An Adversary Aware and Intrusion Detection Aware Attack Model Ranking Scheme*

Liang Lu, Rei Safavi-Naini, Jeffrey Horton, and Willy Susilo

Centre for Computer and Information Security Research
School of Computer Science & Software Engineering
University of Wollongong, Australia
{ll97,rei,jeffh,wsusilo}@uow.edu.au

Abstract. A successful computer system intrusion is often resulted from an attacker combining exploits of individual vulnerability. This can be modelled by attack models and attack graphs to provide a global view on system security against attacker's goal. However, as the size and complexity of attack models and attack graphs usually greatly exceeds human ability to visualize, understand and analyze, a scheme is required to identify important portions of attack models and attack graphs. Mehta *et al.* proposed to rank states of an attack model by the probability of an adversary reaching a state by a sequence of exploiting individual vulnerabilities in a previous scheme. Important portions can hence be identified by ranks of states. However, Mehta *et al.*'s ranking scheme is based on the PageRank algorithm which models a web surfing scenario, but has not considered much on the dissimilarity between web surfing scenarios and computer system intrusion scenarios. In this paper, we extend Mehta *et al.*'s scheme by taking into consideration dissimilarity between web surfing scenarios and computer system intrusion scenarios. We experiment with the same network model used in Mehta *et al.*'s scheme and have the results compared. The experiments yielded promising results that demonstrated consistent ranks amongst varying parameters modelled by our ranking scheme.

1 Introduction

A large computer system often consists of multiple platforms, runs different software packages and has complex connections to other systems. Despite the best efforts by system designers and architects, there will still exist vulnerabilities resulting from bugs or design flaws allowing an adversary (attacker) to gain a level of access to systems or information not desired by the system owners. The act of taking advantage of an individual vulnerability is referred to as an "atomic attack" or an "exploit". A vulnerability is often exploited by a piece of software, a chunk of data, or sequence of commands, resulting in unintended or unanticipated behavior of the system such as gaining control of a computer system or

* This work is partially supported by Cooperative Research Center - Smart Internet Technology (CRC-SIT), Australia.

© Springer-Verlag Berlin Heidelberg 2007

allowing privilege escalation. Although an exploit may have only insignificant impact on the system by itself, an adversary may be able to construct a system *intrusion* that combines several atomic attacks, each taking the adversary from one system state to another, until he reaches the final goal. Therefore, to evaluate the security level of a large computer system, an administrator must not only take into account the effects of exploiting each individual vulnerability, but also consider global intrusion scenario where an adversary combines several exploits to compromise the system.

There has been considerable amount of work on modelling multi-stage attacks by combination of individual vulnerabilities [7] [12] [5]. Recently, Sheyner *et al.* proposed using attack models and attack graphs to provide a global view of system security against exploiting the combination of vulnerabilities [16] [7]. An attack model is a graph that consists of a set of nodes and edges, where each node represents a reachable system state and each edge represents an atomic attack that takes the system from one state to another. An attack graph is a subgraph of the attack model and contains only nodes in the paths that eventually reach a state where the system is considered compromised. With attack models and graphs, a global view on multi-stage attacks by combination of individual vulnerabilities can be obtained by administrators to assist in the implementation of effective security measures.

However, as the size and complexity of attack models/graphs usually greatly exceeds human ability to visualize, understand and analyze, a scheme is required to identify important portions of attack models/graphs. An effective method is to rank states in attack models/graphs based on factors like the probability of an intruder reaching the state. Important portions of attack models/graphs can hence be identified by ranks of their states.

Mehta *et al.* [16] propose to rank states of an attack model by the probability of an adversary reaching a state by a sequence of atomic attacks. The ranking algorithm is based on the PageRank algorithm used by Google to measure importance of web pages on the World Wide Web. Given a system to be analyzed, first an attack model formally describing the system is constructed. Then the ranking algorithm is applied to rank states of the obtained attack model. Meanwhile, an attack graph is generated using the attack graph generation tool [10], and is "projected" on the ranked attack model to obtain a ranked attack graph. The ranked states of an attack graph provide various security analysis for the system, such as measuring security of the system, evaluating effectiveness of counter-measures, and identifying important portion for visual analysis of the system.

Despite the similarity between ranking web pages and ranking system states, differences not considered by Mehta's scheme exist between the two scenarios. The World Wide Web model adopted by PageRank assumes that a random surfer has universally equal probabilities of following one of the links in a current page to the next page, and correspondingly Mehta's ranking scheme assumes that an attacker has equal probability of remaining undetected at all states of an attack model. However, the likelihood of an attacker remaining undetected at a state so as to exploit a vulnerability that takes the system to another state could be considered to be influenced by the number of steps required to reach the state from the starting position. Consider a scenario where a network has implemented some sort of defense-in-depth as an example, the more steps an attack has taken, the more likely that he is discovered. Therefore, we assume the probability of an attacker remaining undetected at a state decreases with number of steps required to reach that state. The decreasing rate is not universal amongst all computer systems but determined by each system's intrusion detection ability, which affects state transitions in attack models and should be considered when ranking system states.

Moreover, the random transition model adopted by PageRank assumed that a WWW surfer navigates to the next page by selecting one of the available succeeding pages at random with equal probabilities. This assumption fits well with intrusions that use a brute force probe-scan approach. However, an adversary may exploit vulnerabilities based on metrics such as cost, age, evaluation on probability of success and being detected. In this case, vulnerabilities are not selected at random and different vulnerabilities have different probabilities of being exploited. The behavior of an adversary selectively exploiting vulnerabilities has considerable effect on his chance to reach the final goal, and therefore should be considered when ranking system states of attack models and graphs.

1.1 Our Contribution

In this paper, we propose a ranking scheme that addresses problems stated above. The proposed ranking scheme is adjusted from Mehta *et al.*'s scheme, but has the advantage of modelling variation in intrusion detection abilities amongst computer systems, and non-uniform distribution in probability that each vulnerability is exploited. First, in addition to modelling vulnerabilities in a system that could be exploited to have system states changed, our ranking scheme also models intrusion detection ability of computer systems defined as the system's effort to detect and prevent such state transitions by intruders exploiting vulnerabilities. Secondly, we provide an instantiation of the biasing idea in [1] by modelling adversaries' behavior in exploiting vulnerabilities probabilistically based on certain metrics as stated above but not by brute-force probing. With the proposed ranking scheme, evaluation on system intrusion detection ability or adversaries' ability in relation to probabilistically exploit vulnerabilities, when available from for example empirical data or log statistics, can be used to obtain more accurate ranks of computer system states modelled by attack models and attack graphs. The proposed scheme can also be applied to other areas such as network research or system design, e.g. determining minimum system intrusion detection strength required to protect against best effort by an adversary.

