Network Vulnerability Assessment using Bayesian Networks

Yu Liu[a] and Hong Man[a]

[a]Department of Electrical and Computer Engineering, Stevens Institute of Technology, Castle Point on Hudson, Hoboken, NJ 07030, USA

ABSTRACT

While computer vulnerabilities have been continually reported in laundry-list format by most commercial scanners, a comprehensive network vulnerability assessment has been an increasing challenge to security analysts. Researchers have proposed a variety of methods to build attack trees with chains of exploits, based on which post-graph vulnerability analysis can be performed. The most recent approaches attempt to build attack trees by enumerating all potential attack paths, which are space consuming and result in poor scalability. This paper presents an approach to use Bayesian network to model potential attack paths. We call such graph as "Bayesian attack graph". It provides a more compact representation of attack paths than conventional methods. Bayesian inference methods can be conveniently used for probabilistic analysis. In particular, we use the Bucket Elimination algorithm for belief updating, and we use Maximum Probability Explanation algorithm to compute an optimal subset of attack paths relative to prior knowledge on attackers and attack mechanisms. We tested our model on an experimental network. Test results demonstrate the effectiveness of our approach.

Keywords: network vulnerability, attack graph, Bayesian network

1. INTRODUCTION

Vulnerability assessment on network level is critical for an enterprise management to plan their defence strategies, and devise appropriate budget for maintaining security of their network. It requires security analyst to perform quantitative vulnerability assessment on a network of hosts. A variety of scanning tools, such as NESSUS,[1] ISS Internet Scanner,[2] and Tripwire,[3] have been developed to identify vulnerabilities exhibited on a single host or multiple hosts on a network. However, existing tools lack the capabilities of correlating different vulnerabilities in view of attacks. Since computer attacks usually take multiple steps and involve multiple hosts, constructing attack graph with chains of exploits complements scanning tools and explores the intra-relationship among vulnerabilities of different activities within a host, and the inter-relationship among vulnerabilities of different hosts across the network. Moreover, attack graph provides groundwork for security analysts to evaluate network vulnerability from a global view.

Our research aims to develop a systematic and automated approach for building attack graphs. We apply Bayesian network (also called belief network) to construct our attack graph and model all potential atomic attack steps on a given network. Within a Bayesian attack graph, each node represents a single security property violation state; each edge corresponds to an exploitation of one or more exhibited vulnerabilities and each path represents a series of exploits that can signify a potential attack.

Our application of Bayesian network to attack graph is motivated by three reasons. First, Bayesian network provides causal relationship between random variables that takes value from given domains. This draws similarities to attack graph. Given an edge in an attack graph, the state of a destination node is conditionally dependent on the state of its source node. That is to say the success of a subsequent atomic attack is conditionally dependent on the success of previous atomic attack state. Second, constructing attack trees to enumerate all possible attack paths is space consuming. Bayesian network, on the other hand, provides a more compact representation of attack graph, yet still keeps all attack paths information. This results in better scalability for mid to large-scale networks. Third, Bayesian network, also known as belief network provides a formalism for reasoning about partial beliefs under conditions of uncertainty. The uncertainties in an attach graph embed in weight assignments to the edges of the graph. For example, one can define the probability of a successful

Further author information: E-mail: {yliu, hman}@stevens.edu exploit as edge weight. Edge weights were assigned subjectively in most recent works. Some models alleviated such subjectiveness by building attacker profiles. However, domain expert knowledge is still needed for the actual weight assignments. Bayesian network can incorporate all these hypothesis beliefs and provide a formal inference method to formulate conclusions. More importantly, if appropriate training data becomes available, Bayesian network can systematically and automatically learn edge weights from data. Hence our approach can mitigate subjectiveness in vulnerability assessment when regular network activity records can be processed and reformatted to support the training of the proposed model. Because the intention of the paper is to introduce the concept of Bayesian attack graph, the training process is not a major focus here. Detailed discussion on training Bayesian Network can be found in[4] and[5]

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 outlines our approach for building Bayesian attack graph. Section 4 describes post-attack graph analysis using belief network inference. Section 5 presents an experiment on a small Linux-based network. Experimental results are presented and discussed. Section 6 is devoted to concluding remarks.

