Ranking Intrusion Likelihoods with Exploitability of Network Vulnerabilities in a Large-Scale Attack Model

Rattikorn Hewett[1] and Phongphun Kijsanayothin[2]
(Corresponding author: Rattikorn Hewett)
Department of Computer Science[1]
Texas Tech University, Lubbock, Texas, USA
Department of Electrical and Computer Engineering[2]
Naresuan University, Phitsanulok, Thailand
(Email: rattikorn.hewett@ttu.edu)
(Received Oct. 30, 2013; revised and accepted July 24, 2014)

Abstract

Network vulnerabilities are common sources of many security threats. Attack models representing chains of all possible vulnerability exploits by attackers can help locate security flaws and pre-determine appropriate preventative measures. To realize the full benefits of attack models, effective analysis is crucial. However, due to the size and complexity of the models, manually pinpointing potential critical attacks can be daunting. Thus, there is a need for an automated analysis approach. Existing techniques are either based on network topology alone or subjective prior knowledge. They do not utilize domain-specific knowledge. This paper presents an approach to automatically ranking states in an attack model in the order of their intrusion likelihoods. Using the degree of exploitability of network vulnerabilities and the Markov property, the proposed approach provides a tractable computation enhanced by domain-specific heuristic knowledge for estimating such likelihoods. The paper discusses the details of the approach, illustrates its use, and compares results with a similar existing technique with experiments on its performance.

Keywords: Attack graphs, security models, network vulnerability, network security, ranking algorithm

1 Introduction

Securing networks requires understanding of *network vulnerabilities*, which are common sources of many attacks. Such vulnerabilities include exploitable errors in configurations (e.g., ports and services enabled) or the network service software (e.g., *Apache Chunked-Code* on Apache web servers, *buffer overflow* on Windows XP SP2 operating environments, and *TNS-Listener* on Oracle software for database servers). These vulnerabilities are unavoidable as long as we need the network to provide their corresponding services. Building attack models as chains of all possible *vulnerability exploits* by attackers can help security administrators locate security flaws and predetermine appropriate preventative measures. To fully realize the practicality of attack models, effective analysis is crucial. By analysis, we mean a systematic method for extracting useful information for security management.

Much work in attack model analysis has been primarily on visualization [9, 16, 17, 22]. Although this can help security administrators assess overall threats to the network, locating hazardous situations and locations to secure networks is still a challenging task due to the size and complexity of the attack models. Besides, visualization often requires human expertise to observe and pinpoint critical information. Thus, visualization can be time consuming and may produce inconsistent findings. There is a need for an automated approach to attack model analysis that can assess network security more effectively.

Several formal approaches to automatic attack model analysis have been proposed using graph theory [10], probabilistic analysis [21] and game theory [13]. The probabilistic analysis by Sheyner et al. [21] estimates the reliability of a given node (or state) in the attack model (or attack graph) in term of the probability of an attacker reaching his goal from the node. However, the assignment of arbitrary prior probabilities of detecting each attack action makes this approach ad-hoc as it relies on subjective opinions. Jha et al. [10] introduced a graph-based approach that identifies the smallest set of exploits to be removed to prevent the network from all possible attacks shown in a given attack model. The intent is to identify the smallest set of counter-measures required to protect the network. However, the choice of an appropriate set

of counter-measures does not always depend on its size alone. Furthermore, the approach is limited to directed acyclic attack graphs (DAG). Lye and Wing [13] applies a game theoretic approach to model "rational" interactions between an attacker and a network administrator during an attack attempt. Unlike others, this approach is not a preventative approach and its application is restricted to complete attack graphs.

A recent approach to attack model analysis aims to efficiently rank the nodes of an attack model based on the likelihood of an attacker reaching these states was introduced by Mehta et al. [14]. The ranking provides useful information for determining which attack path is more vulnerable or requires more immediate attention for network protection. The approach is based on *PageRank* [4], a well-known link analysis algorithm for Google's web search engine. Unfortunately, their ranking results are not always meaningful. This is because network intrusion does not have as much freedom as web browsing where we can randomly visit any website via URLs. In network intrusion, an attacker can only advance his attack position to a node that has connectivity and vulnerability to be exploited. Thus, the approach to computing the probability of advancing each attack action to a new state requires an adjustment. Mehta el al. introduced a modified ranking algorithm to address this issue. However, all of the above approaches tend to view attack model analysis as a general problem in graph theory and only use structural topology of the attack model. None makes use of domain-specific knowledge about network security (e.g., vulnerability and degree of its exploitability) to obtain more meaningful and accurate analysis.

This paper presents an approach to automatically analyzing security attack models that ranks states in the attack model in the order of their likelihoods of being intruded by an attacker. The proposed approach is most similar to Mehta et al.'s approach [14]. However, there are a few major differences that set this work apart from previous work. First, we use knowledge about the *exploitability* of network vulnerability instead of subjective or no prior domain-specific knowledge in estimating the intrusion likelihoods as in Mehta et al.'s approach. In particular, our analysis proposes *ExploitRank*, a new heuristic ranking algorithm that uses public information on the *Common Vulnerability Scoring System* [7] as a measure for quantifying the exploitability of network vulnerability. Second, *ExploitRank* assumes that when an attacker has no more vulnerability to exploit to advance to the next state, he will give up on the current path and start finding an alternative attack path from the beginning (i.e., at initial states). In contrast, Mehta el al.'s approach assumes that an attacker may either persist on attacking the same state (analogous to browsing a web page that has links to itself) or decide to start over. We will show that these slight differences yield drastically different results and that our approach produces results that better match logics in our reasoning than those of Mehta et al.'s. The paper has the following contributions:

1) An automated framework for protecting a computer network against malicious attacks via attack models.

2) An enhanced ranking algorithm for analyzing large-scale attack models by ranking possible attack states based on their relative intrusion likelihoods.

The ExploitRank algorithm help provide priorities for network security management. The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives preliminary concepts. Sections 4 and 5 describe and illustrate our proposed approach with some experimental results. Section 7 concludes the paper.

2 Related Work

Majority of research in attack model analysis includes visualization techniques [16, 17] that have been employed to simplify an attack model. By grouping together nodes that have the same characteristics (e.g., same locality) into a single node, the resulting model is easier to view and less complicated to find ways to prevent attack paths. Because most graph visualization is semi-automated or manual, it tends to be time consuming and the results obtained can still be far too complex to be useful in practice. Our approach is automated and does not aim to simplify the view of the model but helps locate critical nodes.

Sheyner et al.'s probabilistic approach [21] employs Markov Decision Process (MDP) to estimate reliability of each node in the model with the probability of an attacker reaching his goal from a given node. This approach tends to be subjective and impractical since it requires assignments of arbitrary prior probabilities of detecting each attack action. Our approach, however, does not require such prior probability assignment.

