Googling Attack Graphs

Reginald Sawilla
Defence R&D Canada – Ottawa

Xinming Ou
Kansas State University

The work in this report was completed in May 2007 and formally published September 2007.

Defence R&D Canada – Ottawa
Technical Memorandum
DRDC Ottawa TM 2007-205
September 2007

Principal Author

Original signed by Reginald Sawilla and Xinming Ou

Reginald Sawilla and Xinming Ou

Approved by

Original signed by Julie Lefebvre

Julie Lefebvre
Head/NIO Section

Approved for release by

Original signed by Pierre Lavoie

Pierre Lavoie
Head/Document Review Panel

Abstract

Attack graphs have been proposed as useful tools for analyzing security vulnerabilities in network systems. Even when they are produced efficiently, the size and complexity of attack graphs often prevent a human from fully comprehending the information conveyed. A distillation of this overwhelming amount of information is crucial to aid network administrators in efficiently allocating scarce human and financial resources. This paper introduces the AssetRank algorithm, a generalization of Google's PageRank algorithm that ranks web pages in web graphs. AssetRank handles the semantics of dependency attack graphs and assigns a metric to the vertices, which represent network privileges and vulnerabilities, indicating their importance in attacks against the system. We give a stochastic interpretation of the computed values in the context of dependency attack graphs, and conduct experiments on various network scenarios. The results of the experiments show that the numeric ranks given by our algorithm are consistent with the intuitive importance that the privileges and vulnerabilities have to an attacker. The asset ranks can be used to prioritize countermeasures, help a human reader to better comprehend security problems, and provide input to further security analysis tools.

Résumé

On a proposé des graphes d'attaque comme outils utiles dans l'analyse des vulnérabilités de sécurité dans les réseaux informatiques. Même lorsqu'ils sont produits de façon efficiente, la taille et la complexité des graphes d'attaque empêchent souvent un être humain de bien saisir toute l'information ainsi présentée. Il est essentiel de distiller cette masse écrasante d'information pour aider les administrateurs de réseau à allouer de façon efficiente les ressources humaines et financières limitées. Dans ce document, on présente l'algorithme AssetRank, une généralisation de l'algorithme PageRank de Google qui sert à classer les pages Web dans des graphes Web. AssetRank traite la sémantique des graphes d'attaque à dépendances et il attribue une mesure aux sommets, qui représentent les privilèges et les vulnérabilités des réseaux, indiquant leur importance dans des attaques contre le système. Nous donnons une interprétation stochastique des valeurs calculées dans le contexte des graphes d'attaque à dépendances et nous menons des expériences sur différents scénarios de réseau. Les résultats des expériences montrent que le classement numérique donné par notre algorithme correspond à l'importance intuitive qu'ont les privilèges et les vulnérabilités pour un attaquant. Le classement des biens peut être utilisé pour établir l'ordre de priorité des contre-mesures, aider un lecteur humain à mieux cerner les problèmes de sécurité et fournir des données d'entrées pour d'autres outils d'analyse de la sécurité.

This page intentionally left blank.

Executive summary

Googling Attack Graphs

Reginald Sawilla, Xinming Ou; DRDC Ottawa TM 2007-205; Defence R&D Canada – Ottawa; September 2007.

Background: An attack graph is a mathematical abstraction of the details of possible attacks against a specific network. Recent advances have enabled computing attack graphs for networks with thousands of machines. Even when attack graphs can be efficiently computed, the resulting size and complexity of the graphs is still too large for a human to fully comprehend. While a user will quickly understand that attackers can penetrate the network it is essentially impossible to know which privileges and vulnerabilities are the most important to the attackers' success. Network administrators require a tool which can distill the overwhelming amount of information into a list of priorities that will help them to efficiently utilize scarce human and financial resources.

Principal results: This paper presents an approach which can automatically digest the dependency relations in an attack graph and compute the relative importance of graph vertices as a numeric metric. The metric gauges the degree to which attackers depend upon a privilege or vulnerability in their attacks. Our algorithm is based on the Google PageRank algorithm which ranks the importance of web pages. The extended algorithm is called AssetRank, and the value it computes indicates the value of an attack asset (a graph vertex) to a potential attacker.

Significance of results: The results of our experiments indicate that the vertex ranks computed by our algorithm are consistent, from a security point of view, with the relative importance of the attack assets to an attacker. The asset ranks can be used to prioritize countermeasures, help a human reader to better comprehend security problems, and provide input to further security analysis tools. The ranks are affected by the specific assets an attacker wishes to obtain (and a system administrator desires to protect).