To evaluate the effectiveness of the proposed ranking scheme, we implemented a prototype of the scheme in Java. We experiment with the network example used by Mehta *et al.* [16] and have the results compared with their scheme. The experiments yielded promising results that demonstrated consistent ranks amongst varying parameters modelled by the proposed ranking scheme.

1.2 Related Work

Techniques for quantitative security measurement have been a strong focus in the research community [9] [11] [4] [5]. Dacier *et al.* [9] model a computer system as a *Privilege Graphs* exhibiting security vulnerabilities and convert it into a Markov chain corresponding to all possible successful attack scenarios. The Markov chain is then used to compute MTTF (mean time to failure) of the system, used as the quantitative measure of the security level of a system. Time and effort required by each type of attack is estimated from empirical and statistical data. Phillips *et al.* [5] present a framework for evaluating the most likely attack paths in the attack graph generated by an ad hoc algorithm. The framework requires attacker profiles and attack templates in order to compute the likelihood of each type of attack. Madan *et al* [4] proposed an approach to quantifying various security related attributes of a computer system such as system availability, MTTF, and probabilities of system failure. Quantification of security related attributes is by solving the Semi-Markov Process (SMP) model describing state transitions in the system. However, the proposed approach requires availability of a wide range of ad hoc model parameters, restricting the approach feasible only for systems of a small scale. Another related work presented in [11] provides a quantitative analysis of attacker behavior based on empirical data collected from intrusion experiments.

1.3 Paper Organisation

The rest of the paper is organized as follows. Section 2 provides a brief review on relevant background knowledge. The proposed ranking scheme is presented in Section 3. Section 4 provides implementation details and experimental results demonstrating effectiveness of the proposed scheme, and Section 5 concludes the paper.

2 Background and Preliminaries

2.1 Attack Models and Attack Graph

Sheyner *et al.* first formally defined the concept of *Attack Model* and *Attack Graph* [10]. An *Attack Model* is a formal description of security related attributes of the attacker, the defender and the modelled system using graph representation. Nodes represent the states of the system, such as the attacker's privilege level on individual system components. Transitions correspond to actions taken by the attacker which lead to a change in the state of the system. The starting state of the model denotes the state of the system where no damage has occurred and the attacker is looking for an entry point to enter the system. As an example, if we consider the case of a computer network attack model, a state represents the state of the attacker, the running services, access privileges, network connectivity and trust relations. The transitions correspond to actions of the attacker such as exploiting vulnerabilities to obtain elevated privileges on the computer system. Formally,

Definition 1. [10] *Let AP be a set of atomic propositions. An Attack Model is a finite automaton $M = (S, \tau, s_0, l)$, where S is a set of states in relation to a computer system, $\tau \subseteq S \times S$ is the transition relation, $s_0 \in S$ is the initial state, and $l : S \rightarrow 2^{AP}$ is a labelling of states with the set of propositions true in that state.*

The negation of an attacker's goal in relation to an attack model can be used as *security properties* that the system must satisfy in a secure state. An example of a security property in computer networks would be "the intruder cannot gain root access on the database server". States in an attack model where the *security properties* are not satisfied are called *error states*. Given an attack model and the attacker's goal, an *Attack Graph* is a subgraph of the attack model which contains only paths leading to one of the error states, and states on such paths. Formally,

Definition 2. [10] *Let AP be a set of atomic propositions. An Attack Graph is a finite automaton $G = (S, \tau, s_0, S_s, l)$, where S is a set of states in relation to a computer system, $\tau \subseteq S \times S$ is the transition relation, $s_0 \in S$ is the initial state, $S_s \subseteq S$ is the set of error states in relation to the security properties specified for the system, and $l : S \rightarrow 2^{AP}$ is a labelling of states with the set of propositions true in that state.*

Given an attack model and the associated security properties, model checking techniques can be used to generate attack graphs automatically [14].

2.2 Web Graph and PageRank

Web Graph and Notations. For a web model consisting of N nodes (pages), notations used in the our discussion are defined in Table 1. Consider the web graph shown in Figure 1 as an example. Node 1 has three out links (node 2, 3 and 4) and hence $h_1 = 3$. Node 4 is pointed to by two nodes (node 1 and 3) and hence $pa[4] = \{\text{node 1, node 3}\}$. Equation 1 represents the transition matrix W in relation to the web graph. It is noticeable that not all nodes have out links, and we refer to these nodes as *dangling nodes*. When a web surfer reaches at a web page that has no out links, he is often assumed to select a random page to continue surfing [13] [2]. To model this, a *dangling node* in a web graph is typically assumed to be pointing to all other nodes with equal probabilities.

$$\mathbf{W} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ \frac{1}{3}&0&0&0&0&0&0&0&0 \\ \frac{1}{3}&0&0&0&0&0&0&0&0 \\ \frac{1}{3}&0&\frac{1}{3}&0&0&0&0&0&0 \\ 0&1&0&\frac{1}{2}&0&0&0&0&0 \\ 0&0&0&0&\frac{1}{2}&0&0&0&0 \\ 0&0&0&0&\frac{1}{2}&0&1&0&0 \\ 0&0&\frac{1}{3}&\frac{1}{2}&0&0&0&0&0 \\ 0&0&\frac{1}{3}&0&0&0&0&0&0 \end{pmatrix} \quad (1)$$

Table 1. Web Model Notations

Notation	Meaning
h_j	Number of nodes (pages) pointed to by node (page) j
$pa[j]$	Set of nodes (pages) pointing to node (page) j.
d	Probability that a random surfer continues surfing by navigating to one of the pages linked by the current page, usually referred to as *damping factor*. Correspondingly, $1-d$ represents the probability that a random surfer continues surfing and navigates to a random page
W	$W=\{w_{i,j}\}$ is a transition matrix such that $w_{i,j}=\frac{1}{h_j}$ if there is a link from node j to node i, otherwise $w_{i,j}=0$. An important property of W is that $\forall j$, $\sum_{i=1}^{N} w_{i,j}=1$.
Π_N	[1, ..., 1]', i.e. transpose of the N-dimension unit vector

