A Novel Attack Graph Posterior Inference Model Based on Bayesian Network

Shaojun Zhang¹, Shanshan Song²

¹School of Information Security Engineering, Shanghai Jiao Tong University, Shanghai, China
²Information Technology Department, Guotai Junan Futures Co., Ltd, Shanghai, China

E-mail: {leony7888, songss1985}@hotmail.com
Received January 5, 2011; revised January 17, 2011; accepted January 18, 2011

Abstract

Network attack graphs are originally used to evaluate what the worst security state is when a concerned network is under attack. Combined with intrusion evidence such like IDS alerts, attack graphs can be further used to perform security state posterior inference (i.e. inference based on observation experience). In this area, Bayesian network is an ideal mathematic tool, however it can not be directly applied for the following three reasons: 1) in a network attack graph, there may exist directed cycles which are never permitted in a Bayesian network, 2) there may exist temporal partial ordering relations among intrusion evidence that cannot be easily modeled in a Bayesian network, and 3) just one Bayesian network cannot be used to infer both the current and the future security state of a network. In this work, we improve an approximate Bayesian posterior inference algorithm—the likelihood-weighting algorithm to resolve the above obstacles. We give out all the pseudocodes of the algorithm and use several examples to demonstrate its benefit. Based on this, we further propose a network security assessment and enhancement method along with a small network scenario to exemplify its usage.

Keywords: Network Security, Attack Graph, Posterior Inference, Bayesian Network, Likelihood-Weighting

1. Introduction

Network attack graphs [1-5] are widely used as a good tool to analyze network security state in comprehensive consideration of exploits, vulnerabilities, privileges, network connectivity, etc. Originally, they are built to tell what the worst scenarios are when a network is under attack. But later, it is found that security alerts can be mapped to them [6-8] and along with these observed intrusion evidence, attack graphs can also be used to assess the security state of a concerned network dynamically.

In this area, probabilistic approaches have been proposed to perform such analysis. In [9], a method which reasons about complementary intrusion evidence is presented. According to the method, security alerts generated by intrusion detection systems (IDSs) as well as reports generated by system monitoring tools can be integrated into Bayesian networks. And prior conditional probability values which denote the success rate of the corresponding attacks can be assigned to each of the evidence nodes. By doing this, uncertain or unknown intrusion evidence can be reasoned about based on verified evidence. Although quite useful in reasoning observed intrusion evidence, this method cannot tell people what attack will be executed next and with what probability.

In [10], HCPN (Hidden Colored Petri-Net) is used to depict the relationship among different steps carried out by an intruder and model intrusion actions and intrusion evidence together. The initial state of HCPN attack graph is determined by an initial probability distribution. And empirical formulas are defined to reevaluate its state after receiving each alert from the sensors (most commonly are IDSs). This method runs quite well in predicting what the next intrusion actions are. However, at reevaluating the probabilities of the graph nodes according to the alerts, the method only updates probabilities of the successor nodes of an assumed action node, which obviously contravenes our intuition that in most inference algorithms there must be backward belief propagation (i.e. probabilities of the predecessor nodes should also be updated).

To overcome these flaws, we firstly thought about extending the Bayesian network into a general attack graph
There are at least three main differences between an attack graph and a Bayesian network which obstruct this way:

- In a Bayesian network, no directed cycles are allowed. However, in a network attack graph, this restriction is not appropriate since people always want to depict potential intrusion paths succinctly.
- In a Bayesian network, there can not be any partial ordering relations among evidence nodes. However, we can often observe temporal partial ordering relations among intrusion evidence (e.g. when an Ips-sweep is observed before a portsan), which may indicate that some exploits happen before some others.
- At performing attack graph-based posterior inference, two questions are most often raised: 1) what is the current state of a network, and 2) what is the future state of it. Essentially this means one set of observed intrusion evidence should be used to infer two temporally different states. In Bayesian inference, this demands two prior conditional probabilistic distribution, one for current state inference and one for future state inference. Although we think it feasible to define the later one (For example we say an exploit will happen in probability 0.8 if an attacker was given enough time), it is really a disaster to define the former one (how to assess the exploit probability when the attacker has got two hours).

These obstacles almost made us give up Bayesian posterior inference. But fortunately we find a good way to overcome them—we manage to improve the likelihood-weighting algorithm (an approximate Bayesian inference algorithm) into a novel attack graph-based posterior inference algorithm. And based on this, we find a method to quantitively assess the overall security level of a concerned network and identify the most cost-effective security measures to enhance it.

The rest of this paper is organized as follows. Section 2 depicts the aforementioned posterior inference problems in details. Section 3 introduces the underlying formalized attack graph definition. Section 4 describes our improved likelihood-weighting algorithm and Section 5 gives out several examples for benefit testification. Section 6 presents our security assessment and enhancement method and Section 7 gives out an example to exemplify it. The last section concludes the paper.

2. The Primal Motives

2.1. Directed Cycles

Various models and methods have been proposed to represent multi-step network attacks and generate network attack graphs automatically. These models and methods can be roughly divided into two categories: security state enumeration and vulnerability/exploit dependency. Comparatively, the later one is more popular since it exhaustively and succinctly depicts the dependency of security elements such as privilege, trust, vulnerability, exploit, network connectivity, etc. Representatives of this category include the approaches proposed in [2-5]. In this category, although some approaches promise that they only generate attack graphs without directed cycles, we cannot assume that all of them are generating DAG (Directed Acyclic Graph).

Here is an example to demonstrate that directed cycles sometime are useful since we want to depict the intrusion paths succinctly. Assume there are three hosts on a small network which is illustrated in Figure 1. Host Master has a certain vulnerability that can be exploited by the other two hosts to gain its USER privilege. On the other hand, the USER accounts on Master are mapped to a ROOT account on the other two hosts.

We can imagine that a succinct network attack graph for this network is like the one shown in Figure 2.

In Figure 2 we adopt a graph notation widely used in Petri-Net. Circle s1, s2 and s3 respectively denote that the attacker has got ROOT privilege on Slave1, ROOT privilege on Slave2 and USER privilege on Master. Line a1 and a2 denote the attacker exploits the vulnerability of Master respectively from Slave1 and Slave2. Obviously, in this attack graph, there exist directed cycles.

2.2. Evidence Partial Ordering Relations

In a Bayesian network, there cannot be any partial ordering relations among observed evidence. However, at performing security posterior inference, temporal partial ordering relations among evidence nodes often provide us important cues. Figure 3 illustrates an example which demonstrates the benefit of analyzing temporal partial ordering relations among intrusion evidence.

In Figure 3, we assume that the attacker initially occupies resource s1 and her goal is to occupy s6. The attacker can use exploit a1~a6 to perform intrusion. However, exploit a1 and a2 will trigger alert o1, exploit a3 and a5 will trigger alert o2 and a3 and a6 will trigger alert o3. Finally, during the intrusion, an evidence sequence o2→o1→o3 is observed.

Analysis: To achieve her goal, the attacker can choose
In this section, we propose a formalized network attack graph definition as the basis for attack graph-based security posterior inference.

In the early days of Internet, network attacks are often performed to demonstrate the personal skills of the attacker. They were limited to a small number of known methods such as cracking the password and exploiting the operating system vulnerabilities. But lately attacks have evolved into complex procedures which may comprise several interrelated intrusion actions. Execution of these actions incrementally changes the security state of the network, making the attacker take over more and more resources (and most commonly during the intrusion procedure the attacker will not give up resources she has already got [3]) and eventually achieve her goal. Fortunately, security devices such as IDSs will send alerts if there is an attack. Then administrators can use them to assess the real state of the network and take proper measures to compensate.