A graph-based approach by Jha et al. [10] aims at finding a minimum set of countermeasures to guarantee that the attackers' goal states will never be reached. This is done by estimating the smallest set of attacks required to protect the network along with the smallest set of countermeasures to account for each of the attacks. Jha et al.'s approach is limited to a DAG, where each attack path, from an initial state to a goal state, is considered only once, whereas our approach can be applied to any attack graph topology.

Another approach to attack graph analysis aims at ranking graph states by their likelihoods of being attacked [14, 19]. Most of these ranking techniques are based on the well-known PageRank algorithm [4] for ranking web pages. Among these, the work that is most closely related to our approach is Mehta et al.'s approach [14] that modifies the transitions at the end of each attack path to an initial state instead of every node as used in PageRank algorithm. However, Mehta el al. treat each node reachable from a given node to have the same degree of vulnerability and exploitability. Unlike ours, none of

the above approaches exploits domain-specific knowledge about the exploitability of the network vulnerabilities.

3 Preliminaries

3.1 Terms and Concepts in Network Security

Network vulnerabilities refer to the weaknesses of a target system network, for examples, security flaws in server software (e.g., *Apache Chunked-Code*, *Oracle with TNS Listener* software) or network configurations (e.g., enabled ports and services). Known vulnerabilities are publicly available (e.g., [5, 18]). Vulnerability can be exploited when its preconditions are satisfied. These preconditions include connectivity, access privileges on relevant hosts, and network or host configurations.

A *vulnerability exploit* refers to an attacker's *action* to advance his attack. Typically, an exploit involves an *attacking host* (the source on which an attacker performs an exploit), and a *victim host* (the destination on which an attacker gains benefits after the exploit has been carried out). An exploit has two modes: *local* and *remote*. To attack, the network must have vulnerabilities and an attacker must know how to exploit them. Note that each exploit could involve one or more vulnerabilities (e.g., the "Apache Chunked-Code Buffer Overflow" exploit involves software vulnerability (e.g., Apache web server software Version 1.3) and configuration vulnerability (e.g., Apaches default port is enabled on a victim host). Similarly, vulnerability could be involved in more than one exploit.

An *attack model* or *attack graph* represents the behavior of attackers harming a network. Each node in the graph represents a state, typically specified by the relevant network attributes such as connectivity between hosts and an attacker's access privileges. Each link represents an action that an attacker takes to gain his access control in the network. Starting from a set of initial nodes, an attacker can take an action that exploits the network vulnerability to reach a set of states satisfying the attacker goal (e.g., obtain a root privilege on a database server). There are various forms of attack graphs (e.g., access [1], host-centric [8], and network-based [21]). However, they all use the same level of abstraction of attacker's actions. Each attack model can have multiple initial states as well as multiple goal states.

3.2 Link Analysis and the PageRank Algorithm

Ranking web pages is an important function of an Internet search engine. Approaches to ranking web pages are based on a link analysis, where we assign weights to a hyperlinked set of web pages to approximate the relative importance of each web page within the set. Variations of link-based ranking algorithms include *PageRank* [4] and *HITS* [12]. Because of its accuracy and efficiency, the Google's *PageRank* algorithm becomes one of the most predominant ranking algorithms, whose main concepts will be briefly described below.

The rank value of a web page indicates a probability that a web surfer randomly clicking on links will end up visiting the page. Thus, *the sum of page rank values over all of the considered web pages must be one.* It is assumed that the initial approximation of this probability would be equally distributed among all web pages in the considered collection. *PageRank* algorithm simulates the clicking behavior of a web surfer who can visit a web page either via an incoming link to the page or picking a URL of the page at random. The surfer who randomly clicks on links will eventually stop. At any surfing stage, a *damping factor* is the probability that the web surfer will continue surfing using hyperlinks.

Let $r_t(v)$ be the probability of visiting web page v at the time t, d be a damping factor and V be a set of web pages under consideration. For a page v, $out(v)$ and $in(v)$ is a set of web pages in V with an outgoing link from v, and an incoming link to v, respectively. The page rank value is recursively defined and its computation can be viewed as a Markov process whose state are pages and the links between pages represent state transitions that are equal probable. The *PageRank*'s computation is given in the Equation (1) below.

$$r_{t+1}(v) = (1-d)\sum_{u\in V}\frac{r_t(u)}{|V|} + d\sum_{u\in in(v)}\frac{r_t(u)}{|out(u)|} \qquad (1)$$

The second part of Equation (1) represents when the surfer continues surfing (with probability d) to page v at time $t+1$ by clicking a hyperlink, at time t, from each page u that has an outgoing link to v (i.e., $u \in in(v)$). Because the chance of clicking each of such page u is equally likely, the probability of visiting v from each such u is $1/|out(u)|$, assuming that u has no more than one link to v. Alternatively, the surfer may stop using hyperlinks (with probability $1-d$) but visit page v at time $t+1$ by using page v's URL from any page that the user is at time t. The probability of visiting v via URL from any page is $1/|V|$ and thus, we obtain the first part of the equation.

To satisfy the constraint that the sum of page rank values over all of the considered web pages at any time must be one, a web page that has no outgoing hyperlink is assumed to have a link pointing to itself. To see this, consider summing $r_{t+1}(v)$, from Equation (1), over all v. Thus, the left side yields one by the constraint. On the right side of the equation, the first part gives $(1-d)$ and the second part becomes:

$$d\sum_{u\in V}\sum_{u\in in(v)}\frac{r_t(u)}{|out(u)|} = d\sum_{|out(u)|\neq 0} r_t(u). \qquad (2)$$

The second part as shown in Equation (2) needs an additional term, $d\sum_{|out(u)|=0} r_t(u)$, to produce a total of

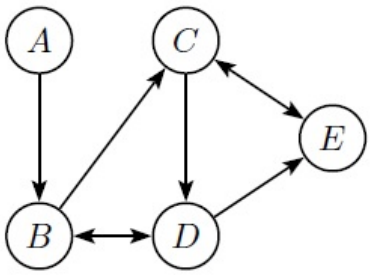

Figure 1: Hyperlinks of Web pages

d in the second part of the equation so that the right side of the equation becomes one as desired. Thus, the extra term is required for the constraint to hold. In fact, adding this extra term is acquired by assuming for a page with no outgoing link to have a self-loop link. With this assumption, the constraint is satisfied. As a result, Equation (1) can be simplified as the following.

$$r_{t+1}(v) = \frac{(1-d)}{|V|} + d \sum_{u \in in(v)} \frac{r_t(u)}{|out(u)|} \quad (3)$$

The above computation iterates over time to obtain a stable estimate of the probability distribution of each page's visit by random clicking behaviors. Thus, the computation terminates when there is no change in the probability distribution obtained.

We now give a small example to illustrate how the *PageRank* algorithm works. Figure 1 shows a collection of five web pages where a hyperlink between the pages is represented by a directed edge.