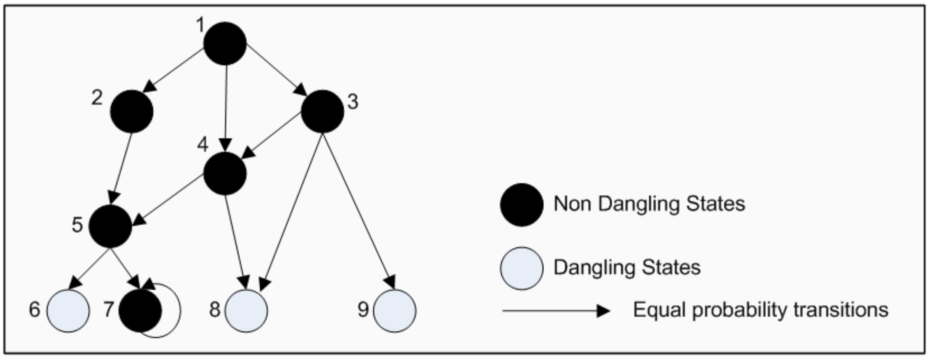

Fig. 1. An example of web graph

PageRank Algorithm. PageRank [13] is the algorithm used by Google to determine the relative importance of web pages on the World Wide Web. PageRank is based on the behavior model of a random surfer in a web graph. It assumes that a random surfer starts at a random page and keeps clicking on links and eventually gets bored and starts on another random page. To capture the notion that a random surfer might get bored and restart from another random page, a *damping factor* d is introduced, where $0 < d < 1$. The transition probability from a state is divided into two parts: d and 1 - d. The d mass is divided equally among the state's successors. Random transitions are added from that state to all other states with the residual probability 1 - d equally divided amongst them, modelling that if a random surfer arrives at a dangling page where no links are available, he is assumed to pick another page at random and continue surfing from that page. The computed rank of a page is the probability of a random surfer reaching that page. That is, consider a web graph with N pages linked to each other by hyperlinks, the PageRank x_p of page (node) p is defined as the probability of the random surfer reaching p, formally

$$x_p = d \sum_{q \in pa[p]} \frac{x_q}{h_q} + \frac{1-d}{N} \tag{2}$$

When stacking all the x_p into a vector $\mathbf{x}$, it can be represented as

$$\mathbf{x} = dW\mathbf{x} + \frac{1}{N}(1-d)\Pi_N \tag{3}$$

Using iterative expression, Equation 3 can be represented as

$$\mathbf{x}(t) = dW\mathbf{x}(t-1) + \frac{1}{N}(1-d)\Pi_N \tag{4}$$

The computation of PageRank can be considered a Markov Process, as can be seen from Equation 4. It has been proved that after multiple iterations, Equation 4 will reach a stationary state where each x_p represents the probability of the random surfer reaching page p [8].

2.3 Mehta et al's Ranking Scheme

Given an attack model $M = (S, \tau, s_0, l)$, the transition probability from each state is divided into d and 1-d, modelling respectively that an attacker is discovered and isolated from the system, or that the attacker remains undetected and proceeds to the next state with his intrusion. Similar to PageRank, the rank of a state in an attack model is defined to be the probability of the system being taken to that state by a sequence of exploits. The ranks of all states are computed using the method for computing PageRank described in Section 2.2. Breadth first search starting from the initial system state s_0 is then performed for each atomic attack in τ to construct the transition matrix W. The only adjustment from PageRank, where a transition from each state pointing to all other states with probability 1-d equally divided amongst all other states, is that a transition from each state pointing back to the initial state with probability 1-d is added to model the situation where an attacker is discovered and has to restart the intrusion from the initial state.

3 Modelling Adversary and Intrusion Detection Capability in Ranking Attack Models

Recall the discussion in Section 1. Unlike the web graph model adopted by PageRank where the probability that a random surfer follows a link to the next page is state independent, the likelihood of an attacker remaining undetected at a state so as to exploit a vulnerability that takes the system to another state could be considered to be influenced by the number of steps required to reach the state from the starting position. Therefore we assume the probability of an attacker remaining undetected at a state decreases with number of steps required to reach that state. The decreasing rate is not universal amongst all computer systems but system specific as determined by each system's intrusion detection ability. Another important dissimilarity between a web surfing scenario and a system intrusion scenario is that exploits taking a computer system from one state to another may be "selected" not at random but based on the adversary's evaluation on metrics such as cost, effort, probability of success and being detected, whereas links taking a web surfer to the next page is always selected at random with equal probabilities. These factors affect an adversary's chance to reach his final goal, and therefore should be considered when ranking states of attack models and graphs.

In this section, we propose an adversary aware and intrusion detection aware ranking scheme that addresses problems stated above. Being adversary aware, the proposed scheme considers how an adversary selectively exploiting vulnerabilities affect his chance to compromise the system. Being intrusion detection aware, the proposed scheme considers system intrusion detection ability and how it affects an adversary's chance to reach his final goal.

3.1 Web Graph Adjustment

The transition model of web graph needs to be adjusted to provide a more accurate simulation of computer system state transitions in relation to system intrusion scenario. As in Mehta *et al.*'s ranking scheme, we add a transition from each state pointing back to the initial state with probability 1-d, modelling the situation where an intrusion is detected and needs to be restarted from initial state. Furthermore, our ranking scheme differs from Mehta *et al.*'s scheme in that

1. We assume that the probability of an attacker remaining undetected at a state decreases with the number of steps required to reach that state, which in an attack model can be represented as length of the intrusion path to reach that state. The decreasing rate is determined by each system's intrusion detection ability. In general, well-protected systems such as systems implementing "defense-in-depth" have better ability to detect intrusions at earlier stages and can be simulated with greater decreasing rates. As it is difficult to predict the actual intrusion path, we simplify the situation by assuming that at each state s_j the probability of an attacker remaining undetected decreases at a rate proportional to $l(s_0, s_j)$, length of the shortest path between state s_j and initial state s_0. That is, the probability of an attacker remaining undetected at state s_j exponentially decays with length of the shortest path from s_0 to s_j. Consequently, transition probability from each state s_j is divided into d_j and 1-d_j representing respectively the situation where an attacker remains undetected and is able to take the system to another state, or where the attacker is discovered and has to restart the intrusion. d_j is the value of d exponentially decaying with $l(s_0, s_j)$ where d is the usual *damping factor*.

2. In a system intrusion scenario, it is more likely that an adversary has the ability to prioritize and exploit "promising" vulnerabilities based on his past experience and knowledge, other than probing the target network with brute-force attack. This is modelled in our ranking scheme by assigning a separate probability to each type of exploit. We divide each d_j among state s_j's successors according to the probability that each type of exploit is selected to take s_j to one of its successors. The probability distribution can be obtained from empirical data or other sources [12]. By doing so, that the adversary probes the system with brute-force attack can be modelled by assigning equal probabilities to all exploits. Similarly, intrusions by an experienced attacker who exploits vulnerabilities selectively to maximize his chance of success can be modelled by assigning higher probabilities to critical exploits.
3. We add a transition from each dangling state pointing back to the initial state s_0 with probability 1, modelling the situation that an adversary has come to a state where he cannot proceed with the intrusion and has to restart from initial state.