2. RELATED WORK

Dacier et al.[6] propose a "privilege graph" for modelling attacks with the vulnerabilities exhibited in a computer system. Ortalo et al.[7] presented experimental results using this model. Each node in a privilege graph represents a set of privileges (i.e. read, write, and execution abilities) owned by a user or a group of users. Each edge is a directed arc representing a vulnerability that can be exploited from a source node to a target node. A weight is assigned to each arc to represent the amount of effort that attackers must spend to exploit a vulnerability exhibited on the target node. Attack state graph is then constructed to characterize the state transition during the exploitation of the privilege graph. Each node in the attack state graph represents the privileges owned by the attacker thus far. In the post graph analysis, attack state graph is transformed into a Markov model based on the assumption that the probability distribution of the amount of effort an attacker willing to spend is exponential, given the success rate in an elementary attack. The mean effort to failure (METF) is calculated to indicate the average time for a system to fail in the security realm, i.e., the average time for a target state to be compromised. As shown in the experiment, METF is not always computable due to the failure of constructing the attack state graph, especially when the number of paths between the attacker and the target increases.

Phillips and Swiler[8] defined an attack graph in a more general way, where any state that corresponds to a security property violation (some may not lead to the compromise of a privilege level) will be modelled. Swiler et al.[9] later implemented this idea and built a computer-attack graph generation tool. Each node in their attack graph represents a possible system state which includes level of penetration by an attacker. Each edge is a directed arc representing a change of state caused by a single attack step. The weight of each edge represents a single security metric or a function of multiple metrics, such as probability of success, average time to succeed, or a cost/effort level of an attacker. Attack graph is constructed by matching the current (node) state against a set of predefined attack templates with forward search from the initial state. The attack graph is transformed to a tree structure by enforcing a partial ordering for vulnerability acquisitions. The size of the attack graph is contained at a sacrifice of partial attack paths information. In the post graph analysis, the set of near-optimal shortest paths is computed to provide a measure of overall system security. The near-optimal shortest paths represent the set of paths that the attacker might realistically consider.

Ritchey and Ammann[10] used model checking techniques to model attack states and produce attack graph for a network of hosts. Model checkers can be used to construct attack graph backward, i.e. from a target state back to an initial state. This results in smaller graph size compared with forward search, since vulnerabilities that are irrelevant to the goal of an intruder are never explored. In their work, model checker SMV is used to obtain a single path tree corresponding to one complex attack, where multiple steps are involved. Sheyner et al.[11] used a modified version of model checker, known as NuSMV (a newer version of SMV), which produces attack graphs representing all possible attacks on a networked system. These attack graphs are essentially similar to,[9] where any path in the graph from a root node to a leaf node shows a sequence of atomic attacks that an intruder can employ to achieve his goal. Two formal analyses of their graphs are presented in.[12] The minimization analysis determines a minimal set of atomic attacks that must be prevented to guarantee the safety of a system or a network. The probabilistic analysis is then presented by annotating attack graphs with probabilities, and then

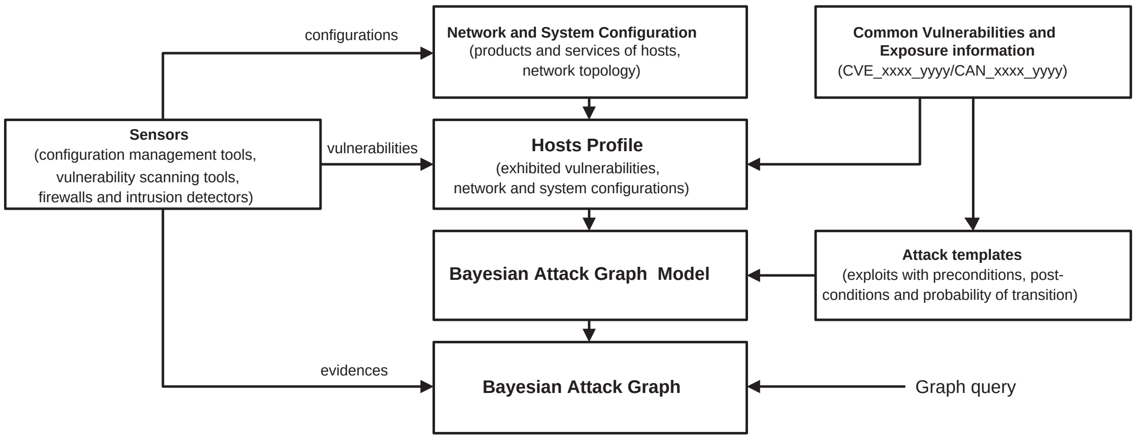
configurations
Network and System Configuration
(products and services of hosts, network topology)
Common Vulnerabilities and Exposure information
(CVE_xxxx_yyyy/CAN_xxxx_yyyy)
Sensors
(configuration management tools, vulnerability scanning tools, firewalls and intrusion detectors)
vulnerabilities
Hosts Profile
(exhibited vulnerabilities, network and system configurations)
Bayesian Attack Graph Model
Attack templates
(exploits with preconditions, post-conditions and probability of transition)
evidences
Bayesian Attack Graph
Graph query

Figure 1. Block Diagram of The Model

interpreting them as Markov Decision Process (MDP). They compute the worse case success probability of an intruder to reach a goal state. The worst case is defined with respect to the side of defence, hence it is the best strategy for intruder to choose the initial atomic attack that will maximize his probability of success.