Based on the above web page structure, we can create a *stochastic matrix* (or *transition matrix*) [2] $A = (a_{ij})$, where a_{ij} represents a probability that a surfer makes a transition from page i to page j. Here we assume that each outgoing page from the same page has equal chance to be visited. Thus, below is a stochastic matrix corresponding to the hyperlinks of web pages in Figure 1. Here B has two outgoing links to page C and D. Thus, $a_{BC} = a_{BD} = 1/2$. Stochastic matrix is used for computing transitions in each iteration step in a Markov process.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Initially at $t = 0$, each of the five pages has the same ranking value of $1/5$ because the probability of visiting each page is equally likely and the sum of these probabilities must be one. Using a commonly used value of $d = 0.85$, at $t = 1$, the ranking values of pages A, B, C, D and E obtained are 0.03, 0.285, 0.115, 0.200, and 0.370, respectively. Table 1 summarizes the results up until iteration 15, where the ranking values no longer change.

Table 1: Ranking results of *PageRank* algorithm

t	$r_t(A)$	$r_t(B)$	$r_t(C)$	$r_t(D)$	$r_t(E)$
0	0.200	0.200	0.200	0.200	0.200
1	0.030	0.285	0.115	0.200	0.370
2	0.030	0.141	0.151	0.200	0.478
...	...	...	...	...	...
14	0.030	0.099	0.072	0.103	0.696
15	0.030	0.099	0.072	0.103	0.696

As shown in the last column of Table 1, page E has the highest value of 0.696 and therefore it has the highest chance to be visited comparing to other pages. In fact, the ranking order of these web pages, based on their visit likelihoods, is E, D, B, C, A, respectively.

4 Proposed Approach

The proposed analysis applies domain-specific knowledge to estimate the probability distribution of intrusion for each attack state in a given attack model. Specifically, it identifies, for each attack state, a relative chance of intrusion based on the degree of exploitability of its vulnerabilities. This section describes two core components of our analysis approach. Section 4.1 defines exploitability as heuristics to be applied in the *ExploitRank* algorithm, which is to be described in Section 4.2.

4.1 Exploitability

Our approach uses knowledge about existing network vulnerabilities that can be found in public databases. It is well recognized that some vulnerability may be exploited more easily than others. In fact, the complexity of the vulnerability can affect its exploitability, which in turn influences the chance of intrusion at different states of the network attack.

We define *exploitability*(v) to be a function that measures a degree of difficulty in exploiting vulnerability v with values ranging from zero to one (i.e., from the hardest to the easiest to exploit, or from the lowest to the highest vulnerability). The Common Vulnerability Scoring System (CVSS) [7] and severity factor provides a standard for computing exploitability of various publicly known vulnerabilities. Basic CVSS is based on vulnerability characteristics that are static over time and user environments. There are three basic metrics: *access vector*, *access complexity*, and *authentication*.

Access vector represents difficulty from the access location (e.g., local, adjacent network accessible, and network accessible or remote) required to exploit the vulnerability. The more remotely an attacker can exploit the vulnerability, the greater the exploitability value will be. *Ac-*

cess complexity indicates the level (i.e., low, medium and high) of effort required to exploit the vulnerability after an access to the target point is gained. For example, a buffer overflow in an Internet server has low complexity since the vulnerability can be exploited once an attacker gains access of the server. The lower the complexity is, the higher the exploitability will be. Finally, *authentication* is defined to measure the number of authentications required (e.g., multiple instances, single instance, or no instance) before network vulnerability can be exploited. Based on the United State National Institute of Standard and Technology [15], qualitative domain values of these three CVSS metrics are quantified to numeric values as the following:

$$\begin{aligned}
\textit{Access Vector} = {} & \text{case } \textit{Access Vector} \text{ of} \\
& \text{Local access: } 0.395 \\
& \text{Adjacent network accessible: } 0.646 \\
& \text{Network accessible: } 1.0 \\
\textit{Access Complexity} = {} & \text{case } \textit{Access Complexity} \text{ of} \\
& \text{High: } 0.35 \\
& \text{Medium: } 0.61 \\
& \text{Low: } 0.71 \\
\textit{Authentication} = {} & \text{case } \textit{Authentication} \text{ of} \\
& \text{Multiple instances : } 0.45 \\
& \text{Single instance : } 0.56 \\
& \text{No authentication: } 0.704
\end{aligned}$$

As an example, consider CVE-2006-5794, the Common Vulnerabilities and Exposures (CVE) in the sshd Privilege Separation Monitor in OpenSSH before Version 4.5. This vulnerability can be exploited by network accessible (i.e., remote) with no authentication, and the method to exploit this vulnerability is of low complexity. Therefore, *access vector*, *authentication*, and *access complexity* of this vulnerability is 1.0, 0.704 and 0.71, respectively. Thus, given a severity factor of 20, the exploitability of the CVE-2006-5794 vulnerability can be calculated as $20 \times AccessVector \times AccessComplexity \times Aunthentication =$ 9.9968. Note that the exploitability of the vulnerability in [3, 8] has a maximum value of 10. To obtain the exploitability value ranging to a maximum of one as defined in this paper, we divide the resulting value by 10. This gives the *exploitability*(CVE-2006-5794) = 0.99968, which indicates that CVE-2006-5794 has a high exploitability degree and thus, high vulnerability (i.e., easy to exploit/attack).

4.2 The *ExploitRank* Algorithm

ExploitRank algorithm estimates the probability distribution of intrusion for each attack state in a given attack model by applying Markov model similarly to how *PageRank* algorithm applies the model for ranking web pages. However, there is a subtle difference between web surfing behaviors and network attacking behaviors.

While a web surfer can randomly pick a web page to visit via its URL, an attacker does not have the same freedom. In fact, an attack model provides a constraint of how an attacker can traverse among attack states. For example, a surfer can arrive at any web page in one single step via URL but an attacker requires more than one step to advance to an attack state that the target system is completely shut down (e.g., by first gaining access privilege of the target system followed by a few steps to exploit the target's vulnerability). For this reason, we cannot employ the same recurrences of Equations (2) and (3) for ranking exploitability in attack states.

During an attack, an attacker has options to *continue* or *quit* attacking on a current path. We assume that if the attacker quits attacking on the current path (because it is too hard to lead to his goal), he will attempt on an alternative path by starting over from one of the set of initial states. Each of the initial state has equal chance to be a starting point of this new attempt. On the other hand, if he continues attacking, he will advance to each of the possible transition states with a *probability* based on how hard its vulnerabilities can be exploited (see more details later).

Based on a Markov model, we obtain Equation (4) and Equation (5) for computing the probability distribution of intrusion of a given attack model where we use the exploitability of vulnerabilities at each attack state along with the structure of the network. The computation gives a relative chance of intrusion for each attack state, or, roughly speaking, a ranking of the exploitability of attack states in the attack model. Thus, it provides a basis for the proposed *ExploitRank* algorithm.

