metrics of interest.

Some background on formal models of security systems is presented in Section 2. Section 3 discusses in detail the sampling approach that was applied. Section 4 discusses details of the model-based penetration testing approach implemented here. Section 5 presents a sample model with slightly over 1700 bits of state and the results of using our estimation technique as compared to exact calculation of the same metrics. We conclude in Section 6 and discuss potential extensions to this work.

2 FORMAL MODELS OF NETWORKED SYSTEMS FOR SECURITY

Formal models for security of networked systems try to capture the relevant properties of a system by maintaining information about several different system aspects. In order to give the reader an understanding of security models, we present a quick description of how such models are constructed. As presented in (Sheyner 2004) the necessary information includes the set of hosts H, a connectivity relationship among hosts C, a trust relationship T among hosts, a model of the intruder I, a set of actions the intruder can take A, and a model of the intrusion detection system IDS. In addition to those, the state also includes the identifier of the attack to be attempted next, and the source and target hosts of that attempt.

Each host is modeled as having a list of services, a unique ID, a list of software running, and a list of host-specific vulnerabilities. The network connectivity is modeled as a ternary relation $C \subseteq H \times H \times P$ where P is the set of ports. Therefore $C(h_i, h_j, p) \rightarrow h_i$ can connect to h_j on port p. This relation allows the model to capture firewall rules in addition to physical network separations. $R(h_i, h_j)$ is used to specify that a network route exists from h_i to h_j. Trust is a binary relation $T \subseteq H \times H$ where $T(h_i, h_j)$ says that a user may log in to h_i from h_j without authentication or that "h_i trusts h_j." Services are a listing of all service names of every service present on the network. Every service name in a host definition comes from this list, and also allows specific ports to be assigned to specific services. The IDS maintains a list of the elements of the set $H \times H \times A$ that it can detect. Therefore $ids(h_i, h_j, a)$ says that attack a executed from h_i on h_j is detectable. Actions are specified by rules about how the system state is modified when actions are taken. They are then parameterized by the hosts they affect. Actions are taken between pairs of hosts. An action has preconditions in addition to rules that specify under what model state the action may be taken. The intruder maintains a store of information about the state of the system that has been uncovered so far. Specifically, the information includes the privilege level attained on each host, and perhaps other information such as the result of port scans or passwords. The privilege level on a host can take one of three values: $None$, $User$, or *Root*.

A model of this form is used throughout this paper.

3 RARE EVENT SIMULATION AND IMPORTANCE SAMPLING

The basic problem of rare event simulation is illustrated by the following example. (Some of the description below follows the discussion in Heidelberger (1995).) Let X be a random variable with probability density function p. Consider estimating the probability, γ, that X is in some subset A of $\mathbb{R}$ is $\gamma = E_p[1_{\{X \in A\}}]$, where the subscript p denotes the probability density assigned to the random variable X and $1_{\{x \in A\}}$ is the indicator of the set A. The standard approach for estimating γ by simulation would be to generate N samples, $X_1, X_2, \ldots, X_N$, using the density p, and then use $\hat{\gamma}_N = \frac{1}{N}\sum_{i=1}^{N} 1_{\{X_i \in A\}}$ as an unbiased estimator of γ ($E_p[\gamma_N] = \gamma$). The variance of $\hat{\gamma}_N$ is $\gamma(1-\gamma)/N$. If A corresponds to a rare event, i.e., γ is very small, the relative error of the estimate, defined as the standard deviation of the estimate divided by its mean, $RE_{\hat{\gamma}_N}$, is approximately $1/\sqrt{\gamma N}$. It is unbounded as $\gamma \rightarrow 0$. In order to minimize relative error, the sample size (N) must be large, and the probability γ should be large. Another view is that for a given γ and target relative error, one must increase N to achieve that relative error.