A network attack graph depicts the above three components (network resource, intrusion action and intrusion evidence) and their causal relationship. In most cases, it can exhaustively and succinctly comprises all of the potential intrusion paths. Based on the above analysis, an attack graph can formally be defined as:

**Definition 1.** An attack graph is a 10-tuple directed graph $AG = (S, S_0, G, A, O, E, Δ, Φ, Θ, Π)$, where:

- $S = \{s_i | i = 1, \cdots, N_s\}$ is a finite set of resource state nodes. The value of each node variable $s_i$ can be either `True` or `False`, denoting if a resource has been taken over by the attacker;
- $S_0 \subseteq S$ is a subset of $S$ representing resources the attacker may initially occupy. Graphically it is the root nodes of $AG$;
- $G$ is the set of possible actions the attacker may perform;
- $A$ is the set of possible actions the attacker may perform;
- $O$ is the set of possible actions the attacker may perform;
- $E$ is the set of possible actions the attacker may perform;
- $Δ$ is the set of possible actions the attacker may perform;
- $Φ$ is the set of possible actions the attacker may perform;
- $Θ$ is the set of possible actions the attacker may perform;
- $Π$ is the set of possible actions the attacker may perform.

Two intrusion paths:

- $a. s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_3} s_4 \xrightarrow{a_5} s_5 \xrightarrow{a_6} s_6$
- $β. s_1 \xrightarrow{a_2} s_3 \xrightarrow{a_4} s_5 \xrightarrow{a_6} s_6$

If we neglect all temporal partial ordering relations, then the three evidence nodes are set to `True`. And since the attack graph is symmetrical (notice there is no ordering relations between evidence nodes), using Bayesian posterior inference we can find that both intrusion paths can be chosen by the attacker. However, if we do consider temporal partial ordering relations, we can infer that only intrusion path $β$ is chosen by the attacker, since executing intrusion path $a$ will violate the temporal partial ordering relation $a_5 \rightarrow a_6$.

### 2.3. Posterior Inference for Multi-State

As we mentioned before, two questions are most often raised at performing attack graph-based posterior inference: 1) what is the current state of a network, and 2) what is the future state of it. In Bayesian inference, this means two different prior conditional probabilistic distribution should be assigned—one for current state inference and one for future state inference. If we say the assignment for the later one is tough but still practical, then it is almost infeasible to define the former one.

People may argue that Hidden Markov Model [12] or Generalized Hidden Semi-Markov Models [13] can be used to resolve this problem. But in HMM or GHSMMs, a key concept is the time instants associated with state changes. This concept is quite natural in technique areas such as speech signal processing. However in security analysis we cannot just fix a time slot for an attacker to perform actions. And even if we do constrainedly figure out this slot, we still face the problem of how to define the probability of an action when the attacker is given one time slot.

Under this understanding, we decide to stick to Bayesian inference and seek if we can use one prior conditional probabilistic distribution with one set of observed intrusion evidence to infer two temporally different security states. Eventually we successfully resolve this challenge by inventing a sample reprocessing method called transientization which will be introduced in Section 4.

### 3. The Underlying Model

In this section, we propose a formalized network attack graph definition as the basis for attack graph-based security posterior inference.

A small network environment.

![Figure 1. A small network environment.](image1)

An attack graph for the network.

![Figure 2. Attack graph for the network.](image2)

A simple network attack graph.

![Figure 3. A simple network attack graph.](image3)
$G \subseteq S$ is a subset of $S$ representing attack goals;

$A = \{a_i | i = 1, \ldots, N_a\}$ is a finite set of intrusion action nodes. The value of each node variable $a_i$ can be either True or False, denoting whether an intrusion action has been conducted by the attacker;

$O = \{o_i | i = 1, \ldots, N_o\}$ is a finite set of intrusion evidence nodes. The value of each node variable $o_i$ can be either True or False, denoting whether the evidence has been observed. Considering that in most occasions intrusion evidence will be preprocessed (e.g. fused) by the low-layer sensors so that their observation numbers are often distorted, here we only consider whether a kind of evidence has been observed, discarding its concrete observation number;

$E = (E_i \cup E_o \cup E^*)$ is a finite set of edges which link nodes together. $E_i \subseteq S \times A$ is a set of edges which denote actions can only be conducted given that all the prerequisite resources are taken over by the attacker; $E_o \subseteq A \times S$ is a set of edges which denote actions may consequently let the attacker take over some other resources and $E^* \subseteq A \times O$ is a set of edges which denote actions may trigger certain intrusion evidence. Generally we use Pre$(n)$ and Con$(n)$ to respectively denote the prerequisite nodes and consequent nodes of node $n$;

$\Delta = \{\delta : \text{Pre}(a_i, a_j) \rightarrow [0, 1]\}$ is the prior conditional probability distribution that an action will be conducted if its prerequisite is satisfied. In this paper, we assume that all elements of Pre$(a_i)$ are in a conjunctive normal form. In other words, an action can be conducted only if all its prerequisite resources are occupied by the attacker;

$\Phi = \{\phi : a_i, \text{Con}(a_i) \rightarrow [0, 1]\}$ is the probability distribution that an action will succeed if it is conducted. Since an action changes its consequent resource state only when it succeeds, $\Phi$ is also the probability that an action set its consequent node variables to True. Here we assume that for any resource node $s_i$, if there are more than one successful actions in Pre($s_i$), then each of them can set $s_i$ to True independently;

$\Theta = \{\theta : (a_i, o_i) \rightarrow [0, 1]\}$ is the probability distribution that a type of intrusion evidence will be observed if one of its prerequisite actions is conducted. Here we also assume that for any evidence node $o_j$ if there are more than one successful actions in Pre($o_j$), each of them can set $o_j$ to True independently;

$\Pi = \{\pi : S \cup A \cup O \rightarrow [0, 1]\}$ is the node belief distribution of $AG$. Here $\pi(s)$ denotes the probability that the attacker has taken over resource $s$;

$\pi(o)$ denotes the probability that action $a$ has been conducted by the attacker and $\pi(o)$ denotes the probability that evidence $o$ has been observed. Specially, $\Pi_0$ is the initial node belief distribution of $AG$, denoting what resources are occupied by the attacker at the very beginning. So, we can expect:

$\pi_0(n) \geq 0, n \in S_0$

$\pi_0(n) = 0, n \in S_0$

Graphically, a network attack graph follows Definition 1 is like the one illustrated in Figure 4.

In Figure 4, the attacker initially occupies resource $s_1$, $s_2$ and $s_3$ (with probabilities defined in $\Pi_0$). Then intrusion actions $a_1$, $a_2$ and $a_3$ will be conducted (with probabilities defined in $\Delta$), and further make the attacker take over resource $s_4$-$s_7$ (with probabilities defined in $\Phi$). As actions being conducted, intrusion evidence $o_1$-$o_6$ will be triggered and observed (with probabilities defined in $\Theta$).

As mentioned before, in this paper, we are only interested in whether a type of evidence has been observed, discarding its concrete observation number. However, we can still utilize the temporal partial ordering relations among attack evidence to assist posterior inference.

Definition 2. There are two categories of evidence temporal partial ordering relations. We say:

- There is a type I temporal partial ordering relation $o_m \rightarrow o_n$ if $o_m$ is observed before $o_n$ is observed. In other words, the first observation timepoint of $o_m$ is earlier than the first observation timepoint of $o_n$.

- There is a type II temporal partial ordering relation $o_m \nrightarrow o_n$ if $o_m$ is never observed after $o_n$ is firstly observed. In other words, all the observation time-points of $o_m$ are earlier than any of the ones of $o_n$.