Ammann et al.[13] revisited the work of Swiler et al. and argued that more useful information can be produced by monotonicity assumption of attacker capabilities (i.e. attacker never has to backtrack) and bypassing the attack tree step. Hence, their attack graphs provide better scalability. In the post graph analysis, they propose three algorithms, findMinimal, findAll, and findShort, to find a minimal attack (i.e., an attack that can be launched with the least number of steps from the initial state), all attack paths and a partial attack paths set corresponding to shortest paths. However, similar to the work Swiler et al., their post-graph analysis does not provide a quantitative assessment of security on a network.

Only Dacier provides a quantitative security evaluation of an overall network by computing MEFT in their post graph analysis. Our model takes advantage of this concept, and we consider posterior probability of Bayesian networks as a general security metric that represents a quantitative vulnerability assessment of a given network.

3. OUR APPROACH

Typical types of security violation states include security breaches in availability, confidentiality, integrity, and access privileges at hosts of a network. Security analysts can define their own set of security breaches according to their network security policies. Each security breach may result in one or more violation states in the network. In our model, we only concern the violation states caused by the exploitations of exhibited vulnerabilities in the network.

Figure 1 presents the block diagram of our model. Each host in a network is profiled with network and system configurations, and exhibited vulnerabilities. They are identified through a list of off-the-shelf tools. Similar to most public available vulnerability bulletins, such as in ICAT Metabase,[14] we classify vulnerabilities into 9 types: input validation error with boundary overflow, input validation error with buffer overflow, access validation error, exceptional condition handling error, environmental error, configuration error, race condition, design error and other. This eases our hosts profiling and updating process. We construct generic, atomic attack templates to describe exploitations of vulnerabilities. Each template contains precondition and postcondition of an atomic attack, along with security metric(s) information. An exploit is an instantiation of an actual attack template. A successful exploit will lead to a security breach. Since we build attack graphs using Bayesian networks, we assign the probability of a successful exploit as edge weight. Figure 2 shows four attack template examples. We use the standardized CVE and CAN numbers as our attack template id numbers.

```
CAN_2003_0693: OpenSSH large packet buffer overflow high
  precond:
            src_host.priv = any  // remote attack
            dst_host.compromised_state < root
            dst_host is running openSSH 3.7.1 and before
            connectivity [src_host, dst_host] = true

  postcond:
            dst_host.compromised_state = root
            prob_of_exploit = high
            exploit = true
```

```
CAN_2004_0492: Apache heap buffer overflow
precond:
            src_host.priv = any  // remote attack
            dst_host.compromised_state < root
            dst_host is running Apache 1.3.25 to 1.3.31
            connectivity [src_host, dst_host] = true

postcond:
            dst_host.compromised_state = ~avail
            prob_of_exploit = high
            exploit = true
```

```
CAN_2002_0839: Apache httpd vulnerability
precond:
            src_host.priv = usr of Apache //local attack
            src_host.id = dst_host.id
            dst_host is running Apache 1.3 - 2.0
            dst_host.compromised_state < root

postcond:
            dst_host.compromised_state = root & ~avail
            prob_of_exploit = high
            exploit = ture
```

```
CAN-2004-0394: Linux kernel panic function buffer overflow
 precond:
            src_host.priv = usr of Apache //local attack
            src_host.id = dst_host.id
            dst_host is running Linux 2.4.x

 prostcond:
            dst_host.compromised_state = confid
            prob_of_exploit = low
            exploit = true
```

Figure 2. Attack Template Examples

3.1. Bayesian Attack Graph

A Bayesian network is defined by a directed acyclic graph (DAG) over nodes representing random variables and arcs signifying conditional dependencies between pairs of nodes. Let a set $\mathbf{X} = \{X_1, ..., X_n\}$ of discrete variables where each variable X_i may take on values from a finite domain. A Bayesian network is a pair $(S, \mathbf{P})$ where S is a network structure that encodes a set of conditional independence assertions about variables in $\mathbf{X}$, and $\mathbf{P}$ is a set of local probability distributions associated with each variable. That is, $\mathbf{P} = \{P_i\}$ where $P_i = P(X_i|\mathbf{Pa}_i)$, $\mathbf{Pa}_i$ denotes the parents of node X_i in S. The directed Markov property with respect to S gives the joint probability distribution of $\mathbf{X}$ as:

$$p(\mathbf{x}) = \prod_i p(x_i|\mathbf{pa}_i) \tag{1}$$

We use Bayesian network to model the global inter-correlations of all potential atomic attacks in a given network. We term such Bayesian network as *Bayesian attack graph*. In such a graph, Each node is a Bernoulli random variable X_i which represents a host with a specific system state. A true state, $x_i = 1$, indicates a host with a compromised system state that accomplished by an attacker. Each edge is a directed arc that represents conditional dependency between a pair of nodes. An edge exists if there is a possible exploit that can be instantiated from a source node X_i to a target node X_j. An associated edge weight is specified by the probability of success from X_i to X_j, denoted as $p(x_j|x_i)$, i.e. $P(X = x_j|X = x_i)$. Each path from a root node (node with no parent) to a non-root node represents a sequence of atomic attacks.