Importance sampling is a way of achieving small relative error using significantly smaller N than this standard approach calls for. We define another probability density function $p'(x)$, with $p'(x) > 0$ for all $x \in A$ such that $p(x) > 0$. Then, $\gamma = E_{p'}[1_{\{X \in A\}} L_{p'}(X)]$, where $L_{p'}$ is the likelihood ratio, i.e., $L_{p'}(x) = p(x)/p'(x)$, and the subscript in the expectation denotes sampling using density p'. We use p' to generate N samples $X_1, X_2, \ldots, X_N$, and $\hat{\hat{\gamma}}_N = \frac{1}{N}\sum_{i=1}^{N} 1_{\{X_i \in A\}} L_{p'}(X_i)$ is an unbiased estimator of γ. Thus, it is possible to estimate γ by simulating using a different probability density and then unbiasing the output by multiplying it by the likelihood ratio. The sampling with a different density is usually referred to as a "change of measure" and p' is called the "importance sampling density." It can be shown (Heidelberger 1995) that to reduce the variance of the estimator $\hat{\hat{\gamma}}_N$, we need to make the likelihood ratio $p(x)/p'(x)$ small on the set A. Since p is a given, this means that we should choose a p' such that $p'(x)$ is large on A, i.e., the change of measure makes A likely to occur. Another factor in the choice of the appropriate change of measure, especially with regard to the feasibility of simulation using the new estimator, is the cost of generating the samples.

Note that the above discussion on importance sampling also applies easily to the construction of estimators, which do not need a very large sample sizes for low variance, of $\alpha = E_p[W(X)1_{\{X \in A\}}]$ where W is some function on the domain of X,

4 FAST MBPT FOR ESTIMATING SECURITY

We now describe our methodology for performing penetration testing of formal models of networked systems for estimating security metrics for those systems.

Our approach broadly consists of constructing formal state/transition models of the networked system, as described in Section 2.

We build randomly constructed paths through the state-space of the model and estimate global security-related metrics as a function of the observed paths. Hence, the stochastic structure is not inherent in the semantics of security, but only in the selection of transitions. This approach can be viewed as a repeated penetration testing on a model of the system. We use importance sampling techniques to reduce the variance of our estimates by biasing transitions according to some heuristic and then taking the bias into account when building the estimate. We now provide a formal description of our methodology, where the security metric to be estimated is the *total number of attack paths* in the model that end up compromising a given asset H.

4.1 Problem Formulation

Let $\mathcal{S}$ denote the (finite) set of all possible states, i.e., the entire state space of the model. Note that this includes "illegal" states in which the attack to be attempted next is not enabled for the given source and target hosts, and the state of the network. Hence, if the encoding for the state requires n bits, then $|\mathcal{S}|$ is 2^n. For any $s \in \mathcal{S}$, $s.att$ denotes the variable in s that specifies the identifier of the next attack to be tried, $s.s$ and $s.t$ denote the variables in s that specify the source and target hosts of the attack respectively, and $s.rem$ denotes the rest of the fields contained in s, such as access privileges gained by the attacker and connectivity information.

Let $\mathcal{G}$ denote the set of goal states, i.e., $s \in \mathcal{G}$ iff executing $s.att$ with source $s.s$ and target $s.t$ would change $s.rem$ in a way that results in the compromise of asset H.

Let $\mathcal{S}_0$ denote the set of valid initial states, i.e., $s \in \mathcal{S}_0$ iff fields in $s.rem$ are set as specified according to the initial state of the system being modeled, and $<s.att, s.s, s.t>$ corresponds to an enabled attack under those conditions.

The term *sample path* is used to denote any infinite sequence $\{s_i, i > 0\}$, where each $s_i \in \mathcal{S}$.

A finite sequence $\{s_i\}_{i=1}^{l}$, $s_i \in \mathcal{S}$, is a *valid prefix* of length l iff all of the following are true:

1. $s_1 \in \mathcal{S}_0$, $s_l \in \mathcal{G}$, $s_i \notin \mathcal{G}$ for $i < l$,
2. for $1 \leq i \leq l$, $<s_i.att, s_i.s, s_i.t>$ corresponds to an enabled attack, and
3. for $2 \leq i \leq l$, $s_i.rem$ are the fields that result when $s_{i-1}.att$ is applied, to $s_{i-1}.rem$ with $s_{i-1}.s$ as the source, and $s_{i-1}.t$ as the target.

In other words, a valid prefix corresponds to a sequence of actual attacks that the attacker may execute, with the end result being the compromise of the asset H. Note that the length of a valid prefix is always finite, at least for the general class of models we consider. The execution of an enabled attack in state s in a valid prefix changes the state such that $<s.att, s.s, s.t>$ would not correspond to an enabled attack again if we continue making valid transitions, with each one acting on the changed state produced by the previous one. Hence, there is an underlying strict monotonicity property of valid transitions; which together with the finite size of the state space, that implies that l is finite.

Let $\mathcal{V}$ denote the set of all valid prefixes, i.e., the set of all actual attack paths.

A sample path is a *valid path* if the first l states in it form a valid prefix for some $l > 0$.