For a given security model, let $r_t(v)$ be the probability of intrusion of attack state v at time t, I be a set of initial states and $h(u, v)$ be the exploitability of a vulnerability exploit from u to v as explained in Section 4.1. We define $r_t(v)$, a ranking score of v at time t, recursively as follows:

Case 1: v is not an initial state

$$r_{t+1}(v) = \sum_{u \in in(v)} r_t(u) \cdot e(u, v) \tag{4}$$

Case 2: v is an initial state

$$r_{t+1}(v) = \sum_{u \in in(v)} r_t(u) \cdot e(u, v) + \frac{1}{|I|} \left(\sum_{\substack{u \in V \\ w \in out(u)}} r_t(u) \cdot \bar{e}(u, w) + \sum_{\substack{u \in V \\ out(u) = \emptyset}} r_t(u) \right) \tag{5}$$

where $e(u, v) = \frac{h(u,v)}{|out(u)|}$ and $\bar{e}(u, v) = \frac{1-h(u,v)}{|out(u)|}$.

When v is not an initial state, the only way to attack v is by continuing exploiting a vulnerability from any state u to v, where u was attacked in a previous step, i.e., $u \in in(v)$. The likelihood of attack from each such u depends on the chance to intrude u, i.e., $r_t(u)$, and the likelihood

of each vulnerability exploit from u to v, i.e., $e(u,v)$. The latter depends on the chance of selecting the move from u to v out of all possible moves from u, i.e., $1/|out(u)|$ and the probability based on how hard it is to apply the exploit, i.e., $h(u,v)$, the exploitability of the vulnerability exploit from u to v. Thus, we obtain Equation (4).

On the other hand, when v is an initial state, an attacker can reach v in two ways. First, by continuing advancement from previously intruded state as derived in Equation (4), we can obtain the first part of Equation (5). The other way to reach v is based on our assumption that when the attacker gives up on the current attack path, he will start over from any initial state. Thus, the likelihood to intrude each initial state v depends on the chance to intrude any possible state u, i.e., $r_t(u)$, and the chance that the attacker will not continue exploiting a vulnerability from u to start over from v. If u has a vulnerability exploit to w, i.e., $w \in out(u)$ and u is not a terminal node, then the chance of u not to continue with this exploit and start over at v (out of all possible initial states) is $\frac{1}{|I|} \times \frac{1-h(u,w)}{out(u)}$. However, if u is a terminal node, u does not have an out-going exploit and by our assumption, the chance of the attacker not to exploit the vulnerability from u is certain. Thus, the chance of u starting over at v becomes $1/|I|$. This gives Equation (5).

Based on the recurrence equation above, we construct the *ExploitRank* algorithm as shown in Algorithm 1, where all variables are as defined. Assume that any given attack model can be represented as a graph, $G(V,E)$, where V and E represents a set of attack states and a set of vulnerability exploits, respectively. *ExploitRank* takes the attack model G with an exploitability degree corresponding for each possible connection between attack states as inputs. The probability distribution of network intrusion is computed recursively and iteratively using the stochastic matrix, defined in line 22, until the process reaches a stationary point in line 31. The algorithm produces a *relative* chance of intrusion at each attack state in a given model. This can be viewed as ranking among attack states in the order of the exploitability of their vulnerabilities. Next we evaluate the proposed approach by comparing the results obtained from the ranking between with and without the proposed heuristic (i.e., ours vs. Mehta et al.'s approach).

Note that Mehta et al. adopted the assumption used in *PageRank* algorithm where the attacker (or web surfer) may still pursue attacking (surfing) the terminal state u with probability d, the damping factor, leaving the chance of attacking v to be $1-d$. This difference with our approach is shown in Figure 2, where A is an initial node. In addition, Mehta et al.'s approach does not provide an explicit formulation of the Markov model as expressed in the equations here. More importantly, their approach does not take the degree of the difficulty in exploiting the vulnerability into consideration.

Algorithm 1 ExploitRank

```
1: Procedure ExploitRank G(V,E), h
2: I ← a set of initial states
3: A ← zero matrix of size |V| × |V|
4: t ← 0
5: for each v ∈ V do
6:    r_0(v) ← 1/|V|
7: end for
8: for each u ∈ V and v ∈ V do
9:    if u ∈ in(v) then
10:      e(u,v) ← h(u,v)/|out(u)|
11:   end if
12:   if v ∈ out(u) then
13:      ē(u,v) ← (1 − h(u,v))/|out(u)|
14:   else
15:      ē(u,v) ← 1 {u is a terminal node}
16:   end if
17:   if v ∉ I then
18:      w(u,v) ← e(u,v)
19:   else
20:      w(u,v) ← e(u,v) + ē(u,v)/|I|
21:   end if
22:   a(u,v) ← a(u,v) + w(u,v)
23: end for
24: repeat
25:   for each v ∈ V do
26:      for each v ∈ V do
27:         r_{t+1}(v) ← a(u,v) × r_t(u)
28:      end for
29:   end for
30:   t ← t + 1
31: until r_{t+1}(v) = r_t(v), ∀v ∈ V
32: return r_t^v, ∀v ∈ V
33: end procedure
```

5 Illustration

This section illustrates the proposed approach in details. Consider a simple but realistic network as shown in Figure 3, where there are two service hosts: *IP1* and *IP2*, and an attacker's workstation, *Attacker*, connecting to each of the servers via a central router. The network has a security requirement that "*no one can obtain a root privilege access to host IP2*".

Three types of vulnerabilities detected by a scanner (e.g., Nessus [3]): (1) CVE-2006-5794 (vulnerability in the sshd Privilege Separation Monitor in OpenSSH Version before 4.5) (2) CVE-2006-5051 (a signal handler race condition in OpenSSH Version before 4.4), and (3) CVE-2004-0148 (a configuration problem on the *restricted-gid* option). The first two can be exploited remotely to bypass the authentication process (thus, maintain a *user* access level in a victim host), and to obtain a denial of service (thus, gain a *root* access level in the victim host), respectively. Local users can exploit the last vulnerability to bypass access restrictions by changing their access permissions of a home directory via the *ftp*, which causes