Specifying probability of an exploit requires domain expert knowledge. Most existing scanning tools report vulnerabilities with a set of categorical security measurements, such as severity level, range of exploit, and vulnerability consequences. Therefore, considering the nature of a network, one can define a multi-dimension security matrix using these categorical information and quantify ordinal levels of each category into numerical values. Each matrix entry value can then be computed by a function of contributions from various dimensions, such as a linear addictive function or multiplicative function. Then, it can be converted to a value within the range [0,1] by applying a scale function to the matrix. Such value represents the probability of an exploit. For example, One can define a two dimension $m \times n$ security matrix $W = (w_{ij})$, with one dimension w_i to denote severity levels and another dimension w_j to denote ranges of exploits. A 3-scale severity level may be specified as $\{high = 0.9, medium = 0.5, low = 0.3\}$, and 2-scale exploit ranges may be specified as $\{remote = 0.5, local = 0.9\}$. If applying a multiplicative function to the matrix, then each entry value is given by $w_{ij} = w_i \times w_j$.

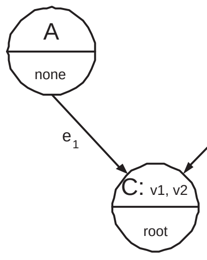

Figure 3. A Simple Example of Bayesian Attack Graph

Figure 3 shows a simple Bayesian attack graph with three nodes A, B, C, each represents a specific host with a potential security violation state. It is important to note that two nodes may represent the same host but with different states, for instance, one with user privilege, and one with root privilege. Two vulnerabilities v_1 and v_2 exhibit on node C. v_1 is exploitable from host A (i.e. exploit e_1 is instantiated) with the probability of success $p(e_1)$. v_2 is exploitable from host B with the probability of success $p(e_2)$. Both exploitations result in the same compromised state on node C. The conditional probability distribution for node C is given in Table 1.

Table 1. Conditional Probability specifications for node C in Figure 3

A	**B**	**P(C=F)**	**P(C=T)**
0	0	1	0
1	0	$1 - p(e_1)$	$p(e_1)$
0	1	$1 - p(e_2)$	$p(e_2)$
1	1	$1 - p(e_1 \cup e_2)$	$p(e_1 \cup e_2)$

3.2. Constructing Bayesian Attack Graph

We construct Bayesian attack graphs based on two assumptions:

- given a node X_i, each parent node of X_i can independently influence the state of X_i
- an attacker will never backtrack once reach a compromised state

The second assumption is similar to the monotonicity assumption as described in.[13] It is reasonable because backtrack does not contribute extra information about attack paths. In addition, it helps to eliminate potential cycles in the graph. Consequently, we observe that equation (1) holds in our attack graph.

Bayesian attack graphs are built through our graph generation routine by matching a list of hosts with profile information against a set of predefined attack templates. For each attack template, if all preconditions are met, values of postcondition attributes are updated and $exploit = true$ is returned, an edge is then added to the graph. Specifically, we take the following steps:

1. Specify one state or a set of states with respect to the security properties on the network that we are interested in. In other words, specify the goal state(s) of attackers, e.g. the root privilege on a private file server. Outside attackers are specified by a subset hosts on the public network. Identifying such subset are subjective. We can extend our model to build attacker profiles as in.[8]
2. Determine a set of variables (i.e. nodes) to model. Given h hosts on the network, and l violation states, then the number of variables N we have to model is $h \times l$. Security analysts can set $l = 6$, which corresponds to the six common types of security breaches as mentioned in Section 3. The number of nodes can be reduced

in the subsequent step if no path can lead to any security violations on that node. The number of nodes can also be reduced if only a subset of hosts are monitored. For instance, some enterprise may consider gaining access/privilege of a group of server machines is more serious than losing integrity of data or even losing availability of these machines. In such case, security analysts can narrow down the security violation types to $\{root, user, other\}$.