Let the sample space, Ω, be the set of all sample paths. We define a probability measure $\mathcal{P}$ on Ω, such that all of the sample paths in Ω are equally likely. Now, we can assign each element in $\mathcal{S}$ a unique identifier from the set $\{0, 1, \ldots, |\mathcal{S}|-1\}$. Hence, each state can be uniquely identified by a digit in the number system with base $|\mathcal{S}|$. We define a random variable X on Ω such that the image of the sample path $\{s_i, i > 0\}$ under X is the real number $(0.D_1D_2D_3\ldots)_{|\mathcal{S}|}$, where the subscript indicates that the number is expressed in the base $|\mathcal{S}|$, and D_i is the digit in base $|\mathcal{S}|$ associated with the state s_i. Henceforth, we suppress the subscript on real numbers in $[0, 1)$, and unless otherwise specified, the numbers are in base $|\mathcal{S}|$. It is easy to see that X describes a one-to-one correspondence between Ω and the set $[0, 1)$. Hence, under $\mathcal{P}$, X is uniformly distributed over $[0, 1)$, i.e., it has a probability density function $p(x)$, defined as $p(x) = 1$ if $x \in [0, 1)$ and $p(x) = 0$ otherwise.

Let $\mathcal{R} \subseteq \Omega$ be the set of all valid paths in Ω, and equivalently, let $\mathcal{A}$ be the corresponding subset of $[0, 1)$. Now consider an actual attack path of length l beginning in a valid initial state and ending with the compromise of asset H, in other words, consider a valid prefix. Here we use the term "valid prefix" to refer to the sequence (of length l) of states as described above, as well as the finite precision (l) real number corresponding to it, with the meaning clear from the context. Let the valid prefix be $(0.D_1D_2\dots D_l)$ for some D_i's in base $|\mathcal{S}|$. Now, the set of all valid paths with this prefix is simply the interval $[0.D_1\dots D_l00\dots, 0.D_1\dots(D_l+1)00\dots)$. The size of the interval is $|\mathcal{S}|^{-l}$. Henceforth, we use I_v to denote the interval associated with a valid prefix v, and l_v to denote the length of the valid prefix.

It is clear that the intervals corresponding to different valid prefixes are disjoint, since a number cannot have two different prefixes simultaneously. It was already argued above that the length of a valid prefix is finite, which in turn implies that the total number of valid prefixes is finite. Thus, $\mathcal{A}$ (and equivalently, $\mathcal{R}$) is a finite union of disjoint intervals corresponding to the different valid prefixes, i.e., $\mathcal{A} = \bigcup_{v\in\mathcal{V}} I_v$. Therefore, the total probability of $\mathcal{A}$ is the sum of the probability masses associated with all of the constituent intervals. Since the default probability density function $p(x)$ is 1 on $[0,1)$, the probability mass associated with an interval is the size of the interval. For a model of any reasonably sized system, the number of states $|\mathcal{S}|$ is huge, so the sizes of the intervals (and hence the probability mass) associated with valid prefixes decrease very rapidly with their lengths. In general, the mass of each interval would be much smaller than the number of such intervals (i.e., the number of valid prefixes). As a result, the total probability mass associated with $\mathcal{A}$ would be very small, and $\mathcal{A}$ represents a rare event under $p(x)$.

4.2 Estimation of Total Number of Attack Paths

Now, we show how to construct an estimator for T, the *total number of attack paths leading to compromise of the asset* H, which is the same as the total number of valid prefixes ($= |\mathcal{V}|$).

We define the function $W : [0,1) \to \mathbb{Z}_+$ that takes the value $|\mathcal{S}|^l$ for all points in an interval corresponding to a valid prefix of length l, and is zero for a sample path that is not valid. Therefore, we have

$$\begin{aligned}
&E_p[W(X)] \equiv E_p[W(X)1_{\{X\in\mathcal{A}\}}] \\
&= \int_{-\infty}^{\infty} W(x)1_{\{x\in\mathcal{A}\}}p(x)\,dx = \int_{x\in\mathcal{A}} W(x)p(x)\,dx \\
&= \sum_{v\in\mathcal{V}}\left(\int_{x\in I_v} W(x)p(x)\,dx\right) = \sum_{v\in\mathcal{V}}\left(\int_{x\in I_v} |\mathcal{S}|^{l_v}\,dx\right) \\
&= \sum_{v\in\mathcal{V}}\left(|\mathcal{S}|^{l_v}\int_{x\in I_v} dx\right) = \sum_{v\in\mathcal{V}} |\mathcal{S}|^{l_v}|\mathcal{S}|^{-l_v} \\
&= |\mathcal{V}| = T, \qquad (1)
\end{aligned}$$

where the introduction of summation is possible because $\mathcal{A}$ is a finite union of disjoint I_v's. Hence, we estimate T by estimating $E_p[W(X)]$.

