SECURITY RISK PRIORITIZATION FOR LOGICAL ATTACK GRAPHS

by

HUSSAIN ALMOHRI

B.S., Kuwait University, Kuwait 2006

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2008

Approved by:

Co-Major Professor
Xinming Ou

Approved by:

Co-Major Professor
William H. Hsu

Copyright

Abstract

To prevent large networks from potential security threats, network administrators need to know in advance what components of their networks are under high security risk. One way to obtain this knowledge is via attack graphs. Various types of attack graphs based on miscellaneous techniques has been proposed [1–5]. However, attack graphs can only make assertion about different paths that an attacker can take to compromise the network. This information is just half the solution in securing a particular network. Network administrators need to analyze an attack graph to be able to identify the associated risk. Provided that attack graphs can get very large in size, it would be very difficult for them to perform the task. In this thesis, I provide a security risk prioritization algorithm to rank logical attack graphs produced by MulVAL (A vulnerability analysis system) [6]. My proposed method (called StepRank) is based on a previously published algorithm called AssetRank [7] that generalizes over Google's PageRank [8] algorithm. StepRank considers a forward attack graph that is a reversed version of the original MulVAL attack graph used by AssetRank. The result of the ranking algorithm is a rank value for each node that is relative to every other rank value and shows how difficult it is for an attacker to satisfy a node.

Table of Contents

List of Figures

List of Tables

Acknowledgments

This work could not be accomplished without great and helpful advice and directions of my major professor **Dr. Xinming Ou.** He faithfully worked tight with me on this project and advised me the best way he could.

I would also thank **Dr. William H. Hsu** for his co-advising this project and his brilliant contributions.

Dr. Doina Caragea who agreed to be part of my thesis committee and provide feedback on my work is very appreciated for her help.

Thanks to **Reginald E. Sawilla** who also provided very important feedback on the work presented in this thesis.

Dedication

Dedicated to my father the creative writer who is not only a parent but also a teacher.

Chapter 1

Introduction

1.1 Background

Maintaining a secure and reliable network is a critical task for network owners and operators. It is often the case that confidential or sensitive data is often stored within the boundaries of a particular enterprise network. Moreover, any interruption of work flow in an enterprise network may lead to costly losses. In order to keep the information and network activity safe, system administrators need to secure their networks. One important step towards securing a large network is to prevent the harm from identifiable threats. Significant effort is expended in many enterprises to identify vulnerabilities and the range of their security risk in order to have safer systems. The result of this effort can be used to find and thereby prevent possible ways of compromising a particular network. Among the tools that can help in preventing future security threats is attack graph. In addition, vulnerability databases [9] and the information they provide regarding individual vulnerabilities are quite beneficial.
In this Chapter we briefly describe attack graphs and *MulVAL*, a vulnerability analysis system that we used to produce attack graphs. Later we formally state the problem and discuss previous work on security risk measurement.

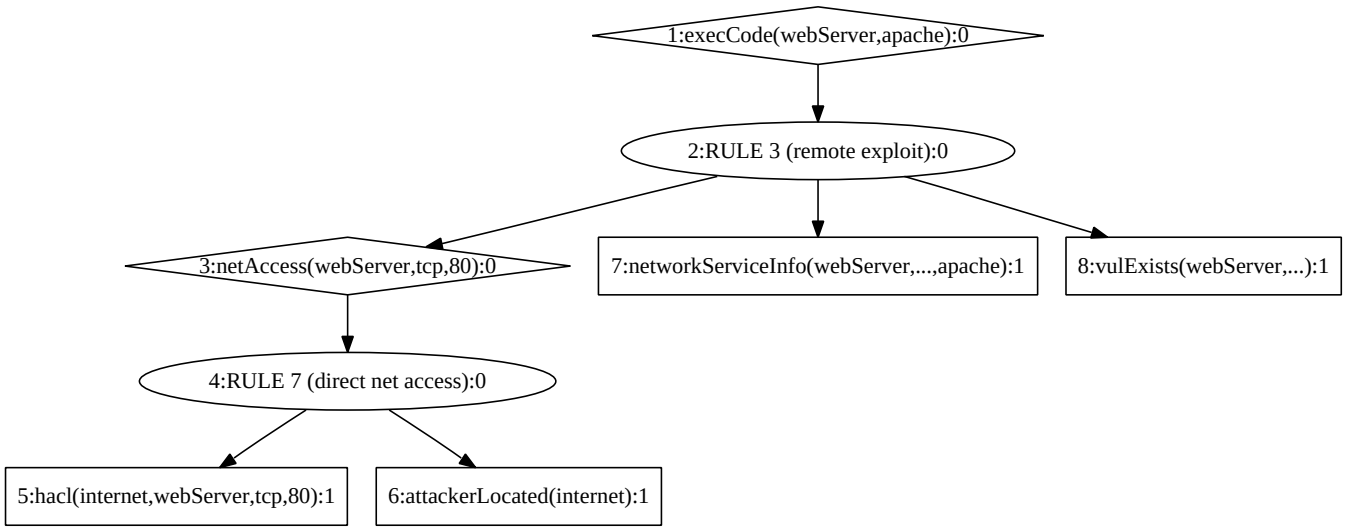

Figure 1.1: *A small MulVAL attack graph.*

1.1.1 Attack Graphs

An attack graph is a formalism that uses network configuration and vulnerability data to analyze a particular network of computers against possible security threats. There have been various proposals for producing attack graphs based on a number of techniques. For example, there are attack graph generation methods based on model checking [3]. Another type of attack graph generation is based on first-order logic [2]. Attack graphs can also be modeled to show states of system execution showing the necessary requirements of a state [4].

In this work, we use the logical attack graphs that are produced by MulVAL [6]. Figures 1.1 and 1.2 show examples of MulVAL and forward attack graphs which is obtained from revesing all the edges in MulVAL's attack graphs. It has been shown that the size of a logical attack for a network with n machines is bounded by $O(n^2)$ [2]. That means the number of nodes can be very large which makes the manual analysis of such graphs a very hard task.