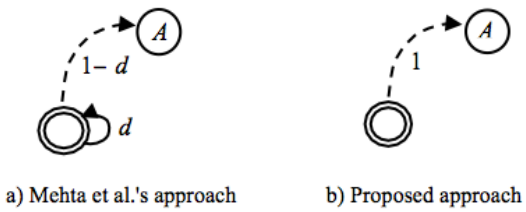

Figure 2: Comparing assumptions at a terminal node

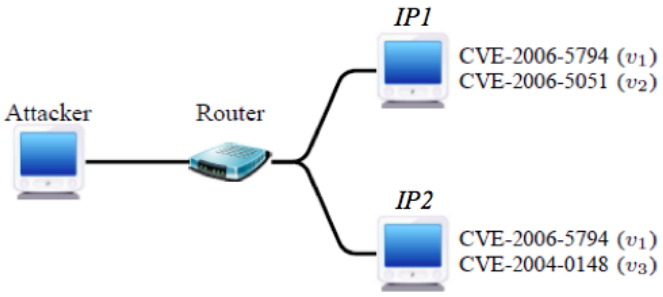

Figure 3: A simple scenario

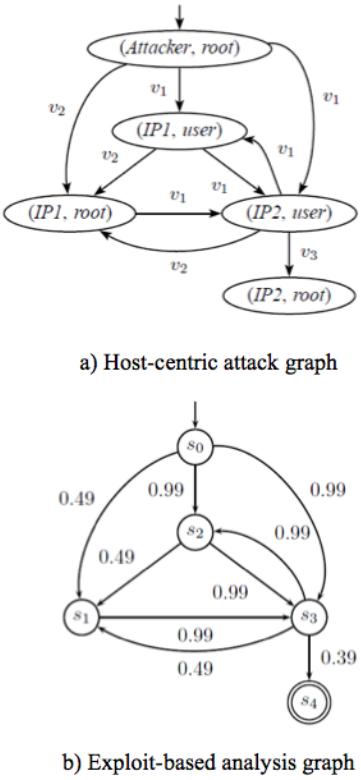

Figure 4: Annotated attack model for analysis

its service program, *wu-ftpd* to, instead, allow access of the root directory. We annotate each configuration of the network in Figure 3 with its corresponding vulnerabilities and their associated labels. For example, *IP2* has two vulnerabilities, namely CVE-2006-5794 (or v_1) and CVE-2004-0148 (or v_3). More details of these common standard vulnerabilities are described in [7, 20]. Although our approach can be applied to any form of a security model, in this study we use a host-centric attack graph model [8]. Suppose the goal of an attacker is to violate a security requirement. Based on the network configurations and the vulnerabilities shown in Figure 4, we can automatically generate a host-centric attack model as shown in Figure 4a) by employing a model-checking tool such as NuSMV [6] as illustrated in [8].

Each state is labeled by a tuple representing a host name and its access level obtained by an attacker. Thus, (*Attacker*, *root*) is an initial state since an attacker has a root access privilege on his own machine. The attacker's goal is to obtain a root access to *IP2* and thus, (*IP2*, *root*) represents a goal state. In Figure 4b), we rename the states (*Attacker*, *root*), (*IP1*, *root*), (*IP1*, *user*), (*IP2*, *user*) and (*IP2*, *root*) as s_0, s_1, s_2, s_3, and s_4, respectively.

Table 2 shows the exploitability computed for each of the relevant vulnerabilities obtained from publically known CVSS as described in previous section. Based on the heuristic values in Table 2, we obtained the corresponding attack graph for analysis as shown in Figure 4b) by replacing a state transition of each vulnerability exploit by a corresponding exploitability from Table 2.

The model obtained in Figure 4b) is used for computing a stochastic matrix $A = (a_{ij})$ in the *ExploitRank* algorithm to estimate a probability of transitions between any two attack states. The normalization is required so that the sum of the probabilities of all possible transitions from each state would be one. Note that for each applicable exploit of exploitability p, an attacker has two possible transitions: pursuing the exploit to the next state with likelihood p, or *not* pursuing the exploit and moving to an initial state to start over with probability $(1p)$. Therefore, the sum of probabilities of all possible transitions for *each exploit* is one. Thus, to obtain the normalized stochastic matrix, each probability of exploit from state s to state t is normalized by a total number of applicable exploits from s. For example, in Figure 4b), there are three applicable exploits to advance from state s_0, to states s_1, s_2 and s_3 with exploitability values 0.49, 0.99, and 0.99, respectively. Thus, the probability of applying the exploit to make a transition from s_0 to s_1 can be estimated from the normalized heuristic value of $0.49/3 = 0.16$. Similarly, the transition probabilities from s_0 to s_2 and s_3 can be estimated to 0.33 and 0.33, respectively. By the above

Table 2: Vulnerability and exploitability

Vulnerability Exploit	Vulnerability	Exploitability
v_1	CVE-2006-5794	0.99
v_2	CVE-2006-5051	0.49
v_3	CVE-2004-0148	0.39

argument, it is clear that the probability of a transition from s_0 (an initial state) to s_0 can be estimated from the normalized sum of probabilities of *not* pursuing all the three exploits from s_0 yielding a likelihood of entering s_0 to be $(0.51 + 0.01 + 0.01)/3 = 0.18$. These results are shown in the first row of the matrix below.

$$\begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccccc} s_0 & s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{pmatrix} 0.18 & 0.16 & 0.33 & 0.33 & 0 \\ 0.01 & 0 & 0 & 0.99 & 0 \\ 0.26 & 0.245 & 0 & 0.495 & 0 \\ 0.38 & 0.16 & 0.33 & 0 & 0.13 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Applying the above transition matrix to the *ExploitRank* algorithm, Table 3 shows the results of the intrusion probability distribution obtained by iteration. As shown in Table 3, a steady state is reached in iteration 16, where we obtain the intrusion likelihoods of each state.

Table 3: Computing intrusion likelihoods

t	$r_t(s_0)$	$r_t(s_1)$	$r_t(s_2)$	$r_t(s_3)$	$r_t(s_4)$
0	0.200	0.200	0.200	0.200	0.200
1	0.366	0.113	0.132	0.363	0.026
2	0.265	0.148	0.240	0.297	0.047
...	...	...	...	...	...
15	0.274	0.147	0.200	0.335	0.043
16	0.274	0.147	0.200	0.335	0.043

The results of ranking attack states in the host-centric attack model are shown in the first column of Table 4. We then apply Mehta et al.'s ranking approach that does not employ the exploitability heuristic and obtain the results in the second column of Table 4.

Table 4: Comparisons of ranking results

State	Our approach	Mehta et al.'s approach
s_0	0.274	0.150
s_1	0.147	0.145
s_2	0.200	0.102
s_3	0.335	0.209
s_4	0.043	0.394

As shown in Table 4, using exploitability heuristics (i.e., our approach) gives a ranking result of $\langle s_3, s_0, s_2, s_1, s_4\rangle$, whereas not using any heuristics (i.e., Mehta et al.'s approach) gives a ranking result of $\langle s_4, s_3, s_0, s_1, s_2\rangle$. Mehta el al.'s approach and ours suggest that s_4 and s_3, respectively has the highest (relative) likelihood of being attacked (i.e., most vulnerable). However, based on the structure of the attack model in Figure 4b), every path from s_0 to s_4 must pass thru s_3. Therefore, attacking s_4 is harder than s_3. Thus, the intrusion likelihood of s_3 should be higher than that of s_4. This is consistent with our ranking result but not Mehta et al.'s.