Future work: We would like to explore how to incorporate business priorities and implementation costs into the rank value, so that the resulting ranks can be used immediately by a system administrator to generate a course of action or automatically implement security hardening measures. We would also like to conduct experiments on operational networks to better understand the advantages and limitations of our proposed algorithm, along with ways of improving it. Finally, we wish to see how the values for arc and vertex weights, representing diverse preferences, can be combined and, similarly, how AssetRanks in various contexts may be combined.

Sommaire

Googling Attack Graphs

Reginald Sawilla, Xinming Ou ; DRDC Ottawa TM 2007-205 ; R & D pour la défense Canada – Ottawa ; septembre 2007.

Contexte : Un graphe d'attaque est une abstraction mathématique des détails d'attaques possibles contre un réseau particulier. Des progrès récents ont permis la création de graphes d'attaque pour des réseaux qui comptent des milliers d'ordinateurs. Même lorsque les graphes d'attaque peuvent être calculés de façon efficiente, la taille et la complexité des graphes ainsi obtenus sont trop grandes pour qu'un être humain puisse comprendre pleinement l'information que contiennent ces graphes. Un utilisateur peut comprendre rapidement que des attaquants peuvent pénétrer dans le réseau, il est essentiellement impossible de savoir quels sont les privilèges et les vulnérabilités qui ont le plus d'importance pour les attaquants. Les administrateurs de réseau ont besoin d'un outil qui peut distiller la masse écrasante d'informations de façon à créer une liste de priorités qui les aidera à utiliser de façon efficiente les ressources humaines et financières limitées.

Principaux résultats : Ce document présente une approche qui peut digérer automatiquement les relations de dépendance dans un graphe d'attaque et calculer l'importance relative de chaque sommet sous forme de mesure numérique. La mesure numérique évalue à quel point les attaquants dépendent d'un privilège ou d'une vulnérabilité pour lancer leurs attaques. Notre algorithme est basé sur l'algorithme PageRank de Google, qui classe l'importance de pages Web. L'algorithme étendu s'appelle AssetRank et la valeur qu'il calcule indique la valeur d'un élément d'attaque (un sommet du graphe) pour un attaquant potentiel.

Importance des résultats : Les résultats de nos expériences indiquent que le classement des sommets calculé par notre algorithme correspond, du point de vue de la sécurité, à l'importance relative des éléments d'attaque pour un attaquant. Le classement des éléments peut être utilisé pour établir l'ordre de priorité des contre-mesures, aider un lecteur humain à mieux cerner les problèmes de sécurité et fournir des données d'entrée pour d'autres outils d'analyse de la sécurité. Le classement varie selon les biens particuliers qu'un attaquant veut obtenir (et le désir de l'administrateur du système de protéger ces biens).

Travaux futurs : Nous aimerions explorer les façons d'incorporer des priorités opérationnelles et les cots de mises en œuvre dans le calcul du classement, de sorte qu'un administrateur de système pourrait utiliser le classement ainsi obtenu pour générer une marche à suivre ou mettre en œuvre automatiquement des mesures de durcissement de la sécurité. Nous aimerions aussi mener des expériences sur des réseaux opérationnels pour mieux

comprendre les avantages et les limites de notre algorithme proposé, et trouver des façons de l'améliorer. Enfin, nous souhaitons voir comment il serait possible de combiner les valeurs des facteurs de pondération pour les arcs et les sommets, qui représentent différentes préférences, et, dans le même ordre d'idées, comment il serait possible de combiner Asset-Ranks dans différents contextes.

This page intentionally left blank.

Table of contents

Abstract i

Résumé i

Executive summary iii

Sommaire iv

Table of contents vii

List of figures viii

List of tables ix

1 Introduction 1

2 Attack Graphs 4

3 AssetRank 5

4 Interpretation of AssetRank 10

5 Experiments 12

6 Discussion 18

7 Related Work 20

8 Conclusion and Future Work 22

References 23

List of figures

Figure 1: An example network 1

Figure 2: Attack graph for the network in Figure 1 3

Figure 3: Vertices and arcs in a dependency attack graph 5

Figure 4: Value flows to dependencies 6

Figure 5: An example AND/OR dependency graph 8

Figure 6: Experiment 2, Scenario 1 15

Figure 7: Attack graph for the Figure 6 network 16

Figure 8: Experiment 2, Scenario 2 17

Figure 9: Attack graph for the Figure 8 network 18

List of tables

Table 1: AssetRanks with only OR vertices 9

Table 2: AssetRanks with AND and OR vertices 10

Table 3: AssetRanks for the Figure 1 network 13

Table 4: AssetRanks for the Figure 6 network 15

Table 5: AssetRanks for the Figure 8 network 19

This page intentionally left blank.