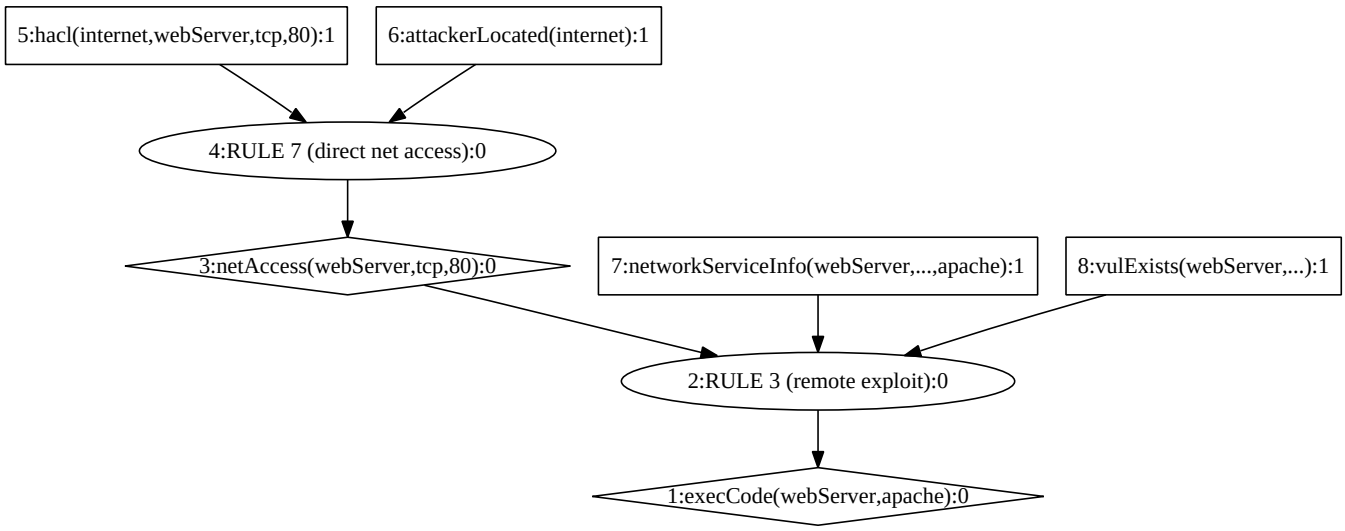

Figure 1.2: *A small forward attack graph.*

1.1.2 MulVAL

MulVAL (Multi-host, multi-stage Vulnerability Analysis) uses Datalog [10] (a subset of Prolog) to produce logical attack graphs. MulVAL's framework is shown in Figure 1.3 [6]. It takes as input a set of first-order logical configuration predicates and produces an attack graph based on that. These configuration predicates include network specific security policies, binding information and vulnerability data gathered from vulnerability databases. MulVAL identifies possible policy violations through logical inference. From a practical point of view, knowing the possibility of potential attacks on a network is just half the solution. The other important half is to help system administrators decide how to prevent such a threat.

1.2 Problem Statement

MulVAL's output provides a useful attack graph that can tell if the overall network admits possible threats. It can also show various ways in which the network can be attacked to gain certain privileges. After letting MulVAL analyze the network and produce an attack

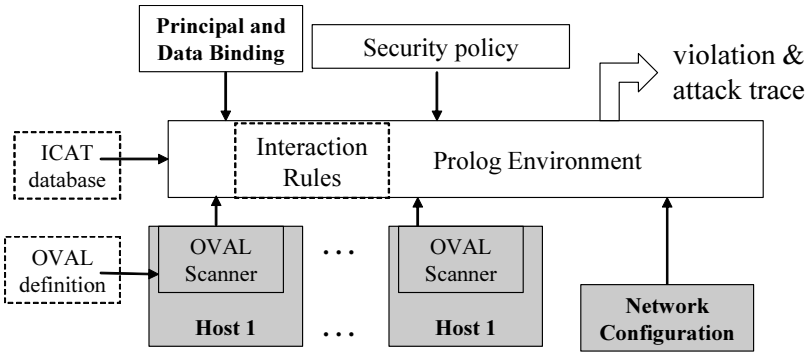

Figure 1.3: *MulVAL's Framework.*

graph, a network administrator needs to diagnose the problems that enable the attacks to happen. Unfortunately, MulVAL's graphs can be very large given relatively small networks. This makes it more difficult to interpret the graph and identify nodes through which more attack paths pass. These nodes are more important to system administrators. To illustrate the issue consider the network configuration in Figure 1.4. In this configuration, an attacker

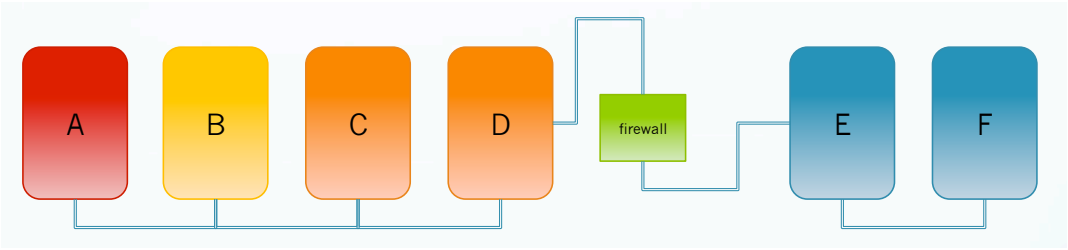

Figure 1.4: *An example of a network configuration with 6 hosts and 2 subnets.*

whose goal is to gain access on machine F is able to start an attack path using hosts A and B. All the hosts have the potential of being compromised. However, E can only be accessed from C and D. The MulVAL attack graph corresponding to this configuration is shown in figure A.2. This graph has 50 nodes with a lot of dependency relations directed by the graph edges. For such a small configuration, it is hard to manually analyze the corresponding attack graph and find out which nodes are the most risky ones. Comparing graph of figure A.2 to the graph of figure A.1 with only 17 nodes we observe a huge increase

in the number of nodes by adding only 3 more hosts.

In practice, networks may have absolutely larger configurations with more complex policies and rules. It can be imagined from the example graphs how big the attack graph would be for a network of 100 or more hosts with complicated configurations. Further, it would be nearly impossible for a team of system administrators to efficiently and effectively analyze the graph of a very large enterprise with thousands of nodes. Thus, it is very convenient to have a mechanism to rank the nodes on an attack graph. By analyzing the ranks one can have a clearer idea of what parts of the configuration are putting the most amount of risk on the whole network. For instance, by sorting the ranks and considering first 10% most risky nodes, system administrators can find solutions that aim to reduce the associated risk.

In this work, we present a method to rank an attack graph produced by MulVAL. Our approach is based on a previously published algorithm called AssetRank [7], which is in turn a generalization over Google's PageRank algorithm [8]. Moreover, we base our ranking on a forward attack graph in which the parent-child relation is reversed compared to the original MulVAL attack graphs. The motivation for using a forward attack graph is to formalize a realistic scenario in which attackers do not have an overall picture of the attack graph. That is, an attacker can see one step ahead at any given time in a forward attack graph and at each step decides on the next stage in his attack path. Another important reason to use a forward attack graph is that we would like to compare risks whereas AssetRank's strategy is to capture "enabling capabilities". The computation is carried out using two algorithms. One computes weights for the edges of the attack graph and the other computes the overall ranks. In Chapter 2 we describe the theoretical foundation of our model along with an interpretation of the rank values. In Chapter 3 we discuss how one can obtain the parameters to our ranking equation. Finally in Chapter 4 we provide the results of some experiments and discuss how to validate the results.

1.3 Related Work

1.3.1 PageRank

In order to rank webpages over the web, Page and Sergey introduced PageRank algorithm. The idea of PageRank is to treat the links between webpages as a dependency graph [11]. In such a model, each page is regarded as edges in a graph vertex. If there is a link from a page p_1 to a page p_2, then it is represented by an edge (p_1, p_2) between the two vertices. The ranking process proceeds from p1 to p2 and computes the ranks of child nodes from those of their parents. The overall computation is viewed as a random walk in which the surfer randomly chooses among the available outgoing edges to go to the next page. The surfer will also have the option to stop and jump to a new page when surfing the pages.

1.3.2 AssetRank

The idea of PageRank was later applied to security risk measurement [7]. The analogy between webpages and MulVAL's attack graph was that in both the relations could be viewed as a dependency graph. In an attack graph, the relation (u, v) means node u depends on v. An example in MulVAL language could be that `execCode(Host,Permission)`could be satisfied with `hasAccount(Principal,Host,Permission)`.The main difference between PageRank and AssetRank is how they handle the AND nodes. PageRank could only consider OR nodes. This is because a random surfer can just choose between any of the links on the page to surf more pages. Thus, there need not be any AND relation between the links of a page. However, in an attack graph, one node may need to be satisfied by all of its children to be enabled, thus forming an AND relation between the outgoing edges.