With the above definition, the problem of network attack graph-based posterior inference can be defined as:

Given an attack graph $AG$, when an evidence sequence $a = o_1 \rightarrow o_2 \rightarrow \cdots \rightarrow o_6$ which conforms to a partial
ordering relation set \( \Omega = \{a_n \sim a_k \} \cup \{a_n \rightarrow a_k \} \) is observed, how to compute the corresponding graph node belief distribution sequence \( \beta = \Pi_1 \Pi_2 \cdots \Pi_k \) ?

4. The Posterior Inference Algorithm

In this section, we propose an algorithm for resolving the above posterior inference problem. Our algorithm is mainly based on the approximate Bayesian inference algorithm—the likelihood-weighting algorithm [14,15] to prove one of the approximate Bayesian network inference algorithms—the likelihood weighting algorithm.

A Bayesian network (or a belief network, a causal network) is a probabilistic graphical model that represents a set of variables and their probabilistic independencies. Essentially, a network attack graph following Definition 1 is a mimetic Bayesian network. However, since a Bayesian network cannot contain any directed cycle or partial ordering relation of evidence nodes, traditional Bayesian inference should not be used to perform this attack graph-based posterior inference.

To support these additional features, we manage to improve one of the approximate Bayesian network inference algorithms—the likelihood-weighting algorithm [14,15] to a novel one. Likelihood weighting enhances logic sampling algorithm in that it never discards samples. It is the most commonly used simulation method for Bayesian network inference. Pseudocode of our improved algorithm is as follow:

\[
\begin{align*}
\text{ImprovedLikelihoodWeighting}(AG, n, O_b, O_d, \Omega, m) \\
\text{Input: } & AG \text{ — a network attack graph;} \\
& n \text{ — effective sample number to generate;} \\
& O_b \text{ — a set of observed evidence nodes;} \\
& O_d \text{ — a set of not observed evidence nodes;} \\
& \Omega \text{ — a temporal partial ordering relation set on } O_b \cup O_d; \\
& m \text{ — inference mode, 0 for future state, 1 for current state.} \\
\text{Output: } & \Xi \text{ — a set of effective samples.} \\
\text{Algorithm: } & \\
& 01: \Xi \leftarrow \emptyset; \; i \leftarrow 0; \\
& 02: \textbf{while} (i < n) \\
& 03: \quad w_i \leftarrow 1; \; C \leftarrow \emptyset; \\
& 04: \quad \textbf{for} (\text{each node variable } X \in AG) \\
& 05: \quad \text{if } (X \leftarrow \text{False}, F_i \leftarrow \emptyset, B_i \leftarrow \emptyset; \\
& 06: \quad \text{end for (04)} \\
& 07: \quad \textbf{for} (\text{each node variable } X \in S_b) \\
& 08: \quad \text{Mark } X \text{ as sampled;} \\
& 09: \quad \text{the sampling result according to } \Pi_b; \\
& 10: \quad \textbf{end for (07)} \\
& 11: \quad \text{converged} \leftarrow \text{False}; \\
& 12: \quad \textbf{while} (\text{converged}) \\
& 13: \quad \text{converged} \leftarrow \text{True}; \\
& 14: \quad \textbf{for} (\text{each node variable } X \leftarrow \text{True in } AG) \\
& 15: \quad \text{for (each node variable } Y \in \text{Con}(X)) \\
& 16: \quad \text{if (edge } X \rightarrow Y \text{ is not sampled) then} \\
& 17: \quad \text{if (UpdateAttackGraph}(AG, X, Y) \text{) then} \\
& 18: \quad \quad \text{converged} \leftarrow \text{False}; \\
& 19: \quad \quad \text{end if (17)} \\
& 20: \quad \quad \text{end if (16)} \\
& 21: \quad \quad \textbf{end for (15)} \\
& 22: \quad \quad \textbf{end for (14)} \\
& 23: \quad \quad \textbf{end while (12)} \\
& 24: \quad \textbf{for} (\text{each node variable } X \in O_b) \\
& 25: \quad X \leftarrow \text{True}; \\
& 26: \quad w_i \leftarrow \text{SelectEvidenceCausation}(AG, X); \\
& 27: \quad w_i \leftarrow w_i * w_{SC}; \\
& 28: \quad \textbf{end for (24)} \\
& 29: \quad \textbf{if } (m = 1) \text{ then} \\
& 30: \quad \quad b \leftarrow \text{Transientize}(AG, O_d, O_b); \\
& 31: \quad \quad \textbf{end if (29)} \\
& 32: \quad \textbf{if } (b \land \text{PartialRelationSatisfied}(AG, O_d \cup O_b, \Omega)) \text{ then} \\
& 33: \quad \quad \xi, AG \leftarrow AG; \; \xi, w \leftarrow w_i; \\
& 34: \quad \quad \Xi \leftarrow \Xi \cup \{\xi\}; \; i \leftarrow i + 1; \\
& 35: \quad \quad \textbf{end if (32)}; \\
& 36: \quad \quad \textbf{end while (02)} \\
& 37: \quad \textbf{return } \Xi; \\
& \end{align*}
\]

The 2nd line of the pseudocode is an outside loop control statement which drives the algorithm to generate \( n \) effective samples in one run. In the loop, effective samples are generated and added into a sample set \( \Xi \) which will eventually be returned.

Pseudocode 3–35 is to generate an effective sample, which could be regarded as one potential attack scenario. This procedure can be further divided into five stages:

1) Initialization (3–6). In this stage, firstly two variables \( w_i \) and \( C \) are initialized. Here \( w_i \) will be used to hold the weight of the sample and \( C \) will be used to hold the node pairs that temporally cannot be updated for their causation relationship. Then each node \( X \) in the attack graph is set to \( \text{False} \) and two assistant set variable \( F_X \) and \( B_X \) are initialized to empty. \( F_X \) will be used to hold the nodes whose value are set to \( \text{True} \) by \( X \). \( B_X \) will be used to hold the nodes who set \( X \) to \( \text{True} \). From another point of view, \( F_X \) and \( B_X \) respectively hold the forward and backward causation pointers of \( X \).

2) Nodes sampling (7–23). In this stage, all the nodes in \( AG \) (except those nodes in \( O_b \)) will be sampled. Firstly, in line 7–10, root nodes are sampled according to the initial node belief distribution \( \Pi_b \). Then, in line 11–23, \( AG \) is circularly updated until no more changes occur. In each cycle, every \( \text{True} \) value node \( X \) is checked out and a subfunction \( \text{UpdateAttackGraph} \) (for space limitation, pseudocodes of all the subfunctions are given out in An-
nex A of this paper) is called on each pair of X and its
descendant node Y iff the edge $X \rightarrow Y$ is not sampled.

3) Observed evidence causation selection (24–28). In
this stage, for each observed evidence node $X$ in $O_{10}$, a
subfunction SelectEvidenceCausation is called on $X$ to
randomly select a causation set from $Pre(X)$ to denote
what set $X$ to True. At the same time, the occurrence
probability of this chosen causation set will affect the
weight of the sample.

4) Transientization (29–31). If what we need to infer is
the current state of the network, then the sample should
be reshaped to represent a budding (not fully developed)
attack scenario. The processing transientization is based
on the idea that although some evidence nodes in $O_{10}$ may
equal to True in the sample, they actually represent evi-
dence that will be observed only in the future (currently
only the ones in $O_{10}$ are observed). So, correspondingly,
the actions that trigger the evidence also have not oc-
curred yet. This means, in order to reshape the sample
to represent current state, all these nodes should be set to
False. In our algorithm, this processing will be performed
through a subfunction Transientize.