1 Introduction

An attack graph is a mathematical abstraction of the details of possible attacks against a specific network. Various forms of attack graphs have been proposed for analyzing the security of enterprise networks [1, 2, 3, 4, 5, 6]. Recent advances have enabled computing attack graphs for networks with thousands of machines [2, 4]. Even when attack graphs can be efficiently computed, the resulting size and complexity of the graphs is still too large for a human to fully comprehend, as has been pointed out by Noel and Jajodia [7]. While a user will quickly understand that attackers can penetrate the network it is essentially impossible to know which privileges and vulnerabilities are the most important to the attackers' success. Network administrators require a tool which can distill the overwhelming amount of information into a list of priorities that will help them to efficiently utilize scarce human and financial resources.

The problem of information overload can occur even for small-sized networks. The example network shown in Figure 1 is from recent work by Ingols *et al.* [2]. Machine A is an attacker's launch pad (for example, the Internet). Machines B, C, and D are located in the left subnet and machines E and F are in the right subnet. The firewall FW controls the network traffic between the two subnets such that the only allowed network access is from C to E and from D to E. All of the machines have a remotely exploitable vulnerability.

We applied the MulVAL attack graph tool suite [4] to the example network. Figure 2 presents a visualization of the resulting attack graph containing 50 vertices and 56 arcs. Even for this simple scenario it is clear that the attack graph is large, cumbersome, and difficult to read without magnification. Essentially, the software vulnerabilities on hosts C and D will enable an attacker from A to gain local privileges on the victim machines, and use them as stepping stones to penetrate the firewall, which only allows through traffic from C and D. In this example, all the machines can potentially be compromised by the attacker, and all the vulnerabilities on the hosts can play a role in those potential attack paths. However, the vulnerabilities on C and D, and the potential compromise of those

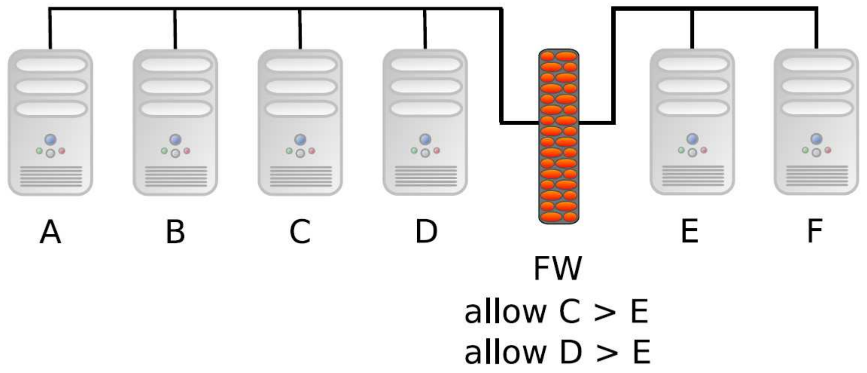

Figure 1: *An example network*

two machines, are crucial for the attacker to successfully penetrate into the right subnet, presumably a more sensitive zone. The attack graph produced by MulVAL does reflect this dependency, but a careful reading of the graph is necessary to understand which graph vertices are the most important to consider. When the network size grows and attack paths become more complicated, it is insurmountably difficult for a human to digest all the dependency relations in the attack graph and identify key problems in a reasonable amount of time.

In order to make the information presented by attack graphs more usable, further analysis is necessary to give guidance in system administration. Previous work has proposed various analyses for attack graphs [3, 8]; however, we observe that those analyses typically treat all the vertices and edges in an attack graph equally. Usually vertices in an attack graph represent potential privileges attackers can obtain or vulnerabilities they can exploit. Those privileges and vulnerabilities are not equal from a security point of view. Some of them are more crucial for the attacker because they are essential to the success of numerous or critical attack paths and thus more attention should be paid to them. In the above example, the vulnerabilities and privileges on machines C and D are more important than those on machine B since they enable the attacker to penetrate the firewall to reach the internal subnet. It would be useful if an automatic algorithm could rank the vertices in an attack graph based on their relative importance to an attacker. The ranking of the vertices would bring the fundamental threats to an administrator's attention, for example, by trimming the attack graph based on the numeric ranks of vertices. It could also help to prioritize mitigation measures based on the criticality of privileges from an attacker's point of view. In reality one cannot always eliminate all security threats. Understanding the relative importance of vulnerabilities is important in deciding upon the best course of action.