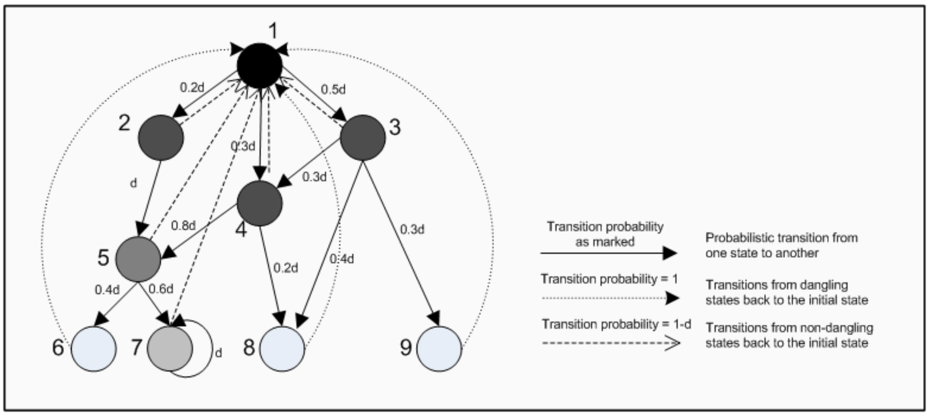
1
2
3
4
5
6
7
8
9
0.2d
0.5d
0.3d
d
0.3d
0.8d
0.3d
0.2d
0.4d
0.4d
0.6d
d
Transition probability as marked
Probabilistic transition from one state to another
Transition probability = 1
Transitions from dangling states back to the initial state
Transition probability = 1-d
Transitions from non-dangling states back to the initial state

Fig. 2. Transitions in attack models

Consider the graph illustrated in Figure 1 as an example. Assume we have some empirical data that enables us to estimate that whenever the system is in s_1, on average it will take the transition to s_2, s_3 and s_4 2, 5 and 3 times, respectively, out of ten. We can then place probabilities 0.2, 0.5 and 0.3 on these transitions. Similarly, assume that the empirical data enables us to place probability 0.3, 0.4 and 0.3 to the transitions taking s_3 to s_4, s_8 and s_9 respectively, probability 0.8 and 0.2 to the transitions taking s_4 to s_5 and s_8 respectively, and probability probability 0.4 and 0.6 to the transitions taking s_5 to s_6 and s_7 respectively. Figure 2 illustrates the graph with adjusted transition model from web graphs, which is a more accurate simulation of computer system state transition in an intrusion scenario. The intensity of color for each state s_j visualize the probability d_j that an intrusion is not detected at that state.

3.2 Transition Matrix Construction

To rank an attack model $M = (S, \tau, s_0, l)$, we need to construct the *transition matrix* $\overline{W} = \overline{w}_{ij}$, the matrix representation of state transitions in an attack model, where $\overline{w}_{ij}$ is the probability of the system being taken to state s_j from state s_i. Let $\tau(s_j \rightarrow s_i)$ denote the proportion between the number of exploits that take the system from s_i to s_j and the total number of exploits applicable to s_i and $l(s_i, s_j)$ denote the length of the shortest path between s_i and s_j, a concrete algorithm for constructing $\overline{W}$ is presented in Algorithm 1. Depth-first-search or model checker such as NuSMV [1] is first used to construct the $N \times N$ *adjacency matrix* AM where N is the number of reachable states in M, such that $AM[i, j] = 1$ if state s_j is one of the successor states of state s_i, otherwise $AM[i, j] = 0$. Then the *transition matrix* $\overline{W}$ is constructed following the above stated adjustment to state transitions in web graphs.

Reconsider the web graph illustrated in Figure 1 as a example. We now construct the *transition matrix* $\overline{W}$ according to the adjustment illustrated in Figure 2 using Algorithm 1. The generated $\overline{W}$ is shown in Equation 5 where each $d_i = d \times e^{-\lambda l(s_1, s_i)}$, d being the usual damping factor used in PageRank.

$$\overline{\mathbf{W}} = \begin{pmatrix} 0 & 1-d_2 & 1-d_3 & 1-d_4 & 1-d_5 & 1 & 1-d_7 & 1 & 1 \\ 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0.3d_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0.8d_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4d_5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.6d_5 & 0 & d_7 & 0 & 0 \\ 0 & 0 & 0.4d_3 & 0.2d_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3d_3 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5}$$

3.3 Ranking Attack Models

Following Mehta *et al.*'s definition, we define the rank for each state s_j in an attack model as the probability that s_j is reached from the initial state s_0. This can be recursively represented as

$$x_p = \sum_{q \in pa[p]} \overline{w}_{qp} \times x_q \tag{6}$$

When stacking all x_p into one vector $\mathbf{x}$, Equation 6 can be represented as

$$\mathbf{x} = \overline{W}\mathbf{x} \tag{7}$$

$\mathbf{x}$ in Equation 7 can be computed by multiple iterations through the following equation until a stationary state is reached.

$$\mathbf{x(t)} = \overline{W}\mathbf{x(t\text{-}1)} \tag{8}$$

Algorithm 1: Generate$\overline{W}(M)$

```
/* The function generates the adjusted transition matrix $\overline{W}$ from the
   given attack model $M$.                                                   */
/* Input: $M = (S, \tau, s_1, L)$: the attack model where $s_1$ is the initial
   state of $M$.                                                             */
/* Output: $\overline{W} = \overline{w}_{ij}$, where $\overline{w}_{ij}$ represents the probability of an
   adversary exploiting the vulnerability that takes the system from
   state $s_j$ to state $s_i$.                                                */
begin
    $AM$ = Construct_Adjacency_Matrix_From_Model($M$)
    /* Set the probabilities of transitions to and from the initial
       state                                                                 */
    for $i = 1$ to $N$ do
        if $AM[1, i] = 1$ then
            $\overline{w}_{i1} = \tau(s_1 \rightarrow s_i)$
        else
            $\overline{w}_{i1} = 0$
        if $\forall j, AM[i, j] = 0$ then
            $\overline{w}_{1i} = 1$
        else
            $\overline{w}_{1i} = 1 - d \times e^{-\lambda l(s_1, s_i)}$
    /* Set the probabilities of transitions to and from other states
       */
    for $i = 2$ to $N$ do
        for $j = 2$ to $N$ do
            if $AM[j, i] = 0$ then
                $\overline{w}_{ij} = 0$
            else
                $\overline{w}_{ij} = d \times \tau(s_j \rightarrow s_i) \times e^{-\lambda l(s_1, s_j)}$
end
```

If Equation 8 reaches a stationary state, i.e. $x(t) = x(t-1)$, after a long run of computation, all states in attack graph can be ranked. However, Equation 8 may or may not reach a stationary state after a long run of computation. Moreover, the result after multiple iterations may not be interesting (for example, the stationary state $\lim_{t\to\infty} \mathbf{x(t)}$ may be a vector of all 0s). A detailed proof of **Theorem 1** is provided in Appendix A to justify that Equation 8 constructed as above can always reach a non-trivial stationary state after multiple iterations.

Theorem 1. *Equation 8 converges at a non-trivial vector x^* where $\sum_i x_i^* = 1$ after multiple iterations.*

Given an attack model and empirical data that enables us to evaluate probabilities of different vulnerabilities being exploited, we first construct the *transition* *matrix* $\overline{W}$ as presented in Algorithm 1. We then assign random initial value to the rank of each state, and run Equation 8 for multiple iterations until it reaches the stationary state, guaranteed to exist by **Theorem 1**.