To further compare the two ranking results, both agree that the initial state s_0 is more vulnerable than s_1, and s_2 since the attacker has already intruded the initial state. However, the ranking order between s_1 and s_2 are in conflict. Consider an attack from the initial state. As shown in Figure 4b), to reach state s_1 (e.g., from s_0, s_2 or s_3) requires exploiting vulnerability v_2, whereas to reach state s_2 (e.g., from s_0 or s_3) requires exploiting vulnerability v_1. However, according to the CVSS standard, since $exploitability(v_1) = 0.99$ but $exploitability(v_2) = 0.49$, v_1 is more vulnerable than v_2. Therefore, intruding s_2 (via v_1) is easier than s_1 (via v_2). For example, from initial state s_0, reaching s_2 requires v_1 exploit compared to a v_2 exploit or a chain of v_1 and v_1 exploits to reach s_1. Therefore, s_2 should rank higher than s_1. This intuitive reasoning conforms to our ranking order but contradicts with the ranking order produced by Mehta et al.'s approach. In this particular example, using exploitability heuristic based on vulnerability appears to offer more sensible ranking results that obtained without the use of heuristic knowledge.

6 Experiments

This section describes two sets of experiments to assess the performance of our approach on relatively large attack models. The first aims to evaluate ranking results of relatively large attack model and the second focuses on computational cost for large-scale models.

6.1 Ranking Large Models

Figure 5 shows an attack graph of 66 nodes studied in [23]. The graph was generated with a security property that the intruder would never attain root privileges on the *Linux* host. As described in [23], Figure 5 shows the shaded nodes to signify areas that the intrusion detection system (IDS) alarm has been sounded. Thus, it is possible for the intruder to escape the detection by attacking a portion to the right of the graph that is not "covered" by the IDS. Figure 5 highlights an example of such an attack scenario (a path with solid square nodes), where each attack step identified by exploit number and name. Here the goal states, shown by double circled nodes, are when the intruder violates the security property (i.e., he successfully gains a root privilege on the *Linux* host). Here the attack model has 16 goal states and one single initial state.

It is clear that making a decision on which vulnerability and security flaws to fix first in order to effectively protect the network can be a complex task, especially when dealing with a large attack model. In this context, we ran the ExploitRank algorithm to rank nodes in the attack graph in order of their intrusion likelihoods. To evaluate the ranking results, since there is no known solution, we compare our results with those obtained from Metha

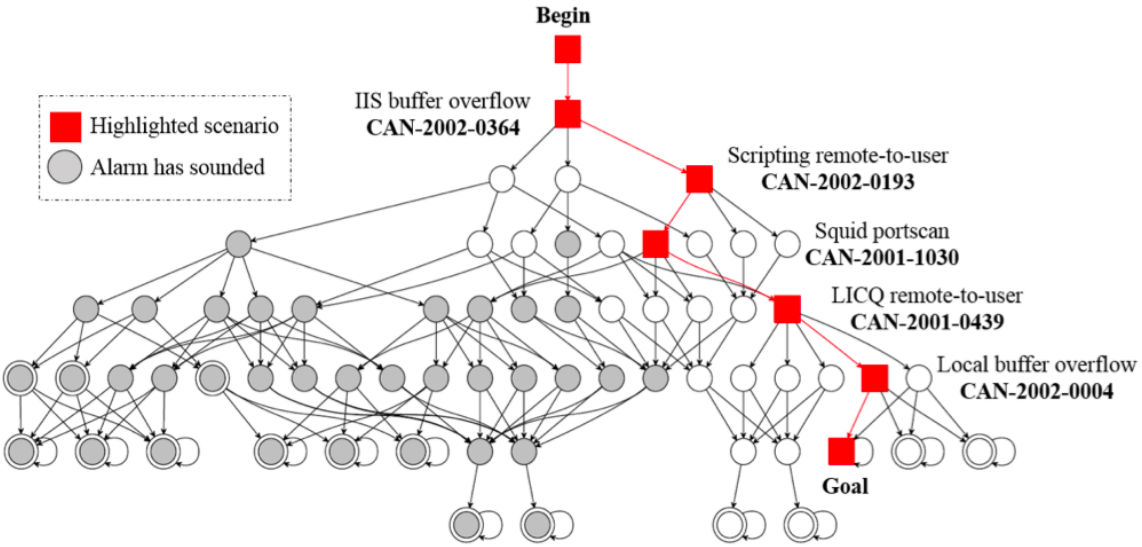

Figure 5: Attack model does not have a full coverage from an IDS [23]

et al.'s approach. Using the attack model, excluding the root, of Figure 5, Figures 6 and 7 show results obtained by Mehta et al.'s approach and ours, respectively.

Recall that Mehta et al.'s approach assumes that every child node is equally likely to be attacked from the parent node, whereas ours differentiates each possibility based on the exploitability of a corresponding vulnerability to be exploited. In this experiment, we assign three exploitability values: 0.1, 0.3, and 0.7 as shown by a dash, solid, and thick solid line to represent the exploit that is hard, somewhat hard, and easy to perform, respectively.

Nodes with equal resulting likelihoods (ranking scores) are labeled with the same rank. As shown in Figure 6, ranking results of Mehta et al.'s approach are the same for nodes that are on the same level with the same degrees of exposure (i.e., incoming arrows), e.g., ranks 6, 14, 43, 53, 61, and so on. The reason for the former (nodes of the same level are of the same rank) is because of the use of Markov property where intrusion likelihoods of states in a current level are impacted by only intrusion likelihoods of states in a previous level, whereas the latter (nodes of the same degree of exposure) is by the assumption on equal likelihoods of attacks from a parent to every child. However, in practice, this is highly unlikely the case, as we know that intrusion likelihoods depend on types of exploits and their exploitability degrees.

Figure 7 shows the ranking results obtained by our approach using a random exploitability assignment as described earlier. Glancing at the results, we no longer obtain the same regularity as observed by Mehta et al.'s approach. It is not necessary that nodes of the same level and the same parent would have the same rank. Both approaches give the root to have the highest rank since it is the easiest to intrude (since the intruder is already there). However, Mehta et al.'s results rank the four goal states at the bottom to be next easiest to intrude. This could be due to the persistence (damping) factor and the assumption that attackers who intrude nodes with no outgoing link (terminal nodes) will persist on their attempt to attack with a probability of the damping factor before giving up to start over (as shown in Figure 2). To compare results of the two approaches in more details, consider three representative scenarios as summarized in Table 5. The number entries, as marked in Figures 6 and 7, represent ranking labels of the nodes, from left to right, in the corresponding depth level of each scenario.