As described in Section 3, a standard unbiased estimator for T would be $\hat{T}_N = \frac{1}{N}\sum_{i=1}^{N} W(X_i)$, where $X_1, X_2, \dots, X_N$ are N random samples drawn from $[0,1)$ (or, equivalently, from Ω) with density $p(x)$. However, since $W(X)$ is non-zero only on the rare set $\mathcal{A}$, the variance of the estimator would be very large. Using analysis very similar to that used above, we can show that

$$Var(\hat{T}_N) = \frac{1}{N}\left(\sum_{v\in\mathcal{V}} |\mathcal{S}|^{l_v} - T^2\right), \qquad (2)$$

which would be very large because $|\mathcal{S}|$, the size of the state space, is a very large number for a model of any reasonably sized system. For instance, in the example system considered in Section 5.1, $|\mathcal{S}|$ is slightly more than 2^{1700}. Therefore this estimator would clearly require a prohibitively large N for the relative error to be small enough, and as a result cannot practically be used for determining T by simulation. The problem is well-suited for the application of importance sampling techniques.

The following is the algorithm we use for generation of a sample point in $[0,1)$.

```
X-Sampling-Algorithm
1.  Initialization
      Create a variable $s$ of type state and initialize s.rem to the initial setup for the system being modeled. Initialize $p_{curr}$ to 1, and $l_{curr}$ to 0.
2.  Pick a transition
      Generate the list $L$ of enabled attacks (each represented by an <attack, source, target> tuple) given the system state as represented by s.rem.
      a.  If $L$ is not empty
            i.    Assign a non-zero probability to each enabled attack using some heuristic, while ensuring that the probabilities sum up to 1.
            ii.   Pick one of the enabled attacks based on the probabilities assigned above. This choice determines the current digit of the sample. Let $q$ be the probability that was assigned to the chosen attack.
            iii.  Update the s.rem according to the effects of the chosen attack.
            iv.   Update $p_{curr}$ by multiplying its current value by $q$.
            v.    Increment $l_{curr}$ by 1.
```

v. Check *s.rem* to see if the attacker objective was satisfied. If yes, then we have reached a goal state, and exit with an indication of success, along with the current value of p_{curr} and l_{curr}; otherwise, we *go back* to the beginning of step 2.

b. ELSE
Exit with an indication of failure because the current sample is not in A.

The above algorithm either returns a valid prefix v (actually, it returns l_v and the probability mass associated with I_v under the new density), or announces that the current sample is not in $\mathcal{A}$. If a valid prefix is returned, we consider the remaining digits to have been chosen such that all of the $|\mathcal{S}|$ digits are equally likely to be chosen at each stage. Similarly, if we are informed that the sample is not in $\mathcal{A}$, we consider the first l_{curr} digits to have been chosen with probability p_{curr}, and the remaining digits to have been chosen such that all of the $|\mathcal{S}|$ digits are equally likely to be chosen at each stage. These considerations result in a new probability density function $p'(x)$. It is clear from the description of the algorithm and assumptions regarding the subsequent choices that $p'(x)$ is a valid probability density function, i.e., $\int_{-\infty}^{\infty} p'(x)dx = \int_0^1 p'(x)dx = 1$. The reason is that we start with a probability mass of 1, and at each step divide the available mass completely among the choices available at that step. Also, if $\mathcal{V}$ is not empty (which implies that $\mathcal{A}$ is not empty), the entire probability mass is assigned to $\mathcal{A}$. In other words, if $Pr[X \in A] \neq 0$ then we *always* generate a valid path (corresponding to a point in $\mathcal{A}$).

Now, according to the description above, the probability mass associated with the interval I_v, for a given valid prefix v, is the product of probabilities of the transitions chosen by the sampling algorithm at each stage when it generated v. Let us call this mass $p_v \prod_{i=1}^{l_v} q_i^v$. The mass is divided uniformly in the interval. Hence, the probability density at $x \in I_v$

$$p'(x) = \frac{p_v}{sizeofI_v} = \frac{p_v}{|\mathcal{S}|^{-l_v}} = \left(\prod_{i=1}^{l_v} q_i^v\right)|\mathcal{S}|^{l_v}. \tag{3}$$

Hence, $p'(x)$ is a valid probability density function, and $p'(x) > 0$ for all $x \in \mathcal{A}$. Therefore, $p'(x)$ is a valid change of measure.