AssetRank computes the ranks by means of attacking agents. These agents start at the root of the graph and have complete information about the whole graph. They follow the attack paths based on a probability distribution until they reach the leaves of the graph. As the agents move, the ranks flow from parent nodes to the children. When propagating the rank values from an OR node, the value will be split between its children (similar to

PageRank). By contrast, for an AND node, the value of the parent will be replicated to all of its children. Ou and Sawilla use this method of handling OR and AND nodes to capture the corresponding relationships and the possible attack patterns that they represent.

1.3.3 Other Works

Sheyner *et al.* provide an attack graph based on model checking. They formalize the intrusion attack in a finite state model. Although they do not provide a complete ranking strategy, they provide a method to compute minimal critical attack sets based on user-defined measures [3]. Another work in security risk analysis [12] performed direct analysis on network configuration and it did not use attack graphs. The threat measure consists of two parts: the threat likelihood and possibility of attack propagation throughout the network.

Mehta *et al.* provide a ranking method using state enumeration attack graphs [13]. Similar to AssetRank, they have applied the idea of PageRank but on a different type of attack graphs (compared to the dependency attack graphs used in AssetRank). Because of the difference in the type of attack graph they use, the interpretation of the ranking (which is affected by the semantics of the attack graph) also differs significantly.

In [14] a software system that analyzes the network through Topological Vulnerability Analysis is discussed. This visualization tool is an integrated component of the CAULDRON attack graph tool that was developed at George Mason University. CAULDRON maintains a database of attacker exploits and uses network configuration as input together with exploit data to produce a graph of all possible ways to compromise the network. A powerful utility of CAULDRON is the ability to perform what-if analysis in which one can specify the start point and goals of the attacker.

Previous work on probabilistic based security metric has also been proposed [15]. This approach also uses a forward attack graph with two types of scores for each type of node.

Both condition [1] and exploit [2] nodes have intrinsic and cumulative scores. All computed scores and those scores provided as parameters, represent probabilities. The authors also discuss three types of cycles that can appear in their attack graph. They provide solutions to each type of cycles.

[1] Condition node represents system state
[2] Exploit node represents transition between system states

Chapter 2

StepRank

In this chapter, we discuss the theoretical details of our proposed algorithm which we call StepRank. The name was chosen to reflect the fact that in our model, we assume that the attacker can only see one *step* ahead when trying to follow an attack path towards hi goal. In Section 2.1 we give a formal definition of logical attack graphs and in 2.2 we discuss its difference with what we call a forward attack graph. In sections 2.3 through 2.7 we describe the details of our ranking formula and finally in 2.8 the procedure to compute StepRank is presented.

2.1 MulVAL Logical Attack Graph

A logical attack graph can formally be defined as $G = (V, E, w, p, t)$ where V is the set of vertices in and E is the set of edges of the form (u, v) meaning u depends on v. An example is shown in Figure 2.1. The definition describes three types of mapping: w maps an edge to its weight; p maps a vertex to its personalization (intrinsic) value; t maps a vertex to its type (AND, OR, SINK). Every attack graph has a root which represents the ultimate goal to be achieved by possibly different attack paths. We further define the in-degree and out-degree of any vertex v in G as $N^+(v) = \{w \in V : (v, w) \in E\}$ and $N^-(v) = \{w \in V : (w, v) \in E\}$ respectively. A SINK vertex is a vertex v with $N^+(v) = 0$.

MulVAL produces attack graphs based on network configuration that is converted into

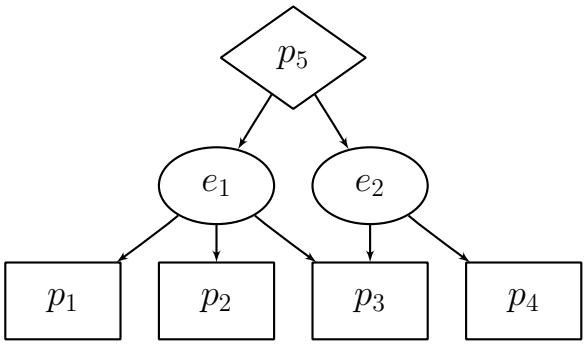

Figure 2.1: *A dependency graph.*

Datalog predicates. It also has a set of system specific rules that are of the form

$$R(x_1, .., x_n) \leftarrow Pr$$

where Pr is a set of conjuncted first order predicates. Note that Datalog cannot contain functions as parameters to a predicate.

AssetRank computes the ranks of attack vertices from a combination of two pieces of information. One is the flow of rank from parents to children which is carried out based on the type of the children. The other is the intrinsic value of a vertex which represents a vertex's weight by its own.

2.2 Forward Attack Graph

Let $(u, v) \in E$ be an edge in a MulVAL attack graph G. Then $(v, u) \in E_f$ is an edge in the corresponding *forward attack graph* $G_f = (V_f, E_f)$. Thus, an edge (u, v) in G_f means that v leads to u. The mapping t now maps a vertex to either of AND,OR or SOURCE where a vertex v with type SOUCRE has $N^+(v) = 0$.

Our ranking approach is based on the idea of a forward dependency attack graph. Compared with MulVAL attack graphs, all the vertex edges are reverted and as a result all the attack paths lead from some starting points to an ultimate goal (G_f's sink node). The rationale behind using such a graph instead of the original graph is to mimic realistic attack scenarios.

In reality, attackers will have initial information about where an attack may start and what his final goal is. However, the attacker will not necessarily know about all intermediate predicates (denoted by nodes) that he has to satisfy in order to reach his goal. Therefore, in a forward attack graph there will be some nodes which we call *gate nodes* that represent the points where an attacker can start an attack scenario. Knowing how to satisfy gate nodes, the attacker can start his attack and can only see one step ahead. This single step is obtained from the outgoing edges of the current node. This process is repeated until the attacker reaches graph's sink where he will stop.

The described realistic attack scenario is easy to formalize with a forward attack graph but not with the original one. This is because, in an original attack graph the attacker should have complete information about the attack graphs and different paths that may be used to achieve a goal. This follows from the semantic of the edges in an attack graph, which entail dependency of the current node over the children.

2.3 Methodology

StepRank is designed to provide a relative rank value for every vertex of a forward logical attack graph. We have developed a relative ranking rather than quantitative ad hoc ranking, to achieve sound design principle and semantics without overspecifying the ranking measure. Using the forward attack graph, a rank value r_v for a vertex $v \in V$ depends on the ranks of its parents. The ranks are injected by those nodes that represent the starting points for an attacker. Once the ranking process is started, the produced ranks will flow level-by-level until it reaches the root of the attack graph. Therefore, the rank values are relative to the particular attack graph that they were produced from. The rank values are always normalized so that we have $0 < r_v < 1$ for all v and $\sum r_v = 1$. In computing ranks, we always assume the most severe attacks that can happen. As the number of attackers reaching a particular vertex increases, the security situation gets worst and more severe. Based on this assumption, the final computed value of r_v for any vertex v represents the

most risky situation that can happen with the vertex.

2.4 Weights of Edges

When an attacker satisfies a certain vertex $v_i \in V$, if v_i is not the goal vertex he can still continue his attack until reaching the goal. Some vertices can have $N^+(v) > 1$ which means there are alternative paths that could be pursued by the attacker. For such a situation, we need to have a representation that can model the fact that an attacker will prefer an easier step to a harder one. This selection can be done by having a probability distribution over the outgoing edges that can determine the likelihood that an attacker will use any given edge. We provide weights for every edge in the graph to represent the values of the probability distribution. Thus, if we have two outgoing edges, one with a weight of 15% and the other with a weight of 35%, then the latter edge carries higher rank value to the next step.