3. Build host profiles and attack templates. A host profile includes its network and system configurations, and exhibited vulnerabilities that identified through a list of off-the-shelf network scanning tools. Network connectivity of hosts is stored in a connectivity matrix. As we stated in the beginning of this section, an attack template include precondition and postcondition of an atomic attack, along with the probability of success. We can gather these information from the description of a vulnerability, i.e. the categorical information reported from the scanning tools.

4. Construct Bayesian attack graph structure S. This step is accomplished by exploring the causal relationship between pairs of nodes. Causal relationships are defined in the attack templates. We use an adjacency-matrix to represent S. Consider a DAG $S = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ denotes a set of nodes and $\mathbf{E}$ denotes a set of edges. The adjacency-matrix representation of S is then given by a $|\mathbf{V}| \times |\mathbf{V}|$ matrix $M = (m_{ij})$, where $|\mathbf{V}| = N$, and $m_{ij} = 1$ if $(i, j) \in \mathbf{E}$, $m_{ij} = 0$ otherwise. The weight $w(i, j)$ of the edge (i, j), or the probability of a successful exploit, is stored as the same entry in the i-th row and j-th column of the adjacency matrix. A value 0 is assigned if an edge does not exist.

   During graph construction, there may be more than one arc between a pair of nodes. That is, the source node can instantiate multiple exploits from the target node. This conflicts DAG structure. We reduce such multiple arcs to one, in view of the fact that all of them corresponds to the same path from the source node to the target node. We then compute the probability of the union set of these exploits as the probability of success for the aggregated arc. This is reasonable, since the chance for a source node A to reach a target node B increases proportional to the number of vulnerabilities B exhibited. Therefore, from the view of attack paths, we lose nothing in this edge reduction process. Independent nodes (i.e. nodes with no ancestor, nor descendent) either represent that the states of these nodes are not relevant to the target nodes, or any attacker has no way to reach them yet if they are part of the target nodes. These nodes are eliminated by simply deleting both the rows and columns that filled with zeros in the adjacency-matrix M, when the index of row and column are equal.

5. Compute the local conditional probability distributions (CPDs) for all non-root nodes in S, i.e. the probability distributions $p(x_i|\mathbf{pa}_i)$. The nodes in our model are similar to the noisy-OR nodes.[15] In general, we can compute CPDs at node i:

   $$p(x_i = 1|\mathbf{pa}_i) = 1 - \prod_j (1 - p(x_i = 1|x_j)), \tag{2}$$

   where $j \in \mathbf{pa}_i$, and $p(x_i = 1|x_j)$ is computed in the previous step. For all root nodes, Bernoulli priors (probabilities of these nodes initiate an attack) must be specified. Note that all these probability distributions, either for root nodes (Bernoulli priors) or non-root nodes (CPDs), are priors that subject to uncertainty. Incorporate attacker profiles to our model can alleviate this problem, but can not eliminate it. Fortunately, Bayesian network model is robust to the imprecisions of priors, and these hypothesis beliefs can be updated in the light of new knowledge about the underlying network.

In step 3 above, we use backward exploration to discover the conditional relationships between pairs of nodes. That is, the search of matching a list of host profiles against attack templates starts from the goal state(s) of attackers. As indicated in[9] and,[10] backward exploration results in significant savings in space in comparison with forward exploration, since only nodes and arcs that are relevant to the target node(s) are generated. Another advantage of backward exploration in our model is that we include all insiders (nodes within the network) as part of the attacker group. This is illustrated in our example attack graph as shown in Figure 5 in Section 5. Therefore, in our model we cope with the fact that approximately 33% of attacks are due to insiders for an enterprise network,[16] and identify the risks from inside nodes.

4. ANALYSIS OF ATTACK GRAPHS

4.1. Bayesian Inference

Once Bayesian attack graph is built, we can then use it to perform probabilistic inference. In particular, we are interested in 1) computing the posterior probability for belief updating; 2) given some observed variables, finding the Most Probable explanation (MPE) assignment, i.e. finding a set of attack paths with maximum probability assignment to all the hidden nodes (unobserved nodes).

As described in Subsection 3.1, suppose we have a set $\mathbf{X}$ of discrete variables, our Bayesian attack graph is defined by a pair $(S, \mathbf{P})$, where S is a DAG structure that encodes conditional independence assertions over $\mathbf{X}$, and $\mathbf{P}$ is a set of local probability distributions associated with S. The joint probability distribution encoded by the pair $(S, \mathbf{P})$ is defined in equation (1). Let a set $\mathbf{D} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ of cases, where each case is an instance of some or all the variables in $\mathbf{X}$. Let D denote a sample from $\mathbf{D}$, that is, D denotes a set of our observations (evidences) on $\mathbf{X}$. Let S^h be the hypothesis that the underlying distribution satisfies the independencies. The posterior distribution we are interested in is

$$P(S^h|D) \propto P(D|S^h) \times P(S^h) \tag{3}$$

where $P(D|S^h)$ is called marginal likelihood of D given S^h, and $P(S^h)$ is the prior probability distributions of the structure S.