Now, as described in Section 3 and in Equation 1, $T = E_p[W(X)1_{\{X \in \mathcal{A}\}}] = E_{p'}[W(X_i)1_{\{X \in \mathcal{A}\}}L_{p'}(X)$. Hence, we build the following unbiased estimator for T:

$$\hat{\hat{T}}_N = \frac{1}{N}\sum_{i=1}^{N} W(X_i)1_{\{X_i \in A\}}L_{p'}(X_i)$$

where $X_1, X_2, \ldots, X_N$ are N random sample points drawn from $[0,1)$ with density $p'(x)$. Those points are generated using the X-SAMPLING-ALGORITHM described above. If X_i is not in $\mathcal{A}$, we do not need the actual value of X_i, since the function $W(X_i)1_{\{X_i \in \mathcal{A}\}}$ is zero at such points. For a point $X_i \in \mathcal{A}$, we see from Equation 3 that $L_{p'}(X_i) = p(X_i)/p'(X_i) = 1/(|\mathcal{S}|^{l_v} \prod_{i=1}^{l_v} q_i^v)$, where v is the valid prefix of X_i. Therefore, for such a point,

$$W(X_i)1_{\{X_i \in A\}}L_{p'}(X_i) = |\mathcal{S}|^{l_v} \frac{1}{|\mathcal{S}|^{l_v} \prod_{i=1}^{l_v} q_i^v} = \frac{1}{\prod_{i=1}^{l_v} q_i^v}. \tag{4}$$

Hence, for a point in $\mathcal{A}$ generated by X-SAMPLE-ALGORITHM, we generate the value of $W(x)1_{\{x \in A\}}L_{p'}(x)$ using Equation 4, avoiding any calculation that involves $|\mathcal{S}|$, which might have caused precision problems in the implementation. As already noted, the value is 0 if $x \notin \mathcal{A}$.

Using techniques similar to the ones in Equation 1 and Equation 2, it can be shown that

$$Var(\hat{\hat{T}}_N) = \frac{1}{N}\left(\sum_{v \in \mathcal{V}} \frac{1}{\prod_{i=1}^{l_v} q_i^v} - T^2\right). \tag{5}$$

It is obvious from Equation 5 that $Var(\hat{\hat{T}}_N)$ varies greatly with the heuristic used to decide the q_i^v's, i.e., the heuristic that chooses the attack from among the set of enabled attacks at any stage of the X-SIMULATION-ALGORITHM. For example, a heuristic designed such that it ends up assigning equal probability masses to all of the valid prefixes results in $Var(\hat{\hat{T}}_N)$ being zero, and only one sample would be needed to obtain the exact answer. However, the catch is that it is practically impossible to set the appropriate q_i^v's for that heuristic without prior knowledge about all the valid prefixes. Nevertheless, comparing Equation 5 with Equation 2, we see that even the very simple heuristic that chooses all of the enabled attacks at each stage with equal likelihood would produce a drastic reduction in variance. For this heuristic, q_i^v is 1/(*number of attacks enabled at the* i^{th} *stage of the attack path).* Since the number of enabled attacks at any point would be a much smaller number than $|\mathcal{S}|$, $\prod_{i=1}^{l_v} q_i^v \ll |\mathcal{S}|^{l_v}$, which implies that $Var(\hat{\hat{T}}_N) \ll Var(\hat{T}_N)$. More intelligent heuristics that approximate the perfect one more closely would result in even better accuracy. We have experimentally evaluated some of those heuristics in Section 5.

5 METHODOLOGY EVALUATION

To evaluate the efficacy of the previously outlined approach, we tested the method on a sample model. The results, described in more detail in Section 5.2, were then compared to measures for which the exact values could be calculated to evaluate the approache's accuracy. In addition the execution-time for both the exact answer calculations and our estimates were also made.

5.1 Example Models Used

In order to evaluate the approach, we had to create a network model to study. Our goal in this work was to study a model significantly larger in total state space than model-checking techniques would allow. Using the model presented in (Sheyner et al. 2002) as a starting point, we combined the models presented in (Sheyner 2004) and further extended them to generate a model with a state space of over 1700 bits. A state space of that size is well in excess of what state of the art model checkers are capable of handling. A summary of the model follows.