5) Sample effectiveness verification (32–35). In this
stage, a subfunction PartialRelationSatisfied is called to
check whether the temporal partial ordering relations
among the evidence nodes conform to the causation rela-
tions among the action nodes.

By running the algorithm, we can get a set of samples
which not only have node values generated under the
given probability distribution, but also definitely conform
to the temporal partial ordering relations among evidence.
Then, to use this sample set, a node belief computation
function is defined as follow:

---

```plaintext
07: if (X=1) then
08: N_x ← N_x + ξ w;
10: end if (08)
11: end for (07)
12: W ← W + ξ w;
13: end for (06)
14: for (each node variable X in AG)
15: M ← M ∪ {P_x∈N_c(W)};
16: end for (14)
17: return M;
```

---

By running this function, a set $M$ will be returned
which contains all the node belief values for later queries.
By inputting different intrusion evidence sequences
which correspond to different observation timepoints,
we can get an attack graph node belief distribution sequence
$β = Π_1, Π_2, \cdots, Π_k$ to represent security state evolvement.

5. Node Belief Computation Examples

In order to exemplify the improved likelihood-weighting
algorithm, we design and implement a Java program to
perform following experiments:

5.1. Comparison with Bayesian Inference

Firstly, we use the variable elimination algorithm (a tra-
ditional Bayesian inference algorithm) to compute the
posterior node belief values of the attack graph illus-
trated in Figure 3. The result is listed in Table 1.

In Table 1, different number $i$ denotes different infer-
ence layer. In this example, $i = 0$ denotes the inference is
performed before any evidence is observed, $i = 1$ denotes
the inference is performed after $o_2$ is observed, $i = 2$ de-
notes the inference is performed after sequence $o_2 \rightarrow o_1$
is observed and $i = 3$ denotes the inference is performed
after $o_2 \rightarrow o_1 \rightarrow o_3$ is observed.

Then we use our improved likelihood weighting algo-
rithm to perform the same inference. The result is listed
in Table 2 (10000 effective samples without transienti-
zation) and Table 3 (10000 samples with transienti-
zation).

Since traditional Bayesian network inference methods
does not support intrusion evidence ordering, we are not
surprised to see that in Table 1, when $i > 1$, the node
belief values of intrusion path $a$ and $β$ are mirror sym-
metrical. This makes it difficult for us to judge which
path has been chosen by the attacker. However, by using
improved likelihood weighting algorithm, we can ob-
serve no matter in Table 2 or Table 3, the node belief
values of path $β$ are all higher than path $a$, indicating it is
more likely to have been chosen by the attacker.
Table 1. Node belief values (use traditional inference).

<table>
<thead>
<tr>
<th>i</th>
<th>$\pi(s_{i})$</th>
<th>$\pi(s_{j})$</th>
<th>$\pi(s_{k})$</th>
<th>$\pi(s_{l})$</th>
<th>$\pi(s_{m})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.250</td>
<td>0.250</td>
<td>0.063</td>
<td>0.063</td>
</tr>
<tr>
<td>1</td>
<td>1.000</td>
<td>0.333</td>
<td>0.444</td>
<td>0.111</td>
<td>0.111</td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
<td>0.500</td>
<td>0.500</td>
<td>0.167</td>
<td>0.167</td>
</tr>
<tr>
<td>3</td>
<td>1.000</td>
<td>0.619</td>
<td>0.619</td>
<td>0.524</td>
<td>0.524</td>
</tr>
</tbody>
</table>

Table 2. Node belief values (without transientization).

<table>
<thead>
<tr>
<th>i</th>
<th>$\pi(a_{i})$</th>
<th>$\pi(a_{j})$</th>
<th>$\pi(a_{k})$</th>
<th>$\pi(a_{l})$</th>
<th>$\pi(a_{m})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.500</td>
<td>0.500</td>
<td>0.125</td>
<td>0.125</td>
<td>0.031</td>
</tr>
<tr>
<td>1</td>
<td>0.556</td>
<td>0.889</td>
<td>0.222</td>
<td>0.222</td>
<td>0.056</td>
</tr>
<tr>
<td>2</td>
<td>0.833</td>
<td>0.833</td>
<td>0.333</td>
<td>0.333</td>
<td>0.083</td>
</tr>
<tr>
<td>3</td>
<td>0.746</td>
<td>0.746</td>
<td>0.556</td>
<td>0.556</td>
<td>0.508</td>
</tr>
</tbody>
</table>

Table 3. Node belief values (with transientization).

<table>
<thead>
<tr>
<th>i</th>
<th>$\pi(s_{i})$</th>
<th>$\pi(s_{j})$</th>
<th>$\pi(s_{k})$</th>
<th>$\pi(s_{l})$</th>
<th>$\pi(s_{m})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.498</td>
<td>0.503</td>
<td>0.122</td>
<td>0.132</td>
<td>0.031</td>
</tr>
<tr>
<td>1</td>
<td>0.504</td>
<td>1.000</td>
<td>0.127</td>
<td>0.246</td>
<td>0.028</td>
</tr>
<tr>
<td>2</td>
<td>0.792</td>
<td>1.000</td>
<td>0.190</td>
<td>0.405</td>
<td>0.044</td>
</tr>
<tr>
<td>3</td>
<td>0.663</td>
<td>1.000</td>
<td>0.409</td>
<td>0.744</td>
<td>0.342</td>
</tr>
</tbody>
</table>

Table 4. Our inference result.

<table>
<thead>
<tr>
<th>i</th>
<th>$\pi(s_{i})$</th>
<th>$\pi(s_{j})$</th>
<th>$\pi(s_{k})$</th>
<th>$\pi(s_{l})$</th>
<th>$\pi(s_{m})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.129</td>
<td>0.248</td>
<td>0.502</td>
<td>0.065</td>
</tr>
</tbody>
</table>

Then, to testify that our improved likelihood weighting algorithm can process attack graphs that contain directed cycles, we run the program to compute node belief values for Figure 2. Assuming in initial state the attacker occupies $s_{1}$ and that every other used probability is $0.5$, the inference result is listed in Table 4 (10000 effective samples, and since the graph has no evidence nodes, the result is same no matter the samples are transientized or not).

5.2. Comparison with HCPN-Based Inference

As aforementioned, in HCPN-based inference, empirical formulas are defined to reevaluate the security state of the network after intrusion evidence is observed. Comparatively, our algorithm is not dependent on any empirical formula, which makes the inference results more rational. To prove that, we modify the Figure 3 example to the one illustrated in Figure 5.

Comparing with Figure 3, in Figure 5 three additional nodes $a_{7}, s_{7}, o_{4}$ with the corresponding edges are added. Meanwhile, some of the action-evidence relations are modified and all the probabilities are explicitly labeled on the edges.

Similar to Figure 3, under the initial state the attacker whose final attack goal is also $g_{1}$ is assumed to occupy resource $s_{1}$ with probability 1.0. But during the attack, an evidence sequence $o_{1} \rightarrow o_{2} \rightarrow o_{3} \rightarrow o_{4}$ is observed.

Using the HCPN-based inference method, we can get node belief values listed in Table 5 (since HCPN only defines the belief value of resource nodes, nodes of other types are not listed).

Then we run our improved likelihood weighting inference program to perform the same computation. The result is listed in Table 6 (10000 effective samples without transientization) and Table 7 (10000 effective samples with transientization).