4 Implementation and Experiments

To evaluate the effectiveness of the proposed ranking scheme, we developed a toolkit in Java that ranks attack models with the proposed scheme. We ran the toolkit on the network example used by Mehta *et al* [16] and have the results compared with their ranking scheme. In this section, we first provide implementation details of the toolkit, then present the network model and the experimental results.

4.1 Implementation

The implementation toolkit is developed in Java but relies on NuSMV [1] for model checking functionalities, such as generating the complete set of reachable states given an initial state and the set of allowed state transitions. We made a minor modification to the source code of NuSMV (see below) to achieve interaction with our Java-based implementation toolkit. In the following, we provide details on the architecture of our implementation toolkit and its interaction with the modified NuSMV.

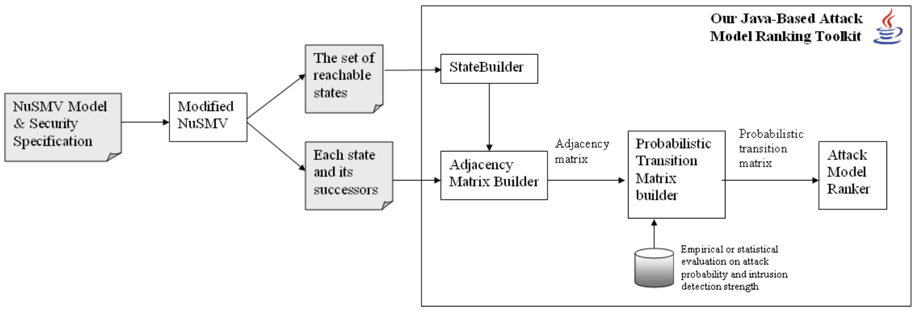
Our Java-Based Attack Model Ranking Toolkit
NuSMV Model & Security Specification
Modified NuSMV
The set of reachable states
Each state and its successors
StateBuilder
Adjacency Matrix Builder
Adjacency matrix
Probabilistic Transition Matrix builder
Probabilistic transition matrix
Attack Model Ranker
Empirical or statistical evaluation on attack probability and intrusion detection strength

Fig. 3. Toolkit Architecture

Toolkit Architecture. The architecture of our Java-based attack model ranking toolkit is illustrated in Figure 3. A network model along with the security specification written in NuSMV modelling language are fed to NuSMV. NuSMV then generates the complete set of reachable states S in the given model. We also modified NuSMV so that for each state $s \in S$ it generates the set of successors. The results generated as above are then saved as files, and feeded to the implementation toolkit to construct the adjacency matrix for states in the model. Combining the adjacency matrix, the empirical evaluation on the probability that each type of vulnerability is exploited, and the evaluation on the system's intrusion detection ability, the implementation toolkit generates the *transition matrix* $\overline{W}$ and ranks the states in the attack model as described in Section 3.

Toolkit Components

State Builder. With the NuSMV command *print_reachable_states -v*, we generate the set of reachable states from the specified system initial state which are saved to a text file. The State Builder then reads the set of reachable states into Java-specific representation from the generated text file.

Adjacency Matrix Builder. We modified NuSMV such that it generates and saves into a text file the successor states of a given state with the *-st* command line option. Iteratively using the *-st* option for each reachable state, the Adjacency Matrix Builder generates an $N \times N$ adjacency matrix AM where N is the number of states in the attack model such that $AM[i,j] = 1$ if state j is one of the successor states of state i, otherwise $AM[i,j] = 0$.

Transition Matrix Builder. Combining the adjacency matrix, the empirical evaluation on the probability that each type of vulnerability is exploited, and the evaluation on the system's intrusion detection ability, the Transition Matrix Builder follows Algorithm 1 to generate the *transition matrix* $\overline{W}$.

Attack Model Ranker. Given the *transition matrix* $\overline{W}$, the Attack Model Ranker computes the ranks for all reachable states in the attack model using Equation 8 iteratively until the stationary is reached. A non-trivial stationary state is guaranteed to exist after multiple iterations by **Theorem 1**.

4.2 The Network Model for Experiments

The network model used for our experiments is illustrated in Figure 4. There are two target hosts ip_1 and ip_2, and a firewall separating them from the rest of the Internet. As shown each host is running two of three possible services (ftp, sshd, database). We model the same 4 types of atomic attacks summarized in Table 2 as in [15] [16] for comparable results. A detailed explanation of each attack follows.

The intruder launches his attack starting from an external machine ip_a that lies outside the firewall. His eventual goal is to gain access to the database. For that, he needs root access on the database server ip_2.

Table 2. Atomic Attacks Modelled in the Sample Network

Attack	**Vulnerability Exploited**
▶ sshd buffer overflow	Some versions of ssh are vulnerable to buffer overflow
▶ ftp.rhosts	Exploiting the vulnerability resulting from a writable ftp home directory
▶ remote login	Remote trust relation between machines
▶ local buffer overflow	Some `setuid root` executables are vulnerable to buffer overflow

We construct a finite state model of the network such that each state represents the system state including trust relation, connectivity, and adversary privilege on each machine, and each state transition corresponds to a single atomic attack which takes the system from one state to another.

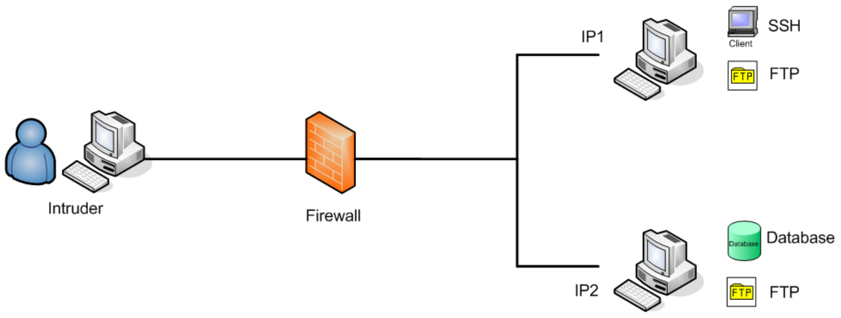

Fig. 4. Network

Connectivity and Trust Relation. Connectivity models the connection between two machines. We denote the connectivity by a binary relation $Reachable \subseteq Host \times Host$, where $Reachable(h_1, h_2) = 1$ if h_1 can connect to h_2, otherwise $Reachable(h_1, h_2) = 0$, i.e. either there is no physical link between h_1 and h_2, or the link is blocked by the firewall. Assuming the firewall policy is that the ftp server (ip_1) is publicly accessible while the database server (ip_2) can only be accessed internally, the connectivity relation is shown in Table 3. Similarly, we denote trust relation between machines by a binary relation $Trust \subseteq Host \times Host$, where $Trust(h_1, h_2) = 1$ if a user on h_1 can login to h_2 remotely without specifying a password, $Trust(h_1, h_2) = 0$ otherwise. The trust relation is summarized in Table 4.