We consider posterior probabilities as overall security metrics to represent quantitative vulnerability assessment on a given network. In other words, a posterior probability denotes how likely attacker(s) may mount attacks given some vulnerabilities exhibited on hosts of the network. For instance, S^h in equation (3) may represent one or a set of attacker nodes, then posteriori probabilities computed from equation (3) represent our beliefs on how likely attacker(s) will mount attacks to the given network. If no observation is obtained on the nodes in S, the posteriori probabilities will depend on our prior knowledge over the nodes in S, which are typical subjective measurements. However, once observations are collected from data over time, our hypothetic beliefs will be updated. We can then easily answer the following questions with certain confidence about the hypothetic beliefs.

- How vulnerable of my network now given the evidence that observed so far?
- Is my network better off now than it was in last month?

The answers to these questions present valuable information to corporate management on how to plan budget for their network security, and on how to place their defence against the vulnerabilities exhibited on the network.

Exact inference is known as "NP-hard". A variety of inference algorithms have been proposed to address the tradeoffs between model accuracy, generality, and complexity. We use the Bucket Elimination (BE) algorithm proposed by R. Dechter[17] for our belief updating and MPE inference.

As proved in,[17] the complexity of belief updating on BE is (time and space) exponential in the induced width $w^*(d, e)$ of the network's ordered moral graph, where w^* is the minimal induced width over all its ordering, d is a specific ordering of the nodes, and e is an evidence. The width w of each variable in the ordered graph is defined as the number of its earlier neighbors in the orders. The complexity of MPE in time and space is exponential in the cardinality of the maximum family size.

In our problem domain, in order to reduce exponential factors of complexity, security analysts can first divide their entire network into different groups and then evaluate each group individually. The grouping choice can be based upon the similarity in responsibilities of different hosts, or upon its networks security policy. Security analysts can also randomly divide their network into equal-size groups for cross examination.

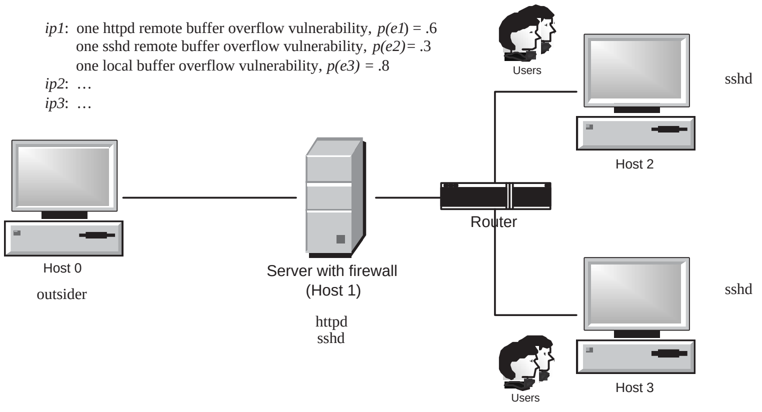
ip1: one httpd remote buffer overflow vulnerability, p(e1) = .6
one sshd remote buffer overflow vulnerability, p(e2)= .3
one local buffer overflow vulnerability, p(e3) = .8
ip2: …
ip3: …
Users
sshd
Host 2
Router
Host 0
outsider
Server with firewall
(Host 1)
httpd
sshd
sshd
Users
Host 3

Figure 4. An Example Network

4.2. Causality Reasoning

Another common use of Bayesian network is for causality reasoning. Our model can be used for "bottom up" reasoning, i.e., from effects to causes once we confirm a compromise state. For example, in Figure 3, if node C is verified of being compromised, then as shown in the previous subsection, we can compute the posterior probability of node A and B, which are $P(A = 1|C = 1)$ and $P(B = 1|C = 1)$ respectively. The likelihood ratio gives the explanation of causality. If A and B are descendant nodes, the reasoning process can go further toward the parent nodes. This process helps network administrators to improve efficiency and effectiveness of troubleshooting along the attack paths, hence can quickly identify the victim machines.