We modeled a network as a collection of hosts, $h_0 ... h_{(NumHost-1)}$. Each host has associated with it a privilege level, *plvl*, and an array of Boolean attributes and services, s_i. The services that were included in the model are the conjunction of the services outlined in (Sheyner et al. 2002, Sheyner 2004) and an additional service that indicates which hosts the intrusion detection system (IDS) is running. The services are briefly outlined in Table 1, and are used in Figure 1. Our model used a total of 13 hosts, each with a different configuration of the services. As shown in the figure, the network studied in the model consists of a web server separated from a small corporate network consisting of two different and separated labs. Each lab can communicate with a load-balancing proxy cluster for communications with the database server. The database server only allows connections from machines in the proxy cluster, and the machines of the proxy cluster all require that ssh connections be utilized. The IDS system is deployed to monitor all connections to and from the proxy cluster, including the database server. The overall goal for the attacker is to gain root privilege on the database server without being detected by the IDS.

The network connectivity is expressed as a connectivity matrix consisting of 7 boolean entries for each pair (h_i, h_j). The first entry indicates whether h_i is physically connected to h_j. The rest indicate whether host h_i can open a connection to h_j on a particular port. That does not mean that a server is listening on that port, but rather that a TCP connection can be prevented without interference from a firewall (for example). The connection types for our sample model are *Physical*, *Port80*, *PortICQ*, *PortSQL*, *PortSSH*, *PortFTP*, and *PortIDS*. This connectivity information is invariant and does not change throughout the course of the model execution. Our approach does not require that invariance. It was merely a by-product of the attack set used. The intruder in our model initially begins with *root* privilege on host 0 and *none* on any other host. The intruder may execute any attack from a source to a target provided that the attack's preconditions are met. An attacker's privilege level can only increase, no event can cause the attacker to lose any newly attained privilege. Thus the attacker is assured of finding a path to the target eventually, if one exists.

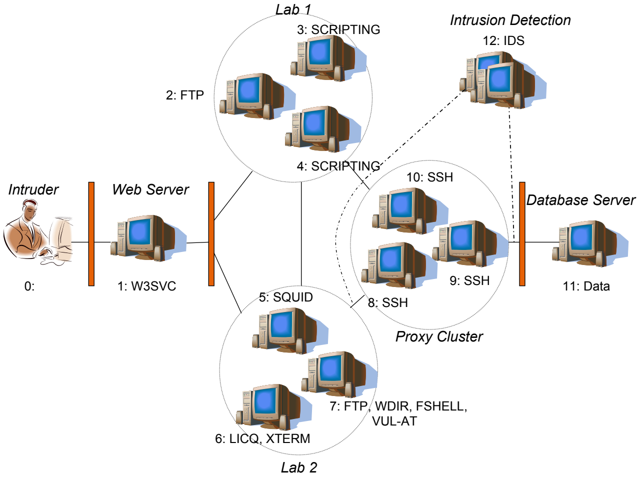

Figure 1: Network Configuration Used for Sample Model

Table 1: Host State Attribute Descriptions

Attribute	Description
w3svc	IIS web service is running
squid	*Squid* proxy is running
licq	*LICQ* running on host
scripting	HTML scripting is enabled on this host
vul-at	*at* executable vulnerable to overflow
ssh	ssh service is running
ftp	ftp service is running
data	database server running
wdir	ftp directory is writeable
fshell	ftp user has executable shell
xterm	xterm executable is vulnerable to overflow
IDS	Intrusion detection process running

11 attacks are possible in this model. Each attack has its own set of preconditions based on the source and target of the attack and the state of the model. When satisfied, those preconditions enable the attack. The effects of an attack alter model state when the attack is used by the intruder. We refer the reader to

(Sheyner 2004, Sheyner et al. 2002) for specifications of 9 of the 11 attacks in our system, and only present the preconditions and postconditions for our 2 new attacks.