![Figure 5. A network attack graph.](image-url)
We can observe that in Table 5, from layer 1 to 3, different actions denoting different attack paths are assumed to be conducted by the attacker. In these layers, inferred node belief values of intrusion path $\pi_a$ are all higher than the values of path $\pi_4$. That is mainly due to the different probability values assigned to the two paths. In layer 4, the predominant attack path $\pi_a$ is excluded from further consideration as $\pi_a$ can only be triggered by $a_7$ which is on attack path $\pi_4$. That judgment is quite reasonable. However, we find that most node belief values on attack path $\pi_4$ are still not increased. It is due to the empirical formulas defined in HCPN-based inference method only update the belief values of the successor nodes of $\pi_a$ (obviously inconsistent with our intuition and what we usually see in most inference models that there should be a backward belief propagation procedure).

In comparison, in Table 6 and Table 7, no action is needed to be assumed to perform the inference. From layer 1 to 3, inferred node belief values of path $\pi_a$ are all higher than the values of path $\pi_4$. And in layer 4, path $\pi_4$ is confirmed by the observation of $\pi_4$, with all the belief values of that path set to 1.0 (this is the backward belief propagation we are expecting).

### 5.3. Algorithm Performance Evaluation

We adjust the specified number of effective samples (i.e. $m$), then record the CPU time that is used to generate the sample set. Figure 6 illustrates three performance curves which respectively correspond to the above three examples. The hardware and software environment of the program is: Intel Core2 Duo CPU 2.00GHz, 2GB DDR2 Memory, Windows XP Professional (with Service Pack 2), Sun JDK 1.6.0_10-rc.

Figure 6 shows that for a certain attack graph, the CPU time to generate a sample set is basically proportional to the number of the samples. Through a detailed analysis it can be found that the sampling time consumption is mainly determined by two facts: 1) the node number $N$ of the attack graph and 2) the evidence temporal partial ordering relation set $\Omega$. According to Definition 1, in any attack graph the prerequisite node number of a single node is always below $N$, so we may define a constant $T_{max}$ and use $T_{max} \cdot N$ to denote the upper bound of the time used to process a node. And the time to generate a full sample will be less than $N^* (T_{max} \cdot N)$. On the other hand, checking against the partial ordering relation set $\Omega$ may force us to discard some generated samples. To con-
Figure 6. CPU time curves of sampling.

Figure 7. Components of asset value.

trol this fact, we may specify a maximal try number \( M \). Once \( M \) samples have been generated (no matter how many effective samples are there), the program will cease sampling. In conclusion, for any attack graph, the sampling procedure can always finish in \( T_{\text{max}} \times M \times N^2 \). In other words, the algorithm has quadratic computational complexity.

6. Security Assessment & Enhancement

In this section, based on the above node belief computation algorithm, we propose a model for assessing network security level and performing security enhancement.

6.1. Security Assessment

Generally, the overall security level of a concerned network is mainly determined by three factors: 1) threat of the network, 2) vulnerability of the network and 3) influence of the potential attacks. In the previously proposed model, the former two factors have been dealt with (by IDS alerts indicating threats and network attack graph itself indicating vulnerabilities). However, we still have a problem with how to model the influence of potential attacks. In this section, we introduce a concept of asset value breakage rate to quantify it.

Asset value breakage rate is the ratio of the lost asset value to the overall asset value, illustrated in Figure 7. Since we often use asset CIA (confidentiality, integrity and availability) value to achieve more particular quantification, we introduce asset CIA breakage nodes into network attack graph and extend Definition 1 into Definition 3.

**Definition 3.** An extended attack graph is a 12-tuple directed graph \( AG = (S, S_0, A, G, O, E, \Delta, \Phi, \Theta, \Pi, P, Y) \) where \( S, S_0, A, G, O, E, \Delta, \Phi, \Theta, \Pi \) are the same elements as defined in Definition 1 and:

- \( P = (P_C \cup P_I \cup P_A) \) is a set of asset breakage nodes where \( P_C \) is a set of asset confidentiality breakage nodes, \( P_I \) is a set of asset integrity breakage nodes and \( P_A \) is a set of asset availability breakage nodes. Values of each node variable \( \rho_X \) (\( X = C, I, A; i = 1, \cdots, N \) where \( N \) is the total asset number) all lie in \([0, 1]\), denoting the breakage percentage of every asset in particular aspect. Apart from that, we define a function \( \lambda : P \rightarrow [0, +\infty) \) to map each asset to its overall value in confidentiality, integrity and availability. So we can use \( \rho_i \times \lambda (\rho_i) \) to denote the absolute loss value of an asset in CIA.

- \( E = (E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5) \) is a finite set of edges which link graph nodes together. Here \( E_1, E_2 \) and \( E_3 \) share the same definition as in Definition 1, while \( E_4 \subseteq S \times P \) denotes that if the attacker gains certain resources, she will do damage to certain assets, and \( E_5 \subseteq A \times P \) denotes that if the attacker executes certain actions, she will do damage to certain assets.

- \( Y = \{ \nu_1: (s, \rho) \rightarrow [0, 1], \nu_2: (a, \rho) \rightarrow [0, 1] \} \) is the asset breakage conductivity rate distribution where \( \nu_1 \) denotes when the attacker gains a resource, how much damage will she do to an asset and \( \nu_2 \) denotes when the attacker executes an action, how much damage will she do to an asset. Just like the other prior conditional probability distributions, values of \( Y \) also lies in \([0, 1]\) where a larger value denotes a greater potential damage.

Based on Definition 3, we can use a function \( \tau (P) \) to quantify the network security level:

\[
\tau (P) = \frac{\sum (1 - \rho) \lambda (\rho)}{\sum \lambda (\rho)} = \frac{\sum (1 - \rho_C) \lambda (\rho_C) + \sum (1 - \rho_I) \lambda (\rho_I) + \sum (1 - \rho_A) \lambda (\rho_A)}{\sum \lambda (\rho_C) + \sum \lambda (\rho_I) + \sum \lambda (\rho_A)}
\]
In the above equation, \( \tau(P) \) is the function to compute the normalized asset residual value. The right expression uses asset residual value as the numerator and asset overall value as the denominator.

Just like other network attack graph node variables, all the belief values of asset breakage nodes also can be computed by the inference algorithm in Section 4. So, by inputting different evidence sequences corresponding to different observation timepoints, eventually we can get a sequence \( \tau_0 \tau_1 \cdots \tau_k \) indicating the security evolution.

6.2. Security Enhancement

Broadly speaking, as long as a measure can help enhancing network security, it is referred to as a network security enhancement measure. In most cases, the implementation of a security enhancement measure may affect a network attack graph in two ways:
1) it changes the structure of the attack graph, or
2) it changes the conditional probability distributions of the attack graph including \( \Delta, \Phi, \Theta, \Pi, \gamma \).

However, since commonly the implementation of a security enhancement measure will cut off certain intrusion paths, the resulting (enhanced) attack graph is often the sub-graph of the original attack graph. Based on this, we can always convert the above situation 1 into situation 2 by adjusting certain conditional probability.

For example, in the previous example illustrated in Figure 2, if the vulnerability on Master is patched, we need not generate a new attack graph, but set the conducting probability of \( a_1 \) and \( a_2 \) to 0.0.

On this basis, we introduce a security enhancement measure tuple \( (M, \mathcal{S}, \nu) \):

- \( M = \{m_1, \cdots, m_k\} \) is a candidate measure set.
- \( \mathcal{S} : 2^M \rightarrow 2^\Gamma \) is a function which maps a combination of measures to a rectified attack graph probability distribution \( \Gamma = \{\Delta, \Phi, \Theta, \Pi, \gamma\} \).
- \( \nu : 2^M \rightarrow R^+ \) is a function which maps a combination of measures to its implementation cost.