5. EXPERIMENTAL RESULTS

We ran our model on a Linux-based experimental network as shown in Figure 4. Host 0 (ip_0) is an outsider, and only has connection to Host 1 (ip_1). Host 1 runs two daemon programs, httpd and sshd. Host 2 (ip_2) and host 3 (ip_3), each runs one sshd program. Host 1, 2, and 3 are interconnected. We select a subset of vulnerabilities that are identified on the hosts, and we only model two levels of privilege violations $\{user, root\}$ to keep the graph and CPT tables simple for illustration. Let e_i denotes an exploit on one or a set of vulnerabilities exhibited on the hosts of the given network, and $p(e_i)$ corresponds to the probability of success on e_i. We apply a two-dimensional security matrix, as described in the aforementioned Subsection 3.1, to compute $p(e_i)$. The potential exploits and the probabilities of success for three inside nodes are listed as follows.

ip_1:
one httpd remote buffer overflow vulnerability, $p(e_1) = 0.6$
one sshd remote buffer overflow vulnerability, $p(e_2) = 0.3$
one local buffer overflow vulnerability , $p(e_3) = 0.8$

ip_2:
one sshd remote buffer overflow vulnerability, $p(e_2) = 0.3$
one local buffer overflow, $p(e_3) = 0.8$

ip_3:
one sshd remote buffer overflow vulnerability, $p(e_2) = 0.3$
two local buffer overflow, $p(e_3) = 0.8$ and $p(e_4) = 0.5$

Since we only model privilege violations, as explained in Subsection 3.2, number of violation levels $l = 2$, and number of hosts $h = 4$, hence the total (and maximum) number of nodes on the attack graph is $N = 8$. They are: $ip_0.user$, $ip_0.root$, $ip_1.user$, $ip_1.root$, $ip_2.user$, $ip_2.root$, $ip_3.user$, and $ip_3.root$.

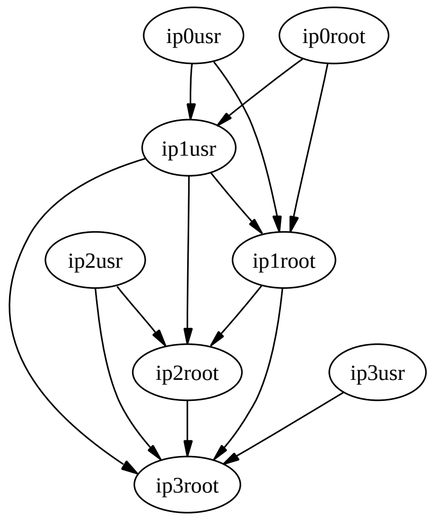

Figure 5. Attack Graph

We run our attack graph generator and generate a Bayesian attack graph with structure S represented in adjacency-matrix M. Each matrix entry contains the connectivity of an edge and an associated weight $p(i, j)$. Figure 5 is the attack graph for this example network, visualized using GraphViz.[18] The conditional probability table (CPT) for each node is computed using equation (2). The CPT for node $ip_2.root$ is illustrated in Table 2.

We hypothetically assign the prior probabilities to root nodes as follows: for outsiders, $P(ip_0.user = 1) = 0.5$, $P(ip_0.root = 1) = 0.5$, for insiders, $P(ip_2.user = 1) = 0.1$, $P(ip_3.user = 1) = 0.1$. The specification of prior probabilities makes more sense if we construct attacker profiles as described in Ref. 9.

Given the experimental network in Figure 4, there are 20 attack paths from the set of root nodes to the leaf node $h_3.root$. Obviously it is quite space consuming if using tree structure to enumerate all possible paths. As we can see from Figure 5, Bayesian attack graph are more compact and scalable.

Table 2. Conditional Probability specifications for node $ip_2.root$ in Figure 5

$ip_1.user$	$ip_1.root$	$ip_2.user$	$P(ip_2.root = F)$	$P(ip_2.root = T)$
0	0	0	1	0
1	0	0	0.7	0.3
0	1	0	0.7	0.3
1	1	0	0.49	0.51
0	0	1	0.2	0.8
1	0	1	0.14	0.86
0	1	1	0.14	0.86
1	1	1	0.098	0.902

5.1. Belief Updating Inference

Using BN inference algorithm, we obtain the following posteriors:

- $P(ip_3.root = 1) = 0.249$; i.e., the probability for ip_3 root privilege being compromised is 0.249, under no knowledge (i.e. observation) of the states in any nodes in the graph.

- $P(ip_3.root = 1|ip_0.user = 1, ip_0.root = 1) = 0.3973$; i.e., the probability for ip_3 root privilege being compromised is 0.3973, if we assume the outsider ip_0 mounts attacks.
- $P(ip_3.root = 1|ip_1.root = 1) = 0.4734$; i.e., the probability for ip_3 root privilege being compromised is 0.4734, if $ip_1.root$ is verified as being compromised.
- $P(ip_3.root = 1|ip_0.user = 1, ip_0.root = 1, ip_2.root = 0) = 0.3045$; i.e., the probability for h_3 root privilege being compromised is 0.3045, if assume ip_0 mounts attacks, and meanwhile we are certain that $ip_2.root$ is not being compromised.
- $P(ip_1.user, ip_1.root, ip_2.usr|ip_3.root = 1)$ is given in Table 3. From the table, $P(ip_1.usr = 0\&ip_1.root = 1\&ip_2.usr = 0|ip_3.root = 1) = 0.1513$, which is higher than $P(ip_1.usr = 0\&ip_1.root = 0\&ip_2.usr = 1|ip_3.root = 1) = 0.1231$.