This paper presents an approach which can automatically digest the dependency relations in an attack graph and compute the relative importance of graph vertices as a numeric metric. The metric gauges the degree to which attackers depend upon a privilege or vulnerability in their attacks. Our algorithm is based on the Google PageRank algorithm [9] which ranks the importance of web pages. We adapted the PageRank algorithm so that it suits the semantics of dependency attack graphs — a type of attack graph whose edges represent dependencies among attackers' potential privileges. The extended algorithm is called AssetRank, and the value it computes indicates the value of an attack asset (a graph vertex) to a potential attacker. Attack assets consist of privileges, such as the ability to execute code on a particular machine, and facts, such as the existence of vulnerable software. We give a stochastic interpretation of AssetRank in the context of network attacks and conduct experiments on various network settings. The results of our experiments show that the vertex ranks computed by our algorithm are consistent, from a security point of view, with the relative importance of the attack assets to an attacker. The asset ranks can be used to prioritize countermeasures, help a human reader to better comprehend security problems, and provide input to further security analysis tools. The ranks are affected by the specific assets an attacker wishes to obtain (and a system administrator desires to protect). It is

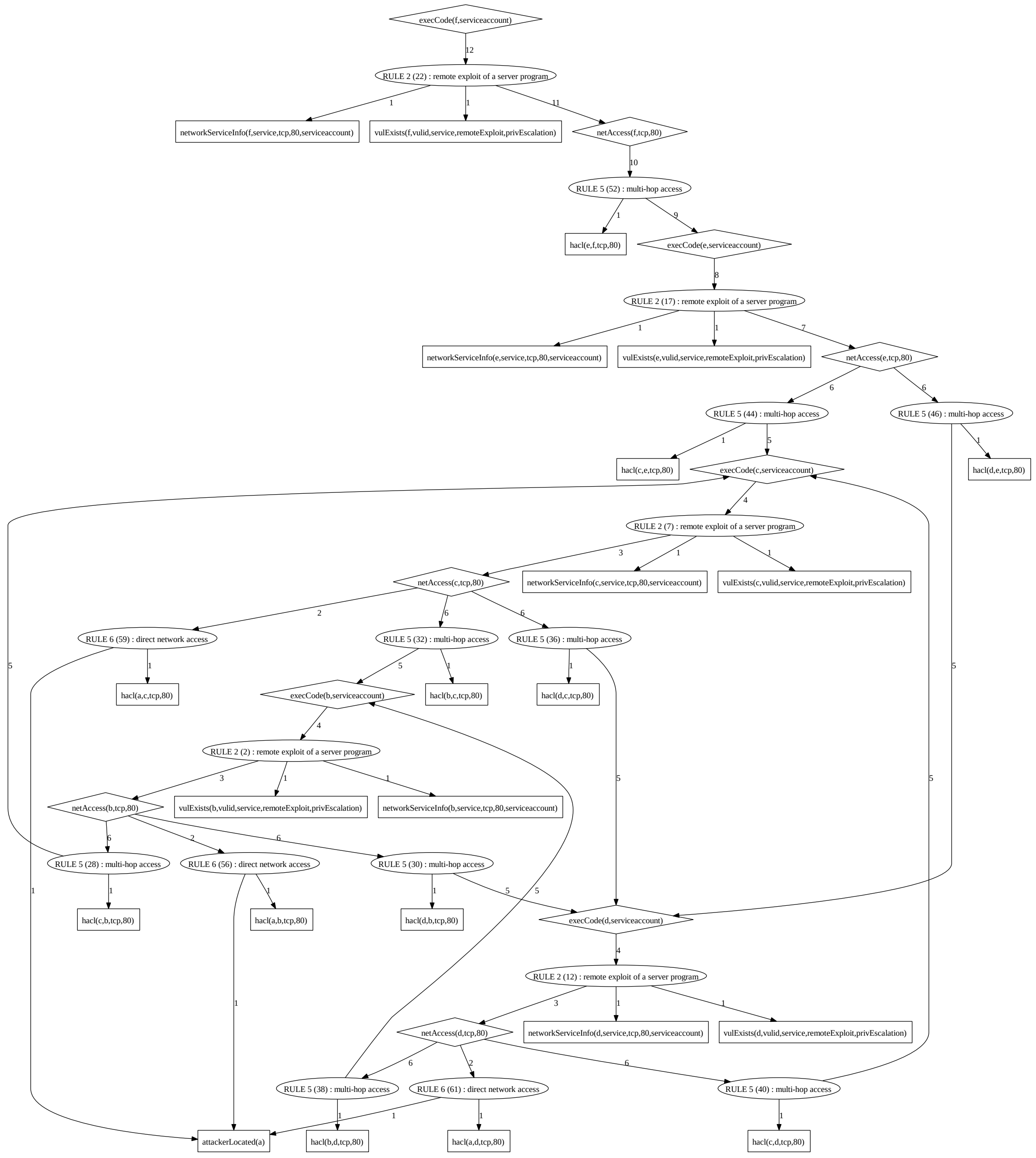
execCode(f,serviceaccount)
RULE 2 (22) : remote exploit of a server program
networkServiceInfo(f,service,tcp,80,serviceaccount)
vulExists(f,vulid,service,remoteExploit,privEscalation)
netAccess(f,tcp,80)
RULE 5 (52) : multi-hop access
hacl(e,f,tcp,80)
execCode(e,serviceaccount)
RULE 2 (17) : remote exploit of a server program
networkServiceInfo(e,service,tcp,80,serviceaccount)
vulExists(e,vulid,service,remoteExploit,privEscalation)
netAccess(e,tcp,80)
RULE 5 (44) : multi-hop access
RULE 5 (46) : multi-hop access
hacl(c,e,tcp,80)
execCode(c,serviceaccount)
hacl(d,e,tcp,80)
RULE 2 (7) : remote exploit of a server program
netAccess(c,tcp,80)
networkServiceInfo(c,service,tcp,80,serviceaccount)
vulExists(c,vulid,service,remoteExploit,privEscalation)
RULE 6 (59) : direct network access
RULE 5 (32) : multi-hop access
RULE 5 (36) : multi-hop access
hacl(a,c,tcp,80)
execCode(b,serviceaccount)
hacl(b,c,tcp,80)
hacl(d,c,tcp,80)
RULE 2 (2) : remote exploit of a server program
netAccess(b,tcp,80)
vulExists(b,vulid,service,remoteExploit,privEscalation)
networkServiceInfo(b,service,tcp,80,serviceaccount)
RULE 5 (28) : multi-hop access
RULE 6 (56) : direct network access
RULE 5 (30) : multi-hop access
hacl(c,b,tcp,80)
hacl(a,b,tcp,80)
hacl(d,b,tcp,80)
execCode(d,serviceaccount)
RULE 2 (12) : remote exploit of a server program
netAccess(d,tcp,80)
networkServiceInfo(d,service,tcp,80,serviceaccount)
vulExists(d,vulid,service,remoteExploit,privEscalation)
RULE 5 (38) : multi-hop access
RULE 6 (61) : direct network access
RULE 5 (40) : multi-hop access
attackerLocated(a)
hacl(b,d,tcp,80)
hacl(a,d,tcp,80)
hacl(c,d,tcp,80)