With the above security measure tuple, we can perform the following analysis:

1) **Static Security Enhancement.** This analysis is to find the best combination of security measures to be implemented before any potential intrusion happens. A typical usage of this analysis is to enhance a network system before it is placed online. To achieve this, all candidate measure combinations need to be iterated. For each measure combination \( M_c \in 2^M \), we set the probability distribution to \( \Gamma' = \delta(M_c) \) and recompute the network normalized asset residual value \( \tau_0' \). Then the net profit of \( M_c \) is:
\[
u(M_c) = (\tau_0' - \tau_0) \times \text{Overall Asset Value} - \nu(M_c),
\]
where \( \tau_0 \) is the normalized asset residual value when no security measure is implemented (\( M_c = \emptyset \)). Finally, by sorting these measure combinations according to their net profits, we can easily choose the greatest one as the optimal enhancement solution.

2) **Dynamic Security Enhancement.** This analysis is to find the best measure combination when intrusion is happening (or has happened). To achieve this, we firstly need to use the previous inference algorithm to generate a set \( \Xi_T \) of transientized attack samples. Then we iterate all of the candidate measure combinations. For each combination \( M_c \in 2^M \), we rectify the graph probability distribution to \( \Gamma' = \delta(M_c) \). After doing this, we re-sample (a process same to line 11–23 of the **Improved LikelihoodWeighting** algorithm) \( \Xi_T \) according to the new distribution and get a new set \( \Xi_S \) which will be actually used to compute the network normalized asset residual value \( \tau_S' \). Then the net profit of \( M_c \) is:
\[
u(M_c) = (\tau_S' - \tau_S) \times \text{Overall Asset Value} - \nu(M_c),
\]
where \( \tau_S \) is a normalized asset residual value when no measure is implemented (\( M_c = \emptyset \)). Finally, by sorting these measure combinations according to their net profits, we can easily choose the greatest one as the optimal enhancement solution.

7. Assessment & Enhancement Examples

To exemplify the above security assessment and enhancement method, in one experiment we generated an attack graph for a two tier network system. Based on it, we performed corresponding security assessment and used our enhancement method to find out optimal security enhancement measure combinations for static and dynamic security enhancement respectively.

7.1. Basic Posterior Inference

Figure 8 illustrates the topology of the two tier network. In this network, four intranet servers were connected to a switch which was further connected to the Internet by a router. A firewall was placed between the two devices to perform package filtering, besides an IDS was connected to a mirror port of the switch to monitor the outflow and inflow of the servers.

We assumed a scenario that an attacker on the Internet intends to use her personal computer to attack this network. The final goal of the attacker was to get the ROOT privilege on server3 and steal its business data.

For further analysis, we firstly need to generate a network attack graph to find out all the potential intrusion paths. So we scanned the online devices and servers and found out six vulnerabilities (listed in Table 8). Addi-
Figure 8. A two tier network system.

Table 8. Host vulnerabilities.

<table>
<thead>
<tr>
<th>device</th>
<th>OS</th>
<th>application</th>
<th>vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>server1</td>
<td>windows nt 4.0</td>
<td>serv-u ftp 3.0</td>
<td>cve-2004-0330, cve-2004-1992</td>
</tr>
<tr>
<td>server2</td>
<td>windows 2000</td>
<td></td>
<td>cve-2003-0533</td>
</tr>
<tr>
<td>server3</td>
<td>redhat linux 7.3</td>
<td>oracle 9i, cvs 1.11</td>
<td>cve-2004-0417, cve-2004-0415</td>
</tr>
<tr>
<td>server4</td>
<td>redhat linux 7.3</td>
<td>apache 1.3.23</td>
<td>cve-2002-0392</td>
</tr>
</tbody>
</table>

Additionally, we found that the firewall is configured to permit and only permit the Internet user to access intranet servers through HTTP protocol. By importing these information into a network attack graph building system developed by us (design concept of this system mainly follows the framework proposed in [16]), we get a network attack graph shown in Figure 9.

In Figure 9, resource state nodes are represented in circles while action nodes are represented in rectangles. The top row in the figure (11 resource state nodes) includes the 6 vulnerabilities on the servers and the 5 low level privileges which can be used by anyone to access the servers. However, owing to the firewall, initially the attacker can only access server4’s HTTP service and exploit the apache vulnerability (this exploitation is represented in the figure with the action node right below the top resource state node row). After that, the attacker may get the USER privilege of server4 and use this server as a stepping stone to perform further intrusion (mainly by exploiting the rest vulnerabilities listed in Table 8). According to Figure 9, to the maximum extent, the attacker can get the USER privilege of server1, server3 and server4 as well as the ROOT privilege of server2 and server3 (represented by the other 5 circles in the figure exclude the top row).

Then we should assigned conditional probability distributions to the generated graph. In this stage, we mainly used data sources such as CVSS [17] and OSVBD [18] complemented with expertise knowledge. For example, in CVSS, a score metric named Exploitability are defined to indicate the difficulty for an attacker to exploit vulnerability. So we decide to use this metric to evaluate the success rate of an action by the following transformation:

$$\phi(a) = 1.0 - e^{-\text{Exploitability of } a}$$

With all prior conditional probability distributions assigned, we were able to perform posterior inference according to observed intrusion evidence. As an example for exemplification, we assumed that an IDS alert sequence is observed as in Table 9:

By running the improved likelihood weighting algorithm, we computed node belief values for each inference layer. Due to space limitation, detailed result is not given out here. But in Annex B this security evolvement procedure is illustrated graphically. In each figure of the annex, a darker node is used to represent a greater node belief value. We see that with more evidence observed, belief values of some graph nodes increase rapidly, indicating intrusion paths that are most probably chosen.

7.2. Security Assessment

Since our aim is to assess security level of the network and find out an optimal enhancement solution, we selected 5 important service assets from the network system whose CIA values are listed in Table 10 (in thousands SUS). Correspondingly, we introduced into the attack graph 15 corresponding asset breakage nodes.

Meanwhile, we quantified the asset breakage conductivity rate between these 15 nodes and the aforementioned 5 resource state nodes which represent the escalated privileges that may be gained by the attacker. The conductivity rates between them are listed in Table 11.

Table 9. Observed alerts.

<table>
<thead>
<tr>
<th>id</th>
<th>exploited vulnerability</th>
<th>source</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cve-2002-0392</td>
<td>pc</td>
<td>server4</td>
</tr>
<tr>
<td>2</td>
<td>cve-2003-0533</td>
<td>server4</td>
<td>server2</td>
</tr>
<tr>
<td>3</td>
<td>cve-2004-0417</td>
<td>server3</td>
<td>server3</td>
</tr>
</tbody>
</table>

Table 10. Important assets and their CIA values.

<table>
<thead>
<tr>
<th>id</th>
<th>asset</th>
<th>host</th>
<th>$\lambda(p_C)$</th>
<th>$\lambda(p_I)$</th>
<th>$\lambda(p_A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ftp</td>
<td>server1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>file</td>
<td>server2</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>database</td>
<td>server3</td>
<td>100</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>cvs</td>
<td>server4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>apache</td>
<td>server4</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
After doing this, we recomputed the node belief values for each inference layer and get 2 normalized asset residual value sequences \( \tau_0, \tau_1, \tau_2, \tau_3 \) listed in Table 12 and Table 13 (without and with transientization respectively). These sequences are graphically illustrated in Figure 10 to reveal the evolvement of network security. In the figure we can observe that the asset residual values generated with transientization are always greater than the ones without transientization. This is reasonable since with all condition unchanged, the current security level of a network is always higher than its future security level, because from current timepoint to the future the attacker gets additional time to perform more intrusion.