Table 3. Posteriori probabilities of nodes $\{ip_1.user,\ ip_1.root,\ ip_2.usr\}$ in Figure 5, given $ip_3.root = 1$

$ip_1.user$	$ip_1.root$	$ip_2.user$	*posteriori*
0	0	0	0.2564
1	0	0	0.0278
0	1	0	0.1513
1	1	0	0.1007
0	0	1	0.1231
1	0	1	0.0474
0	1	1	0.0252
1	1	1	0.2682

Hence, using posterior probabilities as security metrics provides quantitative measurements on evaluation of network exposures.

Prior probabilities can also be updated if we have new evidences. For instance, we originally assign $p(ip_0.user) = 0.5$. If we observe that $ip_2.root$ is true, then we compute $P(ip_0.user = 1|ip_2.root = 1) = 0.6971$. Thus, once we know $ip_2.root$ has been compromised, the probability of $ip_0.user$ mounting attacks is 0.6971, up from 0.5 before the evidence.

5.2. MPE Inference

Given a Bayesian attack graph, and a set of evidences, we can find a set of attack paths with maximum probability assignment to all the hidden nodes.

For example, in Figure 5, if we observe ($ip_0.usr = 1$ & $ip_0.root = 0$ & $ip_1.root = 1$), then using MPE inference routine, we can obtain the maximum probability assignment to the rest of the nodes as $\{ip_1.usr = 1$, $ip_2.usr = 1$, $ip_2.root = 1$, $ip_3.usr = 0$, $ip_3.root = 1\}$. That is, given the evidences in the observed nodes, except node $ip_3.usr$, all other nodes are prone to attacks. Hence the attack paths covered by these nodes are considered as an optimal subset of attack paths given the knowledge about the observed nodes.

6. CONCLUSIONS

This paper presented a method to model all potential attack paths on a given network with Bayesian graphical model. Bayesian attack graphs are more compact and scalable than conventional tree structure attack graphs. Our Bayesian attack graph model is implemented in C++, and it can automatically generate an attack graph with edges and weights represented in an adjacency-matrix. Quantitative vulnerability assessment of a network of hosts is achieved by associating measurement metrics with posterior conditional probabilities of Bayesian network. The Bucket Elimination algorithm is used for probabilistic inference. Approximate inference engine may be used when the complexity of attack graph grows. We believe our method is applicable to enterprise level networks by carefully choosing the modelling hosts and states. It can provide both quantitative and relative assessment of network vulnerabilities at any given time or over a period of time.

REFERENCES

1. *NESSUS: a vulnerability assessment tool*, http://www.nessus.org/.
2. *ISS Internet Scanner: an application-level vulnerability assessment tool*, http://www.iss.net.
3. *Tripwire: a file integrity checker*, http://www.tripwire.org/.
4. B. J. Frey, *Graphical Models for Machine Learning and Digital Communication*, The MIT Press, 1998.
5. M. Jordan, *Graphical Models*, The MIT Press, 1999.
6. M. Dacier, Y. Deswartes, and M. Kaaniche, "Quantitive assessment of operational security models," tech. rep., LAAS Research Report 96493, May 1996.
7. R. Ortalo, Y. Deswarte, and M. Kaâniche, "Experimenting with quantitative evaluation tools for monitoring operational security," in *IEEE Transactions on Software Engineering*, **vol. 25, no. 5**, pp. 633–650, Sept.-Oct. 1999.
8. C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, January 1999.
9. L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," in *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX'01)*, **vol. 2**.
10. R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy (Oakland 2000)*, pp. 156–165, May 2000.
11. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy (Oakland 2002)*, pp. 254–265, May 2002.
12. S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pp. 49–63, June 2002.
13. P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 217–224, November 2002.
14. *ICAT Metabase: A CVE Based Vulnerability Database*, http://icat.nist.gov/icat.cfm/.
15. J. Pearl, *Probalistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.
16. S. Garfinkel, G. Spafford, and A. Schwartz, *Practical Unix & Internet Security*, ch. preface. O'Reilly, 2003.
17. R. Dechter, "Bucket elimination: A unifying framework for probabilistic inference," in *Graphical Models*, M. Jordan, ed., pp. 75–104, The MIT Press, 1999.
18. *GraphViz: a toolkit of graph filtering and rendering tools*, http://www.research.att.com/sw/tools/graphviz/.