The probability distribution should reflect the likelihood of using an edge based on the knowledge of system administrators and security experts. This knowledge could concern the nature of the vertices that are connected to these edges. For example, if we know that a vertex is much more difficult to satisfy than some other vertex, then the edge leading to that vertex should reflect a lower probability to show the likelihood that an attacker will go that way. This is intuitive, because real attackers often go through easier paths to increase their probability of success.

In our current implementation, we obtain weights of edges based on MulVAL rule metric and CVSS scores [16]. The CVSS scores are available for some well-known vulnerabilities. We can use these scores (if available) for nodes of a MulVAL attack graph that are associated with vulnerabilities. On the other hand, MulVAL provides conditional probability estimates for attack completion given that the precondition of a rule is satisfied. These estimates are based upon security expert input. These two values combined together provide the basis of assigning weights to the edges of forward attack graphs. In Section 3.1, we describe an

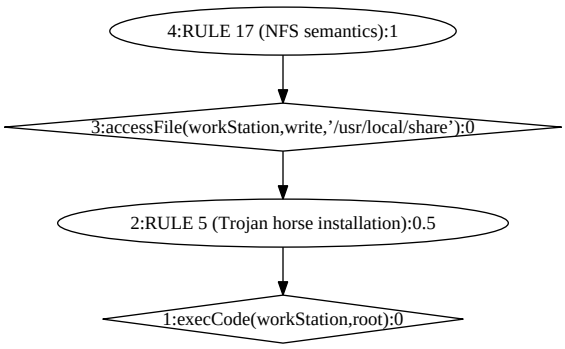

Figure 2.2: *A partial attack graph.*

algorithm that propagates the probabilities throughout the attack graph.

2.5 Personalization Values

In section 2.1 we defined a function $p(v)$ that maps a vertex to its personalization value. We use this function to formalize the idea of *gate nodes* in our computation. A gate node in G_r will have $0 < p(v) \leq 1$, and any other nodes will have $p(v) = 0$. Gate nodes are starting point for the attackers. Intuitively, an attacker has limited choices in terms of where he can start his attack relative to an attack graph. For example, in Figure 2.2, node 1 can never be a starting point for the attacker, because it is his ultimate goal. An attack path within the graph may start from these gate nodes (and no rank has yet been propagated to them). Therefore we need to specify an intrinsic value that helps to establish their risk.

2.6 Damping factor

Based on our attack scenario, an attacker may decide for some reason to stop the attack at any time. This may be due to reaching a level of difficulty that makes an attacker give up and stop expanding the graph further. We formalize this idea into a probability α that an attacker will *not* give up and will continue his attack. The value of α can affect the value of the rank that is computed for v. For example, if α is very low when multiplied with v's

rank, then it relatively lowers the rank. A low α means that it is likely that an attacker will give up at this point. Therefore we will have more difficult and thus less risky node.

2.7 Ranking Equation

Let r_v be the rank of a vertex $v \in V$ given a forward graph G_r. Then, the rank of v is computed as follows:

$$r_v = \alpha \sum_{u \in N^-(v)} (w(u,v) r_u m_u) + (1-\alpha) p_v \tag{2.1}$$

where we have for v:

$$m(u,v) = \begin{cases} 1 & \text{t(v)=OR; } u \in N^-(v) \\ 0 & \text{t(v)=AND; } u \in N^-(v) \backslash u_m \\ 1 & \text{t(v)=AND; for } u_m \end{cases} \tag{2.2}$$

where u_m is the node with with minimum $w(u,v)r_u$ among all incoming values [1]. Equation 2.1 can capture the formalizations described in previous sections. Note that there is a distinction between AND and OR nodes in the way they handle the incoming values from their children. This distinction is due to our interpretation of the ranks. As explained in section 2.2, we imagine attackers who are trying to satisfy vertices over certain paths in G_r to achieve its root. Thus, when we compute the attackers who could reach an OR node, we sum the ranking measure of the incoming attackers. The rationale behind this is that an OR node can be satisfied by any of its children. Thus, if we know that there are multiple attackers who could satisfy children of an OR vertex v, all of them can satisfy v independently. As a result the number of attackers who can achieve v will be the sum of the number of attackers who can achieve any of its children.

In case we are computing the rank for an AND vertex u, we know that to satisfy u, all the children of it must be satisfied. For example suppose u had two children v_1 and v_2 and three attackers could reach v_1 but only 2 attackers could reach v_2. Since both v_1 and v_2 are required for u to be achieved, we can at most (in the worst case) have 2 attackers satisfy u.

[1] In computing this minimum we do not consider any u such that $t(u) = LEAF$ except if we only have leaf nodes.

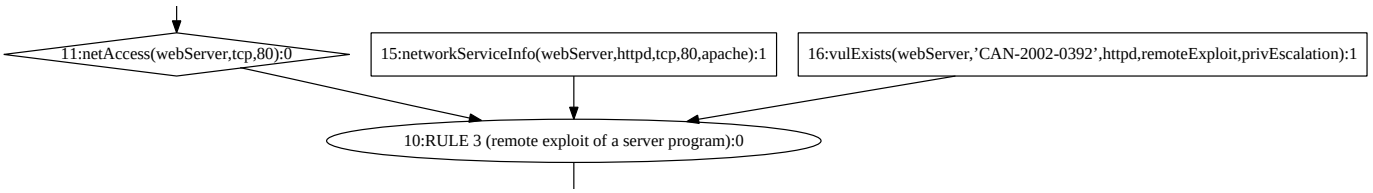

Figure 2.3: *Part of the attack graph with three hosts.*

This is because the third attacker in v_1 didn't prove that it could achieve v_2. Therefore, the attackers on v_1 will need to work together with those of v_2 in order to move further towards u. When computing this behavior, we simply consider the minimum value (computed by equation 2.2) over the incoming ranks of the children of u.

There is one exception when computing the rank for an AND node. In taking the minimum, we simply ignore the SOURCE nodes. To explain the rationale behind this, let us consider the example of Figure 2.3. When computing the rank for node 10, if the value of $w(u, v)r_u$ of nodes 15 or 16 is less than that of node 11, that means we will ignore some of the attackers that could satisfy node 11 by taking the minimum rank only which is not correct in practice. Since node 11 enables the network access and thus the attack to node 10, computation of r_{10} should be affected by all the value of r_{11}. Generally, since these source nodes represent configuration information, they are necessarily true and thus can be ignored in the rank calculation. We have implemented this exception in StepRank to have more meaningful results.

2.8 Computing StepRank

For the purpose of computing the ranks, we define a vector $\vec{r}$ of size $n = |V|$ to store the ranks of all $v \in V$. We also collect all the weights of the edges into a matrix D where we have $D_{vu} = w(u, v)$. If there is no edge from u to v, $D_{vu} = 0$. The personalization values will form the vector $\vec{p}$. The damping factor α will simply be multiplied with the matrix D according to our ranking equation. The scaling value for $\vec{p}$ is given by β.

In order to handle the situation with AND nodes, we form another matrix S with the same size of D. $S_{vu} = 1$ for any OR node u such that $D_{vu} > 0$. $S_{vu} = 0$ for all values of a row corresponding to an AND node u except for the column that has the minimum value of $w(u, v)r_u$ which will have $S_{vu} = 1$. We multiply corresponding elements of S with D to reflect the idea of the distinction between AND and OR nodes. We denote this multiplication by $Mult(S, D)$.