7.3. Security Enhancement

Then, for enhancement, we analyzed the network system and listed 11 plainest security measures as candidates in Table 14. These measures include patching the vulnerabilities on the servers and disabling low level privilege accounts on them. Additionally, we identified a measure of configuring the firewall to deny all incoming access including HTTP. Costs of these security measures were also analyzed and listed in the table (in thousands $US).

By using the security enhancement methods proposed in Section 6, we eventually got all the net profit of the security measure combinations. In Table 15 we list the
Figure 10. Evolvement of network security.

Table 12. Assessment result (without transientization).

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σμ(ρ)</td>
<td>453.000</td>
<td>453.000</td>
<td>453.000</td>
<td>453.000</td>
</tr>
<tr>
<td>Σρμ(ρ)</td>
<td>66.472</td>
<td>92.342</td>
<td>161.153</td>
<td>192.790</td>
</tr>
<tr>
<td>Σ(1−ρ)μ(ρ)</td>
<td>386.528</td>
<td>360.658</td>
<td>291.847</td>
<td>260.210</td>
</tr>
<tr>
<td>τ_i</td>
<td>85.33%</td>
<td>79.62%</td>
<td>64.43%</td>
<td>57.44%</td>
</tr>
</tbody>
</table>

Table 13. Assessment result (with transientization).

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σμ(ρ)</td>
<td>453.000</td>
<td>453.000</td>
<td>453.000</td>
<td>453.000</td>
</tr>
<tr>
<td>Σρμ(ρ)</td>
<td>0.909</td>
<td>13.548</td>
<td>91.443</td>
<td>173.655</td>
</tr>
<tr>
<td>Σ(1−ρ)μ(ρ)</td>
<td>452.091</td>
<td>439.452</td>
<td>361.557</td>
<td>279.345</td>
</tr>
<tr>
<td>τ_i</td>
<td>99.80%</td>
<td>97.01%</td>
<td>79.81%</td>
<td>61.67%</td>
</tr>
</tbody>
</table>

top 5 best security enhancement measure combinations for static security enhancement (SSE) and in Table 16 we list the top 3 best combinations (of each inference layer) for dynamic security enhancement (DSE).

8. Conclusions

As network attack graphs are more and more widely used in real-time network security analysis, the problem of how to use observed intrusion evidence to compute attack graph node belief becomes a concerned issue. Although Bayesian network is an ideal mathematic tool for posterior inference, it can not be directly used in attack graph-based inference for the following limitations: 1) There may exist directed cycles in an attack graph, but in a Bayesian network this is not permitted. 2) There are

<table>
<thead>
<tr>
<th>id</th>
<th>security measure</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>patch CVE-2004-0330 on server1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>patch CVE-2004-1992 on server1</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>patch CVE-2003-0533 on server2</td>
<td>5.0</td>
</tr>
<tr>
<td>4</td>
<td>patch CVE-2004-0417 on server3</td>
<td>5.0</td>
</tr>
<tr>
<td>5</td>
<td>patch CVE-2004-0415 on server3</td>
<td>1.0</td>
</tr>
<tr>
<td>6</td>
<td>patch CVE-2002-0392 on server4</td>
<td>1.0</td>
</tr>
<tr>
<td>7</td>
<td>disable GUEST account on server1</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>disable GUEST account on server2</td>
<td>50.0</td>
</tr>
<tr>
<td>9</td>
<td>disable GUEST account on server3</td>
<td>60.0</td>
</tr>
<tr>
<td>10</td>
<td>disable GUEST account on server4</td>
<td>10.0</td>
</tr>
<tr>
<td>11</td>
<td>add HTTP filtering rule on firewall</td>
<td>10.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>combination</th>
<th>cost</th>
<th>net gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{6}</td>
<td>1.0</td>
<td>65.14</td>
</tr>
<tr>
<td>2</td>
<td>{1, 6}</td>
<td>1.1</td>
<td>65.04</td>
</tr>
<tr>
<td>3</td>
<td>{2, 6}</td>
<td>1.1</td>
<td>65.04</td>
</tr>
<tr>
<td>4</td>
<td>{1, 2, 6}</td>
<td>1.2</td>
<td>64.94</td>
</tr>
<tr>
<td>5</td>
<td>{5, 6}</td>
<td>2.0</td>
<td>64.14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>layer</th>
<th>id</th>
<th>combination</th>
<th>cost</th>
<th>net gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>{1, 2, 6, 7}</td>
<td>2.2</td>
<td>40.87</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{1, 2, 6}</td>
<td>1.2</td>
<td>40.77</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>{1, 2, 5, 6}</td>
<td>2.2</td>
<td>40.67</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>{1, 2, 3, 4}</td>
<td>10.2</td>
<td>55.20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{2, 3, 4}</td>
<td>10.1</td>
<td>54.86</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>{3, 5, 7}</td>
<td>7.0</td>
<td>54.70</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>{1, 2, 4, 6}</td>
<td>6.2</td>
<td>51.23</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{1, 2, 4}</td>
<td>5.2</td>
<td>51.23</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>{1, 2, 4, 5}</td>
<td>6.2</td>
<td>50.73</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>{1, 2, 5}</td>
<td>1.2</td>
<td>11.41</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>{1, 2, 5, 6}</td>
<td>2.2</td>
<td>11.01</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>{1, 2, 4}</td>
<td>5.2</td>
<td>10.21</td>
</tr>
</tbody>
</table>

Copyright © 2011 SciRes. JIS
temporal partial ordering relations among intrusion evidence, but we can not use a Bayesian network to model this information. 3) A Bayesian network cannot be directly used to infer both the current and the future security state of a network. In this work, we resolve these critical problems by developing an approximate Bayesian inference algorithm—the likelihood weighting algorithm. We give out all the pseudocodes of the algorithm and use several examples to show its advantage. Essentially, we believe this algorithm is not only limited in attack graph-based analysis, but also can be applied in other techniques employing traditional Bayesian posterior inference.

Based on the algorithm and its underlying model, we further propose a novel network security assessment and enhancement method. In this paper, we use a small network system to exemplify how to use this method to quantify the overall network security level and make decisions about what security measures are most cost-effective for network security enhancement.

9. Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant Nos.60605019; the National High-Tech Research and Development Plan under Grant Nos.2007AA01Z473; the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No.20070248002.

The authors would also like to thank all the members of Shanghai Key Laboratory for Information Security Integrated Management Technology Research.

If anyone is interested in the algorithms of this article, please feel free to contact leony7888@hotmail to get the java source codes.

10. References


Annex A: Pseudocode of Subfunctions

SF1. UpdateAttackGraph—called in ImprovedLikelihoodWeighting

```
UpdateAttackGraph(AG, X, Y)
Input: AG—an attack sample
X—an ancestor node
Y—a descendant node
Output: a boolean variable indicating if AG is updated
Alg: 01: updated—False;
02: if (Y cand not sampled) then
03: if (for each node X’ in Pre(Y) variable X’=True) then
  04: IsCriticalCausation(AG, Y, X’, Ø)=False then
  05: Y—the sampling result according to \( \delta(Y) \);
  07: if (Y= True) then
  08: for (each node X’ in Pre(Y))
  09: \( F_X \leftarrow F_X \cup \{Y\} \);
  10: \( B_Y \leftarrow B_Y \cup \{X\} \);
  11: end for (08)
  12: updated—True;
  13: for (each node Z in Con(Y))
  14: if (edge Y→Z is not sampled) then
  15: UpdateAttackGraph(AG, Y, Z);
  16: end if (14)
  17: end for (13)
  18: end if (07)
  19: mark Y as sampled;
  20: for (each node X’ in Pre(Y))
  21: mark edge X’→Y as sampled;
  22: end for (20)
  23: end if (04)
  24: end if (03)
  25: end if (02)
  26: if (Y is other type node) then
  27: if (!IsCriticalCausation(AG, Y, X, Ø)) then
  28: \( Y_{old} \leftarrow Y; \)
  29: \( y \leftarrow \) sampling result by \( \delta(X, Y) \) or \( \phi(X, Y) \);
  30: if (y=True) then
  31: Y=True;
  32: \( F_X \leftarrow F_X \cup \{Y\} \);
  33: \( B_Y \leftarrow B_Y \cup \{X\} \);
  34: updated—True;
  35: if (Y= False) then
  36: for (each node Z in Con(Y))
  37: if (edge Y→Z is not sampled) then
  38: UpdateAttackGraph(AG, Y, Z);
  39: end if (37)
  40: end for (36)
  41: end if (35)
  42: end if (30)
  43: mark edge X→Y as sampled;
```