The Adversary and Privilege. Privileges are $\{none, user, root\}$. There is an ordering of privileges: $none < user < root$. The adversary has $root$ on ip_a and no privileges on other machines initially. We use the function $plvl_A(H) : Hosts \rightarrow \{none, user, root\}$ to denote the level of privilege that intruder A has on machine H.

Vulnerabilities and Atomic Attacks. We model the same 4 types of attacks as in [15] [16], each taking the modelled network from one state to another as described by the "effect" section of the attack. An attack is only applicable when both the network precondition and intruder precondition are satisfied. Throughout the following description, we denote source and target machine by S and T. To simplify the notations, we use ssh_H, ftp_H and $local_H$ to denote the presence of a vulnerability by running ssh service, ftp service and a `setuid root` executable respectively on host H.

Table 3. Connectivity

Reachable	ip_a	ip_1	ip_2
ip_a	1	1	0
ip_1	1	1	1
ip_2	1	1	1

Table 4. Trust Relation

Trust	ip_a	ip_1	ip_2
ip_a	1	0	0
ip_1	0	1	1
ip_2	0	1	1

1. sshd buffer overflow: Some versions of ssh services are vulnerable to a buffer overflow attack that allows an intruder to obtain a root shell on the target machine. Formally,

```
attack sshd-buffer-overflow is
    intruder preconditions
        [User-level privileges on host S]
        $plvl_A(S) \geq user$
    network preconditions
        [Host T is running a vulnerable version of ssh service]
        $ssh_T$
        [Host T is reachable from S]
        $Reachable(S,T) = 1$
    intruder effects
        [Root-level privileges on host T]
        $plvl_A(T) = root$
end
```

2. ftp .rhosts: With a writable home directory and an executable command shell assigned to anonymous ftp users, an intruder can modify the .rhosts file in the ftp home directory, so as to create a remote login trust relationship between his machine and the target machine. Formally,

```
attack ftp-rhosts is
    intruder preconditions
        [User-level privileges on host S]
        $plvl_A(S) \geq user$
    network preconditions
        [Host T is running a ftp service in a writable directory,
        which gives a user shell to ftp users]
        $ftp_T$
        [Host T is reachable from S]
        $Reachable(S,T) = 1$
    network effects
        [Trust relation between the intruder's machine and the target]
        $Trust(S, T) = 1$
end
```

3. remote login: Using an existing remote login trust relationship between two machines, the intruder can login from his machine to the target and obtain a user shell without supplying a password. Although remote login is usually considered a legitimate operation by regular users, it is however an atomic attack from an intruder's viewpoint. Formally,

```
    attack remote-login is
        intruder preconditions
            [User-level privileges on host S]
            $plvl_A(S) \geq user$
        network preconditions
            [Host T trusts S]
            $Trust(S, T) = 1$
            [Host T is reachable from S]
            $Reachable(S, T) = 1$
        intruder effects
            [User-level privileges on host T]
            $plvl_A(T) = user$
    end
```

4. local buffer overflow: The attacker exploits a buffer overflow vulnerability in a `setuid root` executable to gain root access. Formally,

```
    attack local-buffer-overflow is
        intruder preconditions
            [User-level privileges on host T]
            $plvl_A(T) \geq user$
        network preconditions
            [Host T runs a vulnerable version of a setuid root executable]
            $local_T$
        intruder effects
            [Root-level privileges on host T]
            $plvl_A(T) = root$
    end
```

4.3 Experimental Results Analysis and Evaluation

Let the security property be "intruder cannot gain root access on ip_2". We ran our attack model ranking toolkit presented in Section 4.1, and visualized the results with the *graphViz* package [3]. Figure 5 illustrates the result obtained as such. For each state, the intensity of color is proportional to the rank of that state. Any path in the graph from the root node to a leaf node represents a sequence of exploits with which the intruder can achieve his final goal. It can be seen that *local buffer overflow* and *remote login* are critical exploits as each path from the root node to a leaf node has exploited them at least once.

After fixing either *local buffer overflow* or *remote login*, NuSMV asserts security property “intruder cannot gain root access on ip_2” to be true. On the other hand, *ftp.rhost* and *ssh buffer over flow* are non-critical exploits as an intruder can still reach his final goal without either of them.

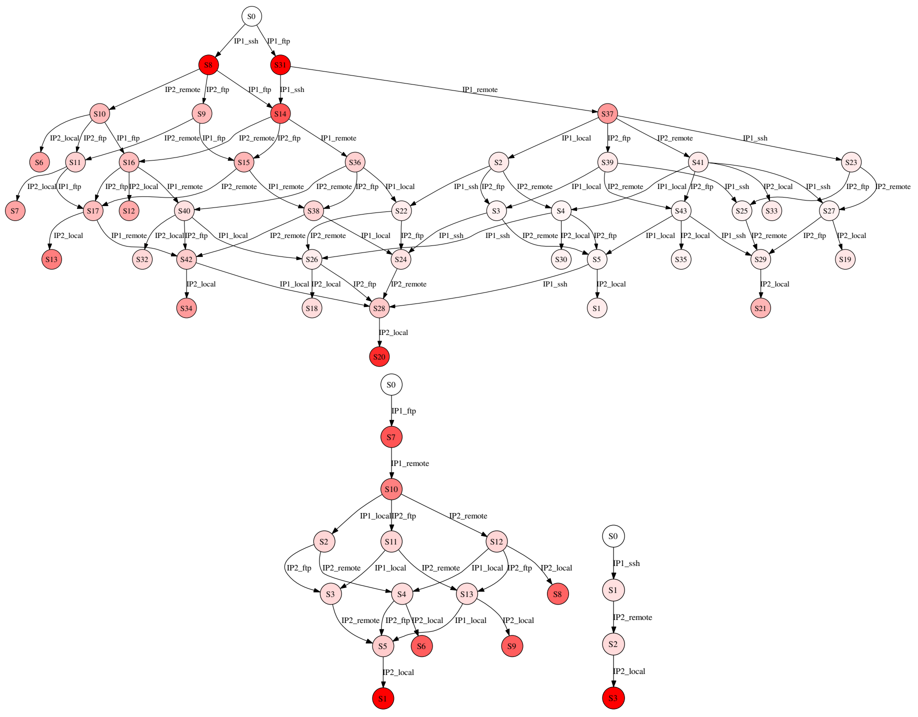

Fig. 5. Comparison of Ranked Attack Models. (a) The complete ranked attack model (b) Attack Model after fixing up the SSH vulnerability (c) Attack Model after fixing FTP vulnerability.