At each iteration t, we compute the minimum for AND nodes as:

$$u_m^t = min\{w(u, v)\vec{r_u}\} \text{ For all } u \in N^-(v) \tag{2.3}$$

The system of equations is given by:

$$\vec{r_t} = \alpha \times Mult(S, D)\vec{r_{t-1}} + (1 - \alpha)\vec{p} \tag{2.4}$$

We iterate over this equation until it converges using StepRank.

Algorithm 1 StepRank$(D, \alpha, S, \vec{p}, \vec{r})$:

```
diff ← +∞
while diff > ε do
    r_i ← ((α * Mult(S, D)) * r) + ((1 − α) * p)
    diff ← ComputeDiff(r_i, r)
    r ← r_{i−1}
    S ← RefineS(S, r, D)
end while
return r
```

In this procedure, for every iteration we refine the matrix S to reflect the minimums for the AND nodes based on the new values of $\vec{r}$. Although we have not yet proved the convergence of StepRank, we have observed its convergence for several attack graphs (refer to Chapter 4).

2.9 Interpretation of StepRank

In our model of ranking, we assume there are a number of actual attackers who wish to compromise a network from whose configuration a forward attack graph G_r is created.

Attackers follow the edges (based on their probability) until they reach the root node in G_r. Attackers could stop crawling the graph at any node. Finally, those nodes that were very difficult to satisfy will receive lower values since a few attackers could achieve them. This way, the rank value of a node represents the relative difficulty to achieve that node and thus how risky it is.

Based on our interpretation, there are general properties that hold for any attack graph and there are some properties that are specific to a particular instance. The following are some general properties of the attack graph which explain our interpretation:

Property 1. For any rule node v_i on a given path towards the root of a forward attack graph, we have $r_{v_i} \leq r_{v_{i-1}}$.

The intuition behind this property is that achieving a rule node means achieving its prerequisites. Thus, reaching a rule node by a group of attackers should be as hard as reaching its prerequisites. If one prerequisite is the hardest to achieve, its rank will somehow dominate others' ranks. This is because of the nature of a rule node that needs all of its predecessors to be satisfied before itself.

Property 2. For any OR node v_i on a given path towards the root of a forward attack graph, it must be as hard as all of the incoming nodes.

For an OR node, the situation is different. For an OR node v only satisfying one node $u \in N^-(v)$ is sufficient to satisfy v. The problem is when there are more than one such nodes, which one should reflect the difficulty of the OR node. In our interpretation, we assume arbitrary group of attackers that may work for achieving the goal. Therefore, it is intuitive that the OR node should reflect the difficulty of all the incoming nodes. Thus, it should be as hard as all of them as if it was only one node.

For example, let E_r be the set of privilege nodes that exist on the easiest path to a privilege node e_r. Then, e_r compared to any $e_i \in E_r$ is the hardest to achieve and thus should receive the lowest rank from StepRank.

Since in our model, we think in terms of groups of attackers that randomly attack the sys-

tem, we always regard every path to e_r as one single path which is distributed in a number of partial paths (OR node). Therefore, $r_{e_r} \leq r_{e_i}$ for all $e_i \in V$.

Consequently the root node receives the least rank among all other privilege nodes in an attack graph. Because every privilege node must occur before the root node in a forward attack graph.

We can conclude from the properties that the value of StepRank is monotonically decreasing on any given path towards the goal node of a forward attack graph.

Chapter 3

Parameter Assignment

In this chapter we shall investigate information sources that are used to assign various parameters of StepRank.

3.1 Weights of the Edges

In a forward attack graph, we may have a situation in which there are multiple outgoing edges from a given node. This situation can only occur for OR nodes which can have choices for the next attack step. In this case, an attacker will have to choose between the available options and intuitively select a path based on the easiness of the immediately next step. It is essential to formalize this fact in the ranking process to affect the flow of the ranks. Edge weights reflect the likelihood of success in the consequent steps, if a particular edge was chosen by the attacker. The weight for an edge is defined as follows.

Let $P(u|v)$ be the likelihood of success of an attacker at u given that v is satisfied. We know that for an OR node, the next step must be one or more AND nodes. Therefore, we can simply pick $P(u|v)$ as the weights of the edge coming to u from an OR node. This choice will directly imply the likelihood of success if the edge to u was chosen.

The value of $P(u|v)$ depends on the graph structure, nature of u and (if available) previous subjective assessments, comprising background knowledge and experience of past successful attacks from security experts. However, while ranking the nodes, we need to consider real attack scenarios. As described earlier, we assume that an attacker can only see one "attack"

step ahead and has no immediate knowledge about future steps in the graph. However, several attack graph nodes may belong to a single attack step in a real scenario. Therefore, we define a real attack step as follows:

Let u_s be an OR node and u_k be a final node so that u_s be a predecessor of u_k. Then, any path consisting of some of the successors of u_s until u_k is considered a real attack step. The final node represents the last node in the successors of u_s in a real attack step. In our implementation we assume that u_k is determined and provided by MulVAL. The propagation of probability in the sequence of nodes in a real attack step is defined as follows:

For any intermediate nodes $s < i < k$:

$$P'(u_i) = Combine(P'(u_j)),\ t(u_i) = OR,\ \forall j \in N^+(u_i) \tag{3.1}$$

$$P'(u_i) = P'(u_{i+1}),\ t(u_i) = AND \tag{3.2}$$

Where *Combine* is a function that based on some criteria, combines the probability along different paths of an OR node which is on a real attack path to be computed. We actually pursue the chain of nodes in T until we have collected all of the attack graph nodes belonging to a real attack step. MulVAL specifies the likelihood for the nodes that do not form a real attack path as 1. That is because they are natural consequences of their predecessors and thus can be satisfied with no doubt. This suggests that we can retrieve the success likelihood from the next node. Therefore, it is convenient to propagate back the success likelihood associated with u_k until we reach the edge that needs this value. Given we have n nodes in the attack graph, the algorithm ComputeProb(M,P) computes the probability values based on the above formulation. The parameters M and P refer to the matrix D in Equation 2.4 and the likelihood metrics provided by MulVAL.

Algorithm 2 ComputeProb(M,P)

```
1: for i ← 0 to n do
2:    if Type(i)=AND then
3:       P[i] ←Propagate(i,M,P)
4:    else if Type(i)=OR then
5:       Propagate(i,M,P)
6:    end if
7: end for
8: return M
```

Algorithm 3 Propagate(i,M,P)

```
1: if Type(i)=AND then
2:    if IsFinal(i) then
3:       return P[i]
4:    else
5:       return Propagate(Child(i),M)
6:    end if
7: else if Type(i)=OR then
8:    for j ← 0 to n do
9:       if M[i, j] > 0 then
10:         M[i, j] ← Propagate(j,M,P)
11:         P[i] ← Combine(P[i], M[i, j]))
12:      end if
13:   end for
14:   return P[i]
15: end if
```

The proposed algorithm can handle multiply-connected graphical models in which different paths going out from one OR node reach the same final point. Recall that we can only have OR nodes with more than one option. This means the different paths describe *independent* attack scenarios that may happen in reality. The independence in this case means that fulfilling the requirements of any node within any of the available path choices does not require presence of any other nodes from other choices. Formally, we can have the following lemmas:

Lemma 1 let T_1 and T_2 represent two different sequences of nodes for two different paths choices out of an OR node u_s assuming an end node u_k common to both paths. Then if T is chosen by an attacker as the next stage for in his attacker path, for all $u_{i1} \in T_1$ may be achieved (based on their difficulty attributes) without achieving any $u_{i2} \in T_2$. Further, participation of the start node of this stage u_s and the end node u_k in any of the two paths is equal to the other.