Table 5: Comparison of three ranking scenarios

Scenario	Mehta et al.'s approach	ExploitRank
Level 1	6, 6, 6	2, 1, 2
Level 4	53, 53	44, 56
Bottom level	4, 4	26, 32

Scenario 1 compares three nodes in Level 1 from the root. Mehta et al.'s approach reports that the three nodes are of the same rank. This is clearly wrong. In Figure 7, node 1 has higher rank (more likely to be intruded) than the other two nodes 2. This is because:

1) all the three nodes are intruded by exploiting vulnerability from the same node 0 (root), and

2) the second exploit has the highest exploitability degree (i.e., easiest to intrude), therefore its destination node 1 should have the highest rank.

Since the other two have the same exploitability that is somewhat hard to exploit, they both must be of lower rank than node 1, hence nodes 2.

Scenario 2 compares relative ranks of two nodes in Level 4. In Figure 7, consider nodes 44 and 56. Both have two incoming exploits from the same parent nodes

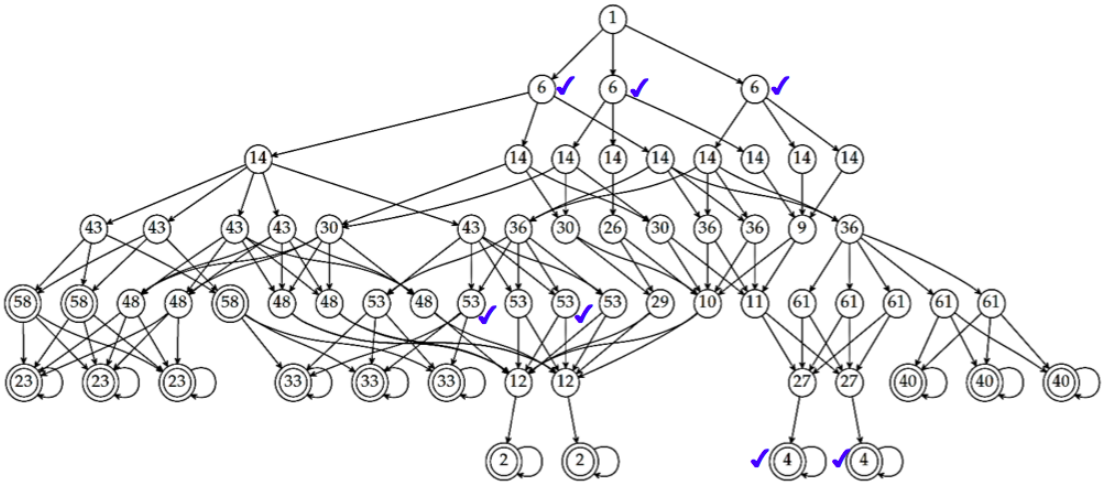

Figure 6: Ranking results by Mehta et al.'s approach

at Level 3. However, both of the incoming exploits to node 44 have exploitability of 0.3 (somewhat hard to exploit), while both of the incoming exploits to node 56 are of exploitability 0.1 (hard to exploit). Therefore, node 44 has a higher order of intrusion likelihood than node 56 as ranked by our approach. However, Mehta et al.'s approach results in equal rank for nodes 53.

Finally, Scenario 3 compares relative ranks of two terminal nodes at the bottom level of the tree. In Figure 7, consider nodes 26 and 32. Since each can only be intruded by exploiting from its parent who has the same intrusion likelihood (i.e., the same rank of 22), the destination of a dark solid link (high exploitability of 0.7), node 26 must be easier to intrude. Thus, node 26 has a higher intrusion likelihood than node 32 as obtained by our approach. Once again, Mehta et al.'s approach does not distinguish such likelihoods. Based on the three case scenarios, *ExploitRank* algorithm outperforms Mehta et al.'s approach. Although we do not compare all possible relative ranking results, we anticipate that our approach would rank correctly based on the logics of our recurrence formulae.

In addition, we have also experimented with a modified Metha et al.'s approach where exploitability is used as a heuristic for estimating prior probability for an exploit. The ranking results of our approach still outperform those of the modified Mehta et al.'s approach (not shown here). For example, consider a relative ranking of 26 and 32 in Figure 7. The modified Metha et al.'s approach that uses exploitability as heuristic gives the same ranking score of 3 for these two nodes. This is clearly wrong since both nodes have parents of the same ranking order; each can be reached by a single exploit where one has a higher exploitability degree than the other. Therefore, the resulting ranks of these two nodes should be different. The main distinction that contributes to this significant difference is due to the fact that Mehta et al.'s approach assumes that an attacker behaves like a surfer when he reaches a terminal node in that there is a chance that we would continue penetrating the node intrusion (or surfing the site with the likelihood of a damping factor), while our approach does not. As a result, Mehta et al.'s assumption increases the intrusion likelihoods of terminal nodes and lessens the impact of the degree of exploitability of the exploits to reach these nodes. Unlike Mehta et al.'s approach, we assume that the attacker starts over when he reaches the terminal point of the attack path.

6.2 Performance on Large-scale Attack Models

This section presents experiments to see if the proposed approach can be computed efficiently enough to cope with large-scale attack models. Our *ExploitRank* algorithm was implemented using NodeJS language on Ubuntu Linux machine with an Intel Core i5 CPU of 3.20 GHz and 2 GB memory.

Table 6 shows a sample of running times of the implemented algorithm with various sizes (number of nodes and edges) of attack graphs. The edges were randomly generated. As shown in Table 6, while the size of the graph roughly grows with a constant rate of four, the running times grow approximately at the rates of 4, 5, 2 and 3.5, respectively, making the ratios between the size growth and the running time growth close to one (except for the case of 1024 nodes).

Table 6: Running times of our ranking approach

#Nodes	#Edges	Graph Size	Running time (sec)
128	8,192	8,320	0.5
256	32,768	33,024	2.1
512	131,072	131,584	10.7
1,024	524,288	525,312	20.2
2,048	2,097,152	2,099,200	71.5