The first attack is used to model the case in which there is an exploitable vulnerability in the IDS system that can cause it to fail or crash in such a way that the IDS is at least temporarily disabled. The other possible attack that is captured here is an attack that spoofs the IDS in such a way that it fails to detect further actions.

```
Attack IDS Circumvent
  intruder preconditions
    $plvl(src) = user$              User privilege on source
    $plvl(trg) = none$              No privilege on target
  network preconditions
    $IDS_{trg}$                     IDS is running on target
    $R(src, trg, PortIDS)$          Src and trg are connected
                                    on portIDS
  intruder effects
    $\emptyset$                     No privilege changes
  network effects
    $IDS_{trg} = false$             IDS no longer working
end
```

The second attack is an exploit on the database server itself. The attacker can send a carefully and maliciously crafted packet to the database server causing the server to fail in a manner that grants root privilege on the target host.

```
Attack SQL Malformed Packet
  intruder preconditions
    $plvl(src) \geq user$           User or greater privilege
                                    on source
    $plvl(trg) = none$              No privilege on target
  network preconditions
    $Data_{trg}$                    Database is running on
                                    target
    $R(src, trg, PortIDS)$          Src and trg are connected
                                    on portSQL
  intruder effects
    $plvl(trg) = root$              Root privilege acquired
  network effects
    $\emptyset$                     No network changes
end
```

In our model, the intruder's goal is to achieve root privilege on host 11, which contains the database server, without being detected by the Intrusion Detection System.

The IDS system can detect and sound alarms for all versions of the sshd buffer overflow and the remote login attacks (defined in (Sheyner 2004)). The IDS also maintains information about which links in the network graph it monitors. In our example, as shown in Figure 1, the IDS monitors all the links to the proxy cluster and database server.

5.2 Experimental Evaluation

In order to evaluate the sampling approach outlined in previous sections, we implemented a simulator to analyze the model from Section 5.1. Tools such as NuSMV, which were used in the analysis of similar models by (Sheyner 2004), have simulation functions; however, they still generate the full state space prior to the simulator execution. Significant effort would have been required to make our sampling approach work within the framework of such a tool. Instead, we implemented our own simulator in C++ to collect the measures we were interested in on the example model. To obtain exact answers to compare to the results of our estimations, we implemented a depth-first search algorithm in addition to our sampling strategies. The experiments were conducted on an AthlonXP 2400+ with 256MB of RAM. The metrics estimated were the total number of unique attack paths in the model of length less than or equal to a given number. For each simulation experiment, samples were repeatedly generated until either a 5% relative error (95% confidence) was obtained or 400,000 samples were collected. Note that the error bound was achieved for all path lengths except infinity.

The depth-first search works by traversing all paths available to the attacker. As the number of total paths increases the depth-first strategy becomes increasingly useless. For example in Table 3 one can see that the running time of DFS increases significantly as the maximum path length increases. By contrast, the time taken to obtain an estimate for the value grows much slower.

To do the sampling as outlined above, we needed a way to weight transitions that would drive the simulation towards the rare set of paths more often. We used a total of three different heuristic functions to guide our simulation. The first was attack severity, in which we did an ad hoc weighting of the attacks based on our opinion of which attacks were more likely to bring the simulation closer to the goal state. The second was a distance function that compared the distance from the goal machine to the target of the attack and tried to guide the simulation towards attacks that were topologically closer to the goal machine. The third was a combination of both functions to determine the weights to assign transitions at every step of the simulation. Results for all three heuristic functions are presented in the tables.

We wanted to estimate the total number of paths in the model from the initial state to the goal state. The number of such paths however is quite large and our DFS algorithm was unable to calculate an exact answer in a reasonable time (we stopped it after 60 hours). We therefore also included results for smaller path lengths so that a comparison could be made. Limiting the length of

Table 2: Number of Attack Paths Calculated by Simulator and DFS on Example Model

	Simulator						
Path Length	Distance-based	95% Conf. Interval	Attack Severity-based	95% Conf. Interval	Combination-based	95% Conf. Interval	DFS (Exact)
$\leq$6	203.915	±10.193	217.296	±10.863	210.58	±10.52	210
$\leq$8	81729	±4086	83906.8	±4195.12	79199.6	±3959.87	82704
$\leq$10	1.6699x10^{7}	±859289	1.65733x10^{7}	±828592	1.6874x10^{7}	±843586	1.65101x10^{7}
$< \infty$	1.4657x10^{10}	±2.062x10^{9}	1.4953x10^{10}	±1.354x10^{9}	2.978x10^{10}	±3.76x10^{10}	-

Table 3: Execution Time (in Seconds) for Simulator and DFS on Example Model

Path Length	Simulator			DFS
	Distance-based	Attack Severity-based	Combination-based	
$\leq$6	55.021	27.689	8.563	0.177
$\leq$8	80.101	43.620	40.10	17.928
$\leq$10	182.623	104.22	504.7	2076.03
$< \infty$	720.9	740.3	890.19	-

paths is also useful in the sense that, an attacker would try to compromise the system using a smaller number of attacks, rather than compromising all elements of the system prior to reaching the goal. We therefore ran both the simulator and the DFS for maximum path lengths up to 10.