Lemma 2 For any AND node v in a forward attack graph, $N^+(v) = 1$. We know from Lemma 1 that different paths from an OR node are independent. We can also conclude that different attack paths in an instance of attack graph are independent. Therefore, an occurrence of v on a path is independent of its occurrence on any other path and it enables exactly one OR node; hence $N^+(v) = 1$.

Although we have not formally proved the correctness of the proposed algorithms, these two lemmas are useful to show the general intuition behind them.

It also turns out that, this computation is not affected by the cycles. That is because, we have already included a damping factor that represents the likelihood that an attacker will not stop his attack. Therefore, if a cycle in an attack path towards the goal exists, we assume that with $1-\alpha$ probability the attacker can indeed give up and move to another path. Thus, a cycle in an attack path should not disturb the results of ComputeProb(M,P).

3.2 Damping Factor

In our current implementation of StepRank, we adapt uniform values for the vertex specific damping factors. As described earlier, a damping factor should reflect the likelihood that an attacker will continue his attack. This value can be left as a default value or can be supplied in the system and overridden by the user, because several issues can affect it. First, previous data about successful and unsuccessful attacks can give good impression about the damping value. Despite the strategy in AssetRank, we would not use a vertex specific damping factor in StepRank because separating the damping value for every vertex is not expected to have great effect on the final rank value. There exist some nodes in a MulVAL attack graph that are *natural consequences* of other nodes. That is, as a result of satisfying u, v is considered naturally accomplished without any effort required from the attacker. For example consider the graph of Figure A.1. When an attacker has achieved node 12, having node 11 is just trivial and requires no action from attacker's side. Therefore, if a vertex specific α is used, node 11 should receive $\alpha_{11} = 1$ since the attacker is not expected to stop at this node at all. Thus it is expected that few nodes will have an intrinsic likelihood of attack propagation that requires a node-specific damping factor. Thus, using a single global damping factor does not significantly diminish the overall quality of the results. For this reason, in our experiments we have considered a uniform value of $\alpha_v = 0.8$ for all $v \in V$.

3.3 Personalization Vector

In section 2.5 the personalization vector was introduced to capture the idea of gate nodes. Currently, StepRank identifies all the SOURCE nodes as gate nodes. That is because in order to start by any MulVAL rule in the graph, all the leaves of that node should be assumed satisfied otherwise we cannot start anywhere in the graph. To illustrate this issue consider again the graph of figure A.1. It is possible for an attacker to start by a direct network access (node 12). However, this requires that an attacker exists (node 14) which is the case and the web server (node 13) is running. We have to assume that the web-server

is running to make the attack possible. Thus, we give a personalization value more than 0 to both of the SOURCE nodes. In fact, the real gate nodes would be some rule nodes (e.g. node 12) which will be available to the attacker if its SOURCE nodes are available.

It is possible that some gate nodes have higher potential of being used by actual attackers. One way of capturing this fact is to modify the personalization value and assign a value that actually reflects the likelihood of the gate node being used by attackers. An alternative solution (solution that we have adopted) is to introduce a dummy source node d. This special source node will be added to the graph and recognized as a gate node. For any rule node v node that has all its parents as leaves (which are also gate nodes) we define the edge (d, v). The reason we use only the rule nodes that have leaf parents is that they represent the actual gate nodes of the graph. That is, because they do not require any other attack step, they rather need their parents to be satisfied. By connecting the dummy node this way within the graph, we can represent the likelihood of taking either of rule nodes as a start node by the weight of (d, v).

Finally, to determine the overall effect of personalization values on the ranks, we scale P with $1 - \alpha$.

Chapter 4

Experiments

4.1 Validation Approach

Our approach in validation is to verify the results of StepRank based on properties of specific scenarios. When given a network configuration we first specify the intuitive expectations of our ranking algorithm. Then we run StepRank on the attack graph and verify if indeed it can rank the nodes based on what we have specified as intuitively correct.

4.2 Experiment 1

We run StepRank on the scenario of Figure 1.4. In this scenario, we assume that the attacker is located at host A. The access to machine E is allowed only through machines C and D. In addition, the machine F can be accessed through host E. The ultimate goal is to gain root privileges on host F. All the machines contain a common vulnerability, and we have set the CVSS score for the vulnerability to be of medium risk. In terms of individual risk associated with each machine, the machines B, C, and D are in the same security risk level. This is because, they all have the same vulnerability, and based on the configuration, they have equal chance of being compromised. However, gaining privileges on E is more difficult because of the need for more attack steps for an attacker to achieve it. The same conclusion is applied to F with the difference that F is even more difficult to be compromised because it is hidden behind E. By looking at the results of table 4.1, we can observe the

same expectations by the provided ranks. We can see that the privilege vertex for B, C, and D all received the same rank. It also shows that the privilege vertex of E has a lower rank (and hence lower risk) followed by that of F.

Vertex	StepRank
RULE 7 (direct network access)	0.007979971
RULE 7 (direct network access)	0.007979971
RULE 7 (direct network access)	0.007979971
netAccess(c,tcp,80)	0.007016203
netAccess(b,tcp,80)	0.007016203
netAccess(d,tcp,80)	0.007016203
RULE 3 (remote exploit of a server program)	0.003131014
RULE 3 (remote exploit of a server program)	0.003131014
RULE 3 (remote exploit of a server program)	0.003131014
execCode(c,serviceaccount)	0.002360001
execCode(b,serviceaccount)	0.002360001
execCode(d,serviceaccount)	0.002360001
netAccess(e,tcp,80)	0.000718861
RULE 6 (multi-hop access)	0.000577303
RULE 6 (multi-hop access)	0.000577303
RULE 6 (multi-hop access)	0.000577303
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
RULE 6 (multi-hop access)	0.000432978
execCode(e,serviceaccount)	0.000335305
RULE 3 (remote exploit of a server program)	0.000317116
execCode(f,serviceaccount)	0.000106868

Table 4.1: *Partial results from Experiment 1 for the scenario of figure 1.4.*

4.3 Experiment 2

Consider the example network shown in Figure 4.1 [7]. The database server is isolated on a subnet and only the web server has access to it. Both the web server and the database server host vulnerable software and the web server is accessible via the internet. The user desktop hosts a vulnerability that can be immediately compromised because it is also connected to the internet. We expect that the web server be quite risky because it is directly connected to the internet. We also expect that the database server be ranked lower than the web

server because the attacker needs to gain enough privileges on the web server first to be able to access the database server. The user desktop should have a high risk level and assigned a high risk value because it hosts a vulnerability that can be exploited through the web browser.

Partial results from StepRank are given in Table 4.2. It clearly indicates our expectations about the ranks. We can observe that obtaining privileges on web server has received higher rank and thus has higher risk. Then user desktop has received a rank lower than the web server and so the database server has received lowest rank on its privilege node.