SF2. IsCriticalCausation—called in UpdateAttackGraph and later SFs

```
IsCriticalCausation(AG, X, Y, H)
Input: AG—an attack sample
X—a candidate causation node
Y—a candidate affection node
H—a set to hold historically processed nodes
Output: a boolean variable indicating if X is a critical causation of Y
Alg: 01: if (Y is an action node) then
  02: if (Y∈H ∨ B_Y=Ø) then
  03: r=False;
  04: end if (02)
  05: if (X∈B_Y ∨ (for each node Z in B_Y, IsCriticalCausation(AG, X, Z, H∪\{Y\})=True)) then
  06: r=True;
  07: else (05)
  08: r=False;
  09: end if (05)
  10: end if (01)
  11: if (Y is other type node) then
  12: if (Y∈H ∨ B_Y=Ø) then
  13: r=False;
  14: end if (12)
  15: if ((X)=B_Y ∨ (for each node Z in B_Y, IsCriticalCausation(AG, X, Z, H∪\{Y\})=True)) then
  16: r=True;
  17: else (15)
  18: r=False;
  19: end if (15)
  20: end if (11)
  21: return r;
```

SF3. SelectEvidenceCausation—called in ImprovedLikelihoodWeighting

```
SelectEvidenceCausation(AG, X)
Input: AG—a network attack graph
X—an observed evidence node (whose value is True)
Output: the occurrence probability of the chosen causation set
Alg: 01: \( p_{x}=0; \)
  02: \( \Phi=\emptyset; \Phi=\{Y|Y∈Pre(X)\land Y=True\};\)
  03: for (each non-empty subset \( \psi \) of \( \Phi \))
  04: \( p_{x}=\prod_{\psi \in \Phi}(1-\Phi(Y, X)) ;\)
  05: \( \Phi=\Phi \cup \{(y, p_{x})\};\)
```
SF4. Transientize—called in ImprovedLikelihood-Weighting

\[
\text{Transientize}(AG,O,O) \\
\text{Input: } AG—\text{an attack sample } O—\text{a set of observed evidence nodes} \\
\text{Output: a boolean indicating } AG’s \text{ effectiveness after transientization} \\
\text{Alg:} \ \\
\text{for (each node } X \text{ in } O) \ \\
\text{for (each action node } Y \text{ in } B_\Theta) \ \\
Y \leftarrow \text{False; } \ \\
\text{mark edge } Y \rightarrow X \text{ as not sampled; } \ \\
B_\Theta \leftarrow B_\Theta \setminus \{Y\}; F_\Theta \leftarrow F_\Theta \setminus \{X\}; \ \\
\text{end if (06)} \ \\
\text{end for (07)} \ \\
\text{end for (02)} \ \\
\text{end for (01)} \ \\
\text{converged} \leftarrow \text{False; } \ \\
\text{while } ! \text{converged} \ \\
\text{converged} \leftarrow \text{True; } \ \\
\text{for (each non-root node in } AG) \ \\
\text{if } X \text{ is an action node then} \ \\
X \leftarrow \text{False; } \ \\
\text{mark } X \text{ as not sampled; } \ \\
\text{for (each node } Y \text{ in } B_\Theta) \ \\
\text{mark edge } Y \rightarrow X \text{ as not sampled; } \ \\
B_\Theta \leftarrow B_\Theta \setminus \{Y\}; F_\Theta \leftarrow F_\Theta \setminus \{X\}; \ \\
\text{end for (23)} \ \\
\text{converged} \leftarrow \text{False; } \ \\
\text{end if (20)} \ \\
\text{else (19)} \ \\
\text{for (each node } Y \text{ in } B_\Theta) \ \\
\text{if } Y \leftarrow \text{False } \leftarrow \text{IsCriticalCausation}(AG,X,Y,O) \text{ then} \ \\
\text{mark edge } Y \rightarrow X \text{ as not sampled; } \ \\
B_\Theta \leftarrow B_\Theta \setminus \{Y\}; F_\Theta \leftarrow F_\Theta \setminus \{X\}; \ \\
\text{end if (20)} \ \\
\text{end for (19)} \ \\
\text{end if (15)} \ \\
\text{end for (02)} \ \\
\text{end for (04)} \ \\
\text{if } ! \text{pairSatisfied} \text{ then} \ \\
\text{typeI_satisfied} \leftarrow \text{False; } \ \\
\text{end if (15)} \ \\
\text{end if (11)} \ \\
\text{end for (04)} \ \\
\text{if } ! \text{existJCoversI} \text{ then} \ \\
\text{pairSatisfied} \leftarrow \text{False; } \ \\
\text{end if (11)} \ \\
\text{end for (06)} \ \\
\text{end if (07)} \ \\
\text{end if (03)} \ \\
\text{for each type I partial ordering relation } O_s / O_s \text{ in } \Omega \ \\
\text{pairSatisfied} \leftarrow \text{True; } \ \\
\text{end for (03)} \ \\
\text{end if (19)} \ \\
\text{end if (18)} \ \\
\text{end while (16)} \ \\
\text{return True;}
\]

SF5. PartialRelationSatisfied—called in ImprovedLikelihood-Weighting

\[
\text{PartialRelationSatisfied}(AG,O,O) \\
\text{Input: } AG—\text{an attack sample} \\
\Omega—\text{a temporal partial ordering relation set on } O \\
\text{Output: a boolean indicating if } AG \text{ conforms to } \Omega (\text{and is effective}) \\
\text{Alg:} \ \\
\text{typeI_satisfied} \leftarrow \text{True; } \text{typeII_satisfied} \leftarrow \text{True; } \ \\
\text{for each type I partial ordering relation } O_s / O_s \text{ in } \Omega \ \\
\text{pairSatisfied} \leftarrow \text{True; } \ \\
\text{end for (03)} \ \\
\text{end if (07)} \ \\
\text{end if (06)} \ \\
\text{end if (04)} \ \\
\text{if } ! \text{pairSatisfied} \text{ then} \ \\
\text{typeI_satisfied} \leftarrow \text{False; } \ \\
\text{end if (15)} \ \\
\text{end if (02)} \ \\
\text{end if (22)} \ \\
\text{end for (21)} \ \\
\text{end for (03)} \ \\
\text{end for (20)} \ \\
\text{return typeI_satisfied } \wedge \text{ typeII_satisfied;}
\]
Annex B: Graph Node Belief Evolvement

Without Sample Transientization

Figure 1. Layer 0–no alert is observed.

Figure 2. Layer 1–one alert is observed.
Figure 3. Layer 2–two alerts is observed.

Figure 4. Layer 3–three alerts is observed.
With Sample Transientization

Figure 1. Layer 0—no alert is observed.

Figure 2. Layer 1—one alert is observed.
Figure 3. Layer 2—two alerts is observed.

Figure 4. Layer 3—three alerts is observed.