The results show two things that are worthy of note. Fairly accurate estimates were obtained in reasonably short simulation times. The second is that none of the fixed heuristic functions that we utilized was best for all path lengths. As the maximum path length increased heuristics that were more aggressive at driving the simulation towards the goal state faster took longer to converge on a 5% error bound. The reason stems from the results from Section 4, where we showed that the ideal heuristic would assign roughly equal weights to all valid paths. Clearly, a heuristic that tends to generate shorter paths will not always be of that form for varrying maximum path lengths. This explains the exceptionally rapid degradation in the performance of the combined heuristic with the increase in allowed path length; the combined heuristic is very heavily biased towards short path lengths.

6 CONCLUSION

This paper demonstrates a technique for security quantification by performing repeated penetration testing of detailed system models. We provide a precise mathematical formulation of how to use importance sampling techniques to estimate security metrics, such as the total number of attack paths that lead to the compromise of a given asset. We have successfully analyzed a detailed model possessing over 1700 bits of state, whereas previous model-checking approaches have not been able to achieve anywhere near that level of complexity. We believe this estimation technique can scale to larger and even more complex models.

Clearly, the results show that the choice of heuristic function used to guide a simulation has a large effect on the convergence of the estimate, and therefore the running time as well. The choice of heuristic functions presented in this paper shows simply that what is good for one measure is not necessarily good for all measures. Further work is needed how to chose heuristic functions for security models.

Computation time can be further reduced by running simulations in parallel, since there would be no dependency between simulation runs. However, such parallel processing may not possible with a model checker.

Other directions for future work include use of other variance reduction techniques, such as structured sampling, in combination with importance sampling. The effects of unknown vulnerabilities on the system are also of great importance with respect to system security. The model and approach could be extended to account for such uncertainties.

ACKNOWLEDGMENTS

This research was supported in part by DARPA Contract N66001-96-C-8530, NSF Grant No. 0086096, NSF Grant CCR-0209144, and Department of Energy contract DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. In addition this project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security.

We would also like to thank Jenny Applequist for her editorial assistance.

REFERENCES

Heidelberger, P. 1995. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simluations* 5 (1): 43–85.

Phillips, C., and L. Swiller. 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*, 71–79. ACM Press.

Shahabuddin, P. 1994. Importance sampling for the simulation of highly reliable Markovian systems. *Management Science* 40 (3): 333–352.

Sheyner, O. 2004. *Scenario graphs and attack graphs*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J. M. Wing. 2002. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Syposium on Security and Privacy*, 273–284. IEEE Computer Society.

AUTHOR BIOGRAPHIES

SANKALP SINGH is a Ph.D. student in the Department of Computer Science at the University of Illinois, Urbana-Champaign. His research interests include security quantification and validation. He received his B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Kanpur in 2001 and his M.S. in Computer Science from the University of Illinois in 2003. His e-mail address is <sankalps@crhc.uiuc.edu>.

JAMES LYONS is a Ph.D. student in the Department of Computer Science at the University of Illinois, Urbana-Champaign. His research interests include networked systems security. He received his B.S.E. in Computer Science and Engineering from the University of Pennsylvania in 2000 and his M.S. in Computer Science from the University of Illinois in 2003. His e-mail address is <jlyons@crhc.uiuc.edu>.

DAVID M. NICOL is Professor of Electrical and Computer Engineering at the University of Illinois, Urbana-Champaign, and member of the Coordinated Sciences Laboratory. He is co-author of the textbook *Discrete-Event Systems Simulation*, and served as Editor-in-Chief at ACM TOMACS from 1997-2003. He is the General Chair of the 2004 Conference on Principles of Advanced and Distributed Simulation, and the General Chair of the 2006 Winter Simulation Conference. From 1996-2003 he was Professor of Computer Science at Dartmouth College, where he served as department chair, and at the Institute for Security Technology Studies served as Associate Director for Research and Development, and finally as Acting Director. From 1987-1996 he was on the faculty of the Computer Science department at the College of William and Mary; 1985-1987 he was a staff scientist at the Institute for Computer Applications in Science and Engineering. He has a B.A. in mathematics from Carleton College (1979), an M.S. (1983) and Ph.D. (1985) in computer science from the University of Virginia. His research interests are in high performance computing, performance analysis, simulation and modeling, and network security. He is a Fellow of the IEEE. His e-mail address is <nicol@crhc.uiuc.edu>