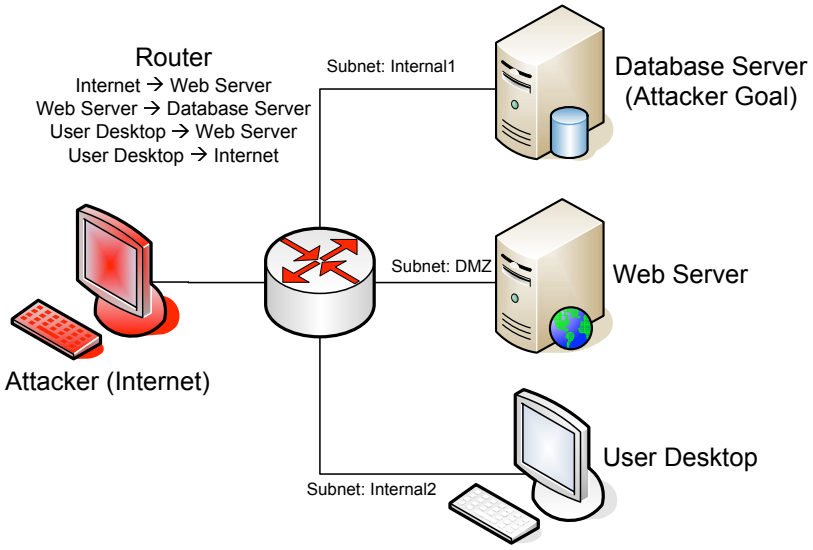

Figure 4.1: *An example network configuration with three subnets*

An important issue that we observed was when we ranked the same graph with using uniform weight for all edges in the graph. This can be done if ComputeProb is not executed prior to executing StepRank. The results of this observation is shown in Table 4.3. The main difference is that in this case user desktop has received higher rank than of web server's rank. Notice that user desktop (as it can appear from the attack graph) can be exploited in a single attack step whereas that is not the case for the web server. Therefore, if the two attack paths for the web server and the user desktop have equal chance of being used by the attacker, obviously user desktop should be more risky than the web server.

Vertex	StepRank
RULE 7 (direct network access)	0.01505546
RULE 13 (incompetent user)	0.01254621
netAccess(webServer,httpProtocol,httpPort)	0.01204677
principalCompromised(joe)	0.00929108
RULE 3 (remote exploit of a server program)	0.00535169
RULE 22 (Browsing a malicious website)	0.00501849
canAccessMaliciousInput(userDesktop)	0.00446433
execCode(webServer,system)	0.00396831
RULE 6 (multi-hop access)	0.00103574
RULE 6 (multi-hop access)	0.00103574
netAccess(databaseServer,tcp,1521)	0.00089747
RULE 23 (Browsing a compromised website)	0.00086312
RULE 4 (remote exploit for a client program)	0.00066093
execCode(userDesktop,joeAccount)	0.00055332
RULE 3 (remote exploit of a server program)	0.00039777
execCode(databaseServer,oracle_user)	0.00029982

Table 4.2: *Partial results from Experiment 2.*

That is because exploiting the user desktop needs less work for the attacker to be done.

4.4 Conclusion

Based upon the interpretation and consequent properties presented in Section 2.9 , and the results obtained from our experiments, we would expect that our algorithm is capable to adequately handle bigger network configuration scenarios. We tried to understand real attack scenarios and the way an attacker plans his attack on the network. By combining different scaling parameters, techniques for personalizing or calibrating their values, and the rank propagation algorithm, we can formalize the incorporation of facts from the real world. It is possible to even add more formalization and parameters to our ranking equation as it can handle more information. We can indeed include additional parameters regarding what can further affect the riskiness of a certain node of an attack graph. We can also incorporate previous data about successful attacks in a local environment or a similar network. This sort of data can be injected into a learning engine to produce rules that might contribute to making the ranks more accurate.

Vertex	StepRank
RULE 0 (Insider threat)	0.02310025
RULE 13 (incompetent user)	0.02310025
RULE 23 (Browsing a malicious website)	0.02310025
RULE 7 (direct network access)	0.02310025
execCode(userDesktop,joeAccount)	0.02203929
canAccessMaliciousInput(userDesktop)	0.0175706
netAccess(webServer,httpProtocol,httpPort)	0.0175706
principalCompromised(joe)	0.01439062
RULE 4 (remote exploit for a client program)	0.01094604
RULE 3 (remote exploit of a server program)	0.01094604
execCode(webServer,system)	0.00681885
RULE 6 (multi-hop access)	0.00343241
RULE 9 (...)	0.00343241
RULE 24 (Browsing a compromised website)	0.00343241
RULE 6 (multi-hop access)	0.00343241
netAccess(databaseServer,tcp,1521)	0.00317998
canAccessHost(userDesktop)	0.00213826
RULE 3 (remote exploit of a server program)	0.00198119
RULE 24 (Browsing a compromised website)	0.00141596
RULE 6 (multi-hop access)	0.00141596
RULE 6 (multi-hop access)	0.00141596
RULE 1 (...)	0.00133212
execCode(databaseServer,oracle_user)	0.00123408

Table 4.3: *Partial results from Experiment 2 without probability propagation.*

The specification of damping factor discussed in 3.2 can become dynamic. In our current view, this value is fixed and pre-defined before the computation of ranks begins. This means that we have estimated the likelihood that, based on available data, an attacker will not stop his attack. It can be seen that this expectation may become more "accurate" if it was involved with the actual rank that we are computing. The ranking is supposed to provide a measurement about how risky a node is. This ranking is partially dependent on the value of the damping factor. Thus, if we have computed the rank for some of the nodes on an attack graph, we can enrich our knowledge about the difficulty of various attack paths on this graph by considering the ranks currently computed. This consideration is, of course based on a set of ranks whose computation is completed. Therefore, it is convenient to somehow map the knowledge gained from current computation to a modification in the value of the damping factor. Thus, we can refine the value of α as we propagate the rank.

Bibliography

[1] K. Ingols, R. Lippmann, and K. Piwowarski, *Practical Attack Graph Generation for Network Defense*.

[2] X. Ou, W. F. Boyer, and M. A. McQueen, *A Scalable Approach to Attack Graph Generation*, 13th ACM Conference on Computer and Communications Security (CCS 2006), 2006.

[3] 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*, pages 254–265, 2002.

[4] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, *Computer-Attack Graph Generation Tool*, DARPA Information Survivability Conference & Exposition II, 2001.

[5] S. Jajodia, S. Noel, and B. O'Berry, Topological analysis of network attack vulnerability, in *Managing Cyber Threats: Issues, Approaches and Challanges*, edited by V. Kumar, J. Srivastava, and A. Lazarevic, chapter 5, Kluwer Academic Publisher, 2003.

[6] X. Ou, S. Govindavajhala, and A. W. Appel, *MulVAL: A Logic-based Network Security Analyzer*, 14th USENIX Security Symposium, 2005.

[7] R. Sawilla and X. Ou, Identifying critical attack assets in dependency attack graphs, in *13th European Symposium on Research in Computer Security (ESORICS)*, Malaga, Spain, 2008.

[8] L. Page, S. Brin, R. Motwani, and T. Winograd, The pagerank citation ranking:

Bringing order to the web, Technical report, Stanford Digital Library Technologies Project, 1999.

[9] National vulnerability database version 2.2, http://nvd.nist.gov/, 2008.

[10] S. Ceri, G. Gottlob, and L. Tanca, *What You Always Wanted to Know About Datalog (And Never Dared to Ask)*, IEEE Transactions on Knowledge and Data Engineering, 1989.

[11] F. Balmas, *Displaying dependence graphs: a hierarchical approach*, Workshop on Analysis, Slicing and Transformation, 2001.

[12] M. S. Ahmed, E. Al-Shaer, and L. Khan, Dynamic risk measurement and mitigation for proactive security configuration management, Technical report, DePaul University, 2007.

[13] V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing, Ranking attack graphs, in *Proceedings of Recent Advances in Intrusion Detection (RAID)*, 2006.

[14] S. OHare, S. Noel, and K. Prole, *A Graph-Theoretic Visualization Approach to Network Risk Analysis*, 5th International Workshop on Visualization for Cyber Security, 2008.