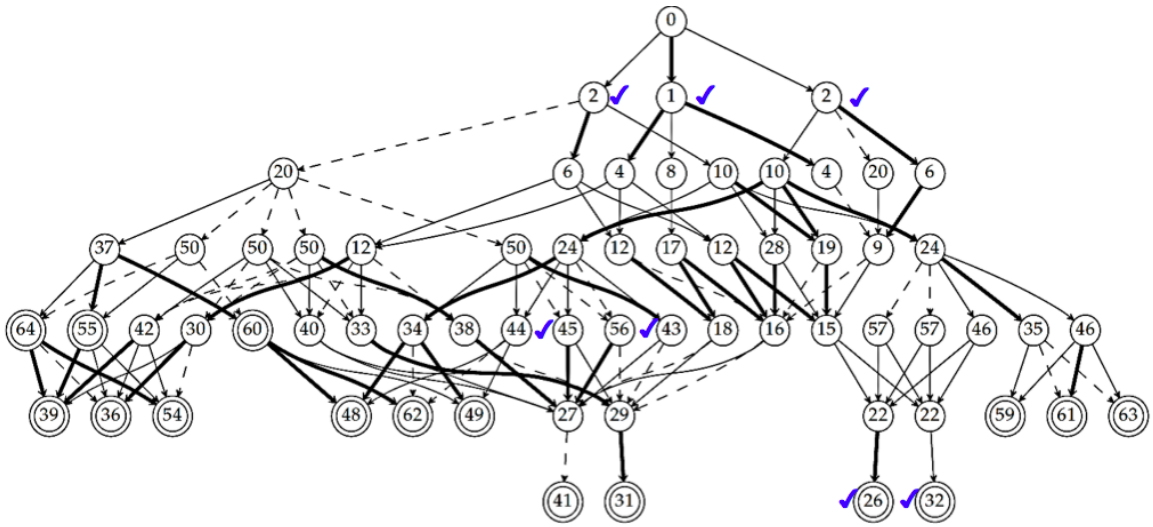

Figure 7: Ranking results by *ExploitRank* algorithm

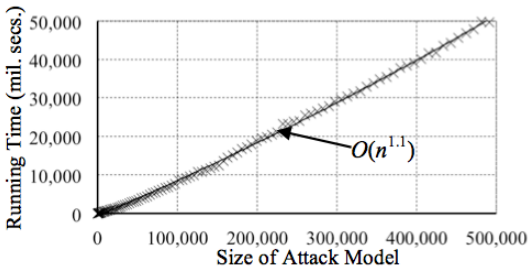

Figure 8: Performance of the ranking algorithm

To further evaluate the performance of our approach, we ran 500 runs of experiments with various sizes of attack models ranging from 60 to 501,000. Figure 8 shows the resulting runtimes in milliseconds. The running time obtained fits to an approximate linear equation: $0.0185n^{1.13}$, where n is the attack graph size. Just like PageRank algorithm that can handle ranking of a huge number of web pages, based on similar concepts of Markov Model, our ExploitRank algorithm can scale with linear time in size of the graph. The advantage of ranking algorithm is that one can give the number of top k nodes to be ranked. This is useful when resources are limited.

7 Conclusions

We present an automated approach to attack model analysis that allows quantitative ranking of network nodes by their intrusion likelihoods. What sets our approach apart from the rest is our use of domain-specific knowledge that can be obtained from public databases or derived in a principled way from the structure of the network. The approach is adapted from a Markov Model-based ranking algorithm that is well-established tractable computational model used for intractable problems (e.g., ranking webpages).

This paper differs from our previous work [11] in that the previous work extends Mehta et al.'s approach to using the exploitability concept. However, as we have illustrated in this paper in the example in Section 6.1 that the basic assumption adapted by Mehta et al.'s approach is not appropriate for use in intrusion analysis. Future work includes additional evaluations of the proposed approach by investigating a large network in real-world applications.

References

[1] P. Ammann, J. Pamula, J. Street, and R. Ritchey, "A host-based approach to network attack chaining analysis," in *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)*, pp. 72-84, 2005.

[2] S. R. Asumssen, "Markov Chains", *Applied Probability and Queues: Stochastic Modelling and Applied Probability*, vol.51, pp. 3-38, 2003.

[3] J. Beale, R. Deraison, H. Meer, R. Temmingh, and C. V. D. Walt, *Nessus Network Auditing*, Syngress Publishing, 2004.

[4] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107-117, 1998.

[5] CIAC, "Computer incident advisory capability (CIAC)," 2009. (http://www.ciac.org/ciac/index.html)

[6] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Proceedings of the 14th*

International Conference on Computer Aided Verification (CAV'02), pp. 359-364, 2002.

[7] CVSS, "Common vulnerability scoring system (cvss)," 2009. (http://www.first.org/cvss/cvss-guide.html)

[8] R. Hewett and P. Kijsanayothin, "Host-centric model checking for network vulnerability analysis," in *Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC'08)*, pp. 225-234, 2008.

[9] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Computer Security Applications Conference*, pp. 121-130, 2006.

[10] S. Jha, O. Sheyner, and J. M. Wing, "Two formal analys s of attack graphs," in *Proceedings of the 15th IEEE workshop on Computer Security Foundations (CSFW'02)*, PP. 49, 20029.

[11] P. Kijsanayothin and R. Hewett, "Exploit-based analysis of attack models", in *Proceeding of the 12th International Symposium on Network Computing and Applications (NCA'13)*, pp. 183-186, 2013.

[12] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604-632, Sep. 1999.

[13] K. Lye and J. M. Wing, "Game strategies in network security," *International Journal of Information Security*, vol. 4, no. 1-2, pp. 71-86, 2005.

[14] V. Mehta, C. Bartzis, H. Zhu, E. M. Clarke, and J. M. Wing, "Ranking attack graphs," in *Recent Advances in Intrusion Detection*, LNCS 4219, pp. 127-144, 2006.

[15] NIST, "The united state national institute of standard and technology," 2009. (http://www.nist.gov/index.html)

[16] S. Noel and S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," in *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC'04)*, pp. 109-118, 2004.

[17] S. Noel and S. Jajodia, "Understanding complex network attack graphs through clustered adjacency matrices," in *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)*, pp. 160-169, 2005.

[18] NVD, "National vulnerability database (NVD)," National Institute of Science and Technology (NIST), 2009. (http://nvd.nist.gov/nvd.cfm)

[19] R. E. Sawilla and X. Ou, "Identifying critical attack assets in dependency attack graphs," in *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, pp. 18-34, 2008.

[20] K. Scarfone and P. Mell, "An analysis of cvss Version 2 vulnerability scoring," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09)*, Washington, pp. 516-525, 2009.

[21] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP'02)*, pp. 273, 2002.

[22] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," *DARP A Information Survivability Conference and Exposition*, vol. 2, pp. 1307, 2001.

[23] J. M. Wing, "Scenario graphs applied to security", *Verification of Infinite-State Systems with Applications to Security,* IOS Press, pp. 229-233, 2005.

Rattikorn Hewett is a professor and Chair of the department of Computer Science, Texas Tech University. She has a Ph.d. in Computer Science from Iowa State University, an M. Eng. Sc. in Computer Science from the University of New South Wales, and a B.A. (Hons) in Pure Mathematics and Statistics from Flinders University, Australia. She was a post-doctoral fellow at the Knowledge System Laboratory, Stanford University. Her research in Artificial Intelligence (AI) includes intelligent control, and machine learning with applications in data mining, bioinformatics and software engineering. Her recent work involves applied AI research to automated science, network security and Internet security. She has published extensively and has served on numerous international conference and work-shop program committees.

Phongphun Kijsanayothin received his B. Eng. (Computer Engineering) from the King Monkuts Institute of Technology Ladkrabang, in 1999, and his M.Eng. (Computer Engineering) from Kasetsart University, in 2003, and his Ph.D. in Computer Science from Texas Tech University, in 2010. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering, Naresuan University. His research interests include network security, cyber security and software testing in the context of object-oriented development.