To investigate how an attacker selectively exploiting vulnerabilities affects his chance of compromising the system, we vary the probability assigned to each type of exploit and have other exploits divide the remaining probability equally. Changes to the ranks of states resulting from varying the probabilities of the exploits reflects how an attacker selectively exploiting vulnerabilities affects his chance to compromise the system. We also set the rate by which probability of an intrusion remaining undetected decays with the shortest path to 0, so that changes to the rank of a state are the result only of varying the probabilities of the exploits. The experimental result is plotted in Figure 6 where the Y-axis represents the total rank of error states, i.e. the probability of an adversary reaching his goal. It can be seen that the total rank of error states increases as the attacker prioritize critical exploits *local buffer overflow* and *remote login*, modelled by assigning higher probabilities to the two attacks. Similarly, the adversary’s chance to succeed decreases as he prioritize non-critical exploits *ftp.rhosts* and *sshd buffer overflow*. In general, our scheme produces a higher rank when the attacker prioritize critical exploits and hence has better chance to succeed. The rank produced by our scheme joins the rank by Mehta's scheme at the equal probability point, i.e. where all exploits are assigned equal probabilities.

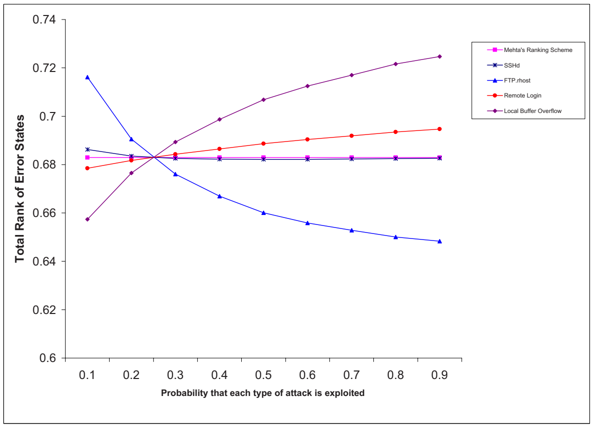

Fig. 6. Rank varies with attack probabilities

To investigate the effect that probability of an intrusion remaining undetected decays at a rate proportional to length of the shortest path from initial state, we vary the decaying rate λ while assigning equal probabilities to all exploits. The experimental result is plotted in Figure 7. It can be seen that the total rank of error states increases as the decaying rate decreases. This corresponds to the fact that an attacker has less chance of success on well-protected systems such as systems implementing "defense-in-depth" which at each step of the intrusion and thus on the whole has a higher probability of being able to discover and thwart the intrusion. The ranks produced by our ranking scheme consistently remain lower than the rank by Mehta's scheme, resulting from the decaying of probability that an adversary remains undetected and is able to proceed.

Figure 8 plots the experimental result by the overall effect of various decaying rates and varying probability assigned to each type of exploit (still other exploits divide remaining probability equally). It can be seen that ranking of system states is dominated by decaying of probability that intrusion remains undetected. Variation in probability assigned to each type of exploit only affects ranking of states to a minor extent. It can also be see that, the greater the decaying rate is, the less variation in probability assigned to each type of exploit affects ranking of states. The result reveals that deployment of well-protected system offsets experienced intruder's strategy in selectively exploiting vulnerabilities to maximize his chance of success, lowering his chance of success to no more than that of brute-force type of attack.

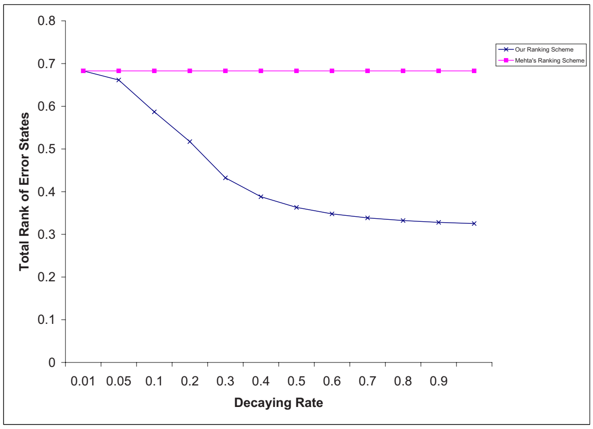

Fig. 7. Rank varies with decaying rate

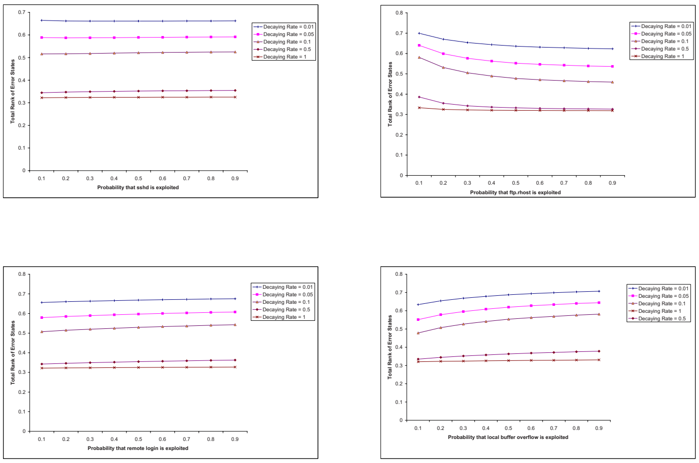

Fig. 8. Rank varies with decaying rate and attack probabilities

The experimental results and analysis presented above demonstrates the advantage and application of the proposed ranking scheme. Firstly, it considers the effect on ranking of system states by an intruder selectively exploiting vulnerabilities to maximize his chance of success. Secondly, it is able to model the effect on ranking of system states by system intrusion detection ability that aims at thwarting exploits of vulnerabilities that take the system to another state where the intruder gains an elevated privilege. Therefore, the proposed ranking scheme can rank attack models more accurately, and provide more realistic evaluation on the probability that a system is in a compromised state.

Intuitively, the probability of a system being in a compromised state increases with the probability that an intruder is able to prioritize critical exploits, and with weakening system intrusion detection ability; however, our ranking scheme provides a quantitative measure for the increase. The proposed scheme can also assist network researchers and architects in network design and analysis, e.g. determining the minimum intrusion detection strength required to thwart the best effort in selectively exploiting vulnerabilities by intruders.

5 Conclusion

As the size and complexity of attack models/graphs usually greatly exceed human ability to visualize, understand and analyze, ranking of states is often required to identify important portions of attack models/graphs. Mehta *et al* proposed a ranking scheme based on the PageRank algorithm used by Google to measure importance of web pages on World Wide Web. We extend their scheme by modelling an attacker selectively exploiting vulnerabilities to maximize his chance of compromising the system, and intrusion detection ability of computer systems detecting and preventing attackers to exploit system vulnerabilities. With the proposed ranking scheme, evaluation on system intrusion detection ability or attackers' ability in relation to probabilistically exploit vulnerabilities, when available from for example empirical data or log statistics, can be used to obtain more accurate ranks of computer system states modelled by attack models and attack graphs.