[15] LingyuWang, T. Islam, T. Long, A. Singhal2, and S. Jajodia, An attack graph-based probabilistic security metric, in *Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08)*, pages 283–296, 2008.

[16] Common vulnerability scoring system (cvss), http://first.org/cvss/cvss-guide.html, 2008.

Appendix A

Attack Graphs

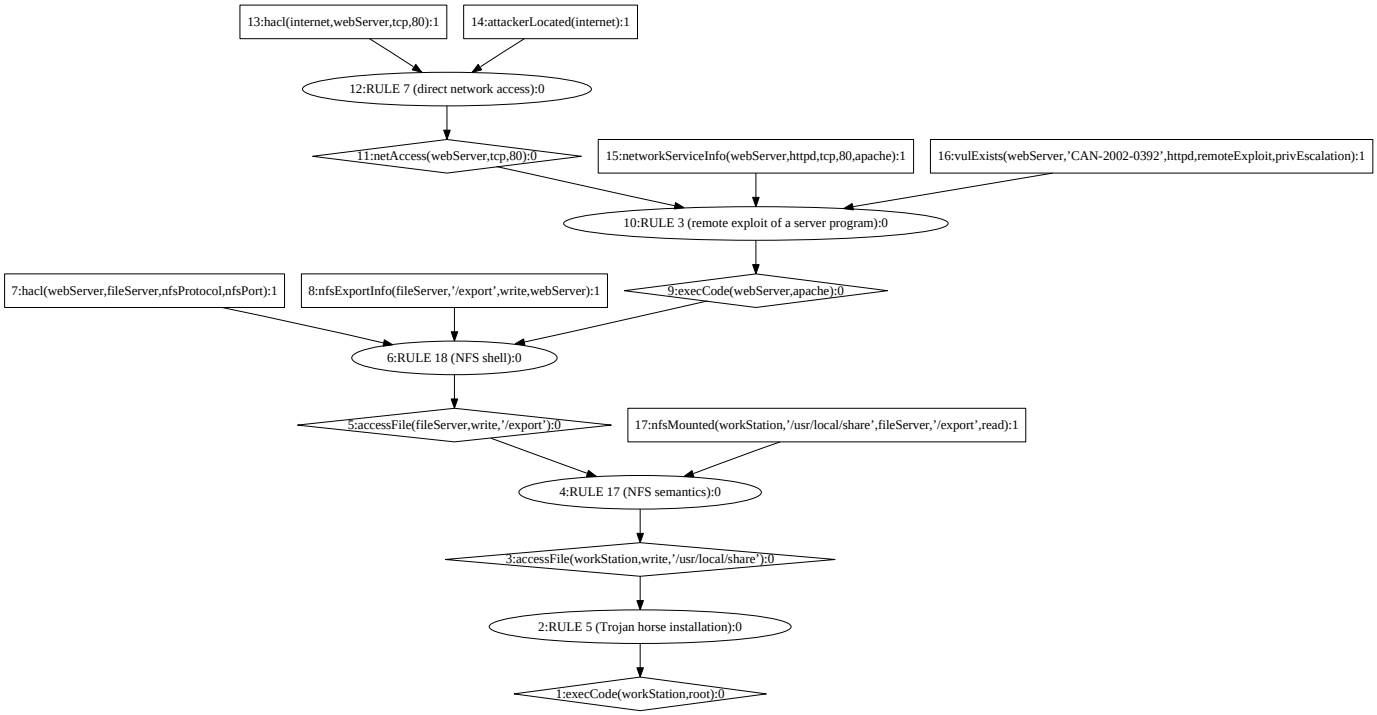

Figure A.1: *An attack graph with 3 hosts*

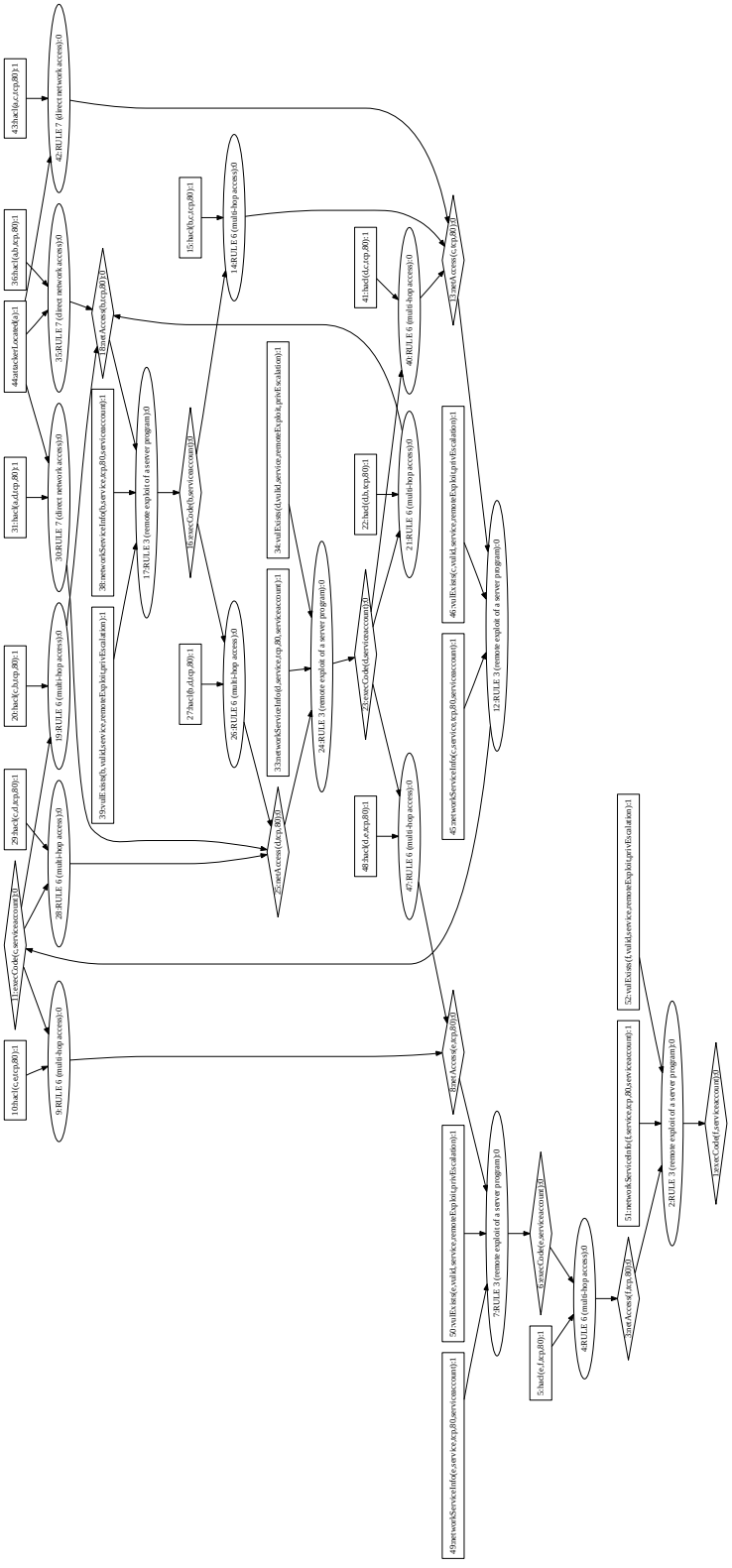

Figure A.2: *An attack graph with 6 hosts*