References

1. NuSMV: a new symbolic model checker. http://nusmv.irst.itc.it/.
2. A. Y. NG, A. X. Zheng, and M. I. Jordan. Link analysis, eigenvectors and stability. In *Proceedings of International Conference on Research and Development in Information Retrieval (SIGIR 2001)*, New York, 2001. ACM.
3. AT&T Research. http://www.graphviz.org/.
4. B. B. Madan, K. G. Popstojanova, K. Vaidyanathan, and K. S. Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. In *Dependable Systems and Networks-Performance and Dependability Symposium*, number 167-186, 2004.
5. C.A. Phillips and L. P. Swiler. A graph based system for network vulnerability analysis. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, 2000.
6. G. H. Golub and V. Loan. *Matrix computation.* The Johns Hopkins University Press, 1993.
7. J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of the Second IEEE International Information Assurance Workshop*, 2004.
8. M. Bianchini, M. Gori, and F. Scarsell. Inside PageRank. *ACM Transactions on Internet Technology*, 5(1):92–118, Feb 2001.
9. M. Dacier, Y. Deswarte, and M. Kaaniche. Quantitative assessment of operational security: Models and tools. Technical Report 96493, LAAS, May 1996.

10. O. Sheyner, J. Haines S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.
11. Y. Deswarte R. Ortalo and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *Software Engineering*, 25(5):633–650, 1999.
12. R. Ortalo, Y. Deshwarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. In *IEEE Transactions on Software Engineering*, pages 71–79, 1999.
13. S. Brin, L. Page, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical Report 1999-66, Standford University, 1999.
14. S. Jha and J. Wing. Survivability analysis of networked systems. In *23rd International Conference on Software Engineering(ICSE01)*, number 03-07, 2001.
15. S. Jha, O. Sheyner, J. Wing. Two Formal Analysis of Attack Graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW'02)*, page 49. IEEE, 2002.
16. V. Mehta, C. Bartzis, H. Zhu, E. Clarke and J. Wing. Ranking Attack Graphs. In *Proceeding of the 9th International Symposium On Recent Advances In Intrusion Detection*, Hamburg, Germany, September 2006. Springer.

A Proof of Theorem 1

Lemma 1. *Each column of the transition matrix $\overline{W}$ constructed by Algorithm 1 sums to 1, i.e.* $\sum_{i=1}^{N} \overline{w}_{i,j} = 1$.

Proof of the above lemma follows directly the way by which $\overline{W}$ is constructed.

Theorem 1. Equation 8 converges at a non-trivial vector x^* where $\sum_i x_i^* = 1$ after multiple iterations.

Proof: Consider a linear transformation of x_p defined in Equation 2. Let

$$x'_p = c_1 \times x_p + c_2 = c_1 \times \sum_{q \in pa[p]} x_q \times \overline{w}_{pq} + c_2 \tag{9}$$

where $c_2 = \frac{1-c_1}{N}$. Stacking all x'_p into one vector x' and using iterative expression, Equation 9 is represented as

$$\mathbf{x(t)'} = c_1 \overline{W} \mathbf{x(t\text{-}1)'} + c_2 \Pi_N \tag{10}$$

A well-known theory states that the condition that $MX(K+1) = NX(K) + b$ converges at $(M-N)^{-1}b$ is $\rho(M^{-1}N) < 1$ [6].

Here we have $M = I$ and $N = c_1\overline{W}$. Therefore $\rho(M^{-1}N) = \rho(c_1\overline{W}) = c_1\rho(\overline{W})$. Assume x is an eigenvector of $\overline{W}$ and λ is the associated eigenvalue, then $\overline{W}x = \lambda x$, i.e. $\forall i$, $\sum_{j=1}^{N} \overline{w}_{i,j} x_i = \lambda x_i$. Extracting the common factor x_i, this can be written as $x_i(\sum_{j=1}^{N} \overline{w}_{i,j} x_i - \lambda) = 0$. As x is an eigenvector, there exist non-zero x_i. Therefore, $\lambda = \sum_{j=1}^{N} \overline{w}_{i,j} x_i$. Following lemma 1, $\sum_{i=1}^{N} \overline{w}_{i,j} = 1$, we know that $\lambda = \sum_{j=1}^{N} \overline{w}_{i,j} x_i = 1$. Therefore, $\rho(\overline{W}) = 1$. On the other hand,

$0 < c_1 < 1$. As a result, $\rho(M^{-1}N) = c_1\rho(\overline{W}) < 1$, and hence Equation 10 converges at a stationary state $\lim_{t\to\infty} \mathbf{x(t)'}$.

We then prove by induction on t that the stationary state $\| \lim_{t\to\infty} \mathbf{x(t)'}\|_1$ of Equation 10 is a unit vector, i.e. $\| \lim_{t\to\infty} \mathbf{x(t)'}\|_1 = 1$.

1. For $t = 0$, Let $\mathbf{x(0)'} = \frac{1}{N}\Pi_N$; hence $\|\mathbf{x(0)'}\|_1 = 1$.
2. Let $t > 0$ and assume by induction that $\|\mathbf{x(t)'}\|_1 = 1$. Then, based on the definition of stochastic matrices,

$$\begin{aligned}\|\mathbf{x(t{+}1)'}\| = \Pi'_N\mathbf{x(t{+}1)'} &= c_1\Pi'_N W\mathbf{x(t)'} + c_2\Pi'_N\Pi_N \\ &= c_1\Pi'_N\mathbf{x(t)'} + (1 - c_1) = 1 \end{aligned} \tag{11}$$

We hence proved that with the initial *unit vector* $\mathbf{x(0)'} = \frac{1}{N}\Pi_N$, $\| \lim_{t\to\infty} \mathbf{x(t)'}\|_1 = 1$. As stationary solution of Equation 10 is independent of the initial value $\mathbf{x(0)'}$ [6], it can be concluded immediately that $\| \lim_{t\to\infty} \mathbf{x(t)'}\|_1 = 1$ with any initial vector $\mathbf{x(0)'}$. Note that it can be seen from Equation 9 that $x'_p > 0$; hence $\|\mathbf{x(t)'}\|_1 = \sum_{p=1}^N x'_p = 1$

The stationary state $\mathbf{x(t)}$ of Equation 2 can be retrieved from $\mathbf{x(t)'}$ with linear conversion $\mathbf{x(t)} = \frac{(\mathbf{x(t)'} - c_2)}{c_1}$. $\mathbf{x(t)}$ is not trivial, because

$$\sum_{p=1}^{N} x_p = \sum_{p=1}^{N} \frac{x'_p - c_2}{c_1} = \frac{\sum_{p=1}^N x'_p - N \times c_2}{c_1} = \frac{1 - N \times c_2}{c_1} = 1 \tag{12}$$

That is, the stationary state $\mathbf{x(t)}$ is a unit vector. □