Figure 2: *Attack graph for the network in Figure 1*

similar to Google which presents a user with an ordered list of web pages based upon the structure of the World Wide Web and their relevance to the user's search terms. AssetRank presents a user with an ordered list of attack assets based upon the structure of the attack graph and their relevance to an attacker's goal.

2 Attack Graphs

There are basically two types of attack graphs. In the first type, each vertex represents the *entire* network state and the arcs represent state transitions caused by an attacker's actions. Examples are Sheyner's scenario graph based on model checking [10], and the attack graph in Swiler and Phillips' work [11]. This type of attack graph is sometimes called a *state enumeration attack graph* [7]. In the second type of attack graph, a vertex does not represent the entire state of a system but rather a system condition in some form of logical sentence. The arcs in these graphs represent the causality relations between the system conditions. We call this type of attack graph a *dependency attack graph*. Examples are the graph structure used by Ammann *et al.* [1], the *exploit dependency graphs* defined by Noel *et al.* [3, 7], the MulVAL *logical attack graph* by Ou *et al.* [4], and the *multiple-prerequisite graphs* by Ingols *et al.* [2].

The key difference between the two types of attack graph lies in the semantics of their vertices. While each vertex in a state enumeration attack graph encodes all the conditions in the network, a vertex in a dependency attack graph encodes a single attack asset of the network. A path $s_1 \rightarrow s_2 \rightarrow s_3$ in a state enumeration attack graph means that the system's state can be transitioned from s_1 to s_2 and then to s_3 by an attacker. But the condition that enables the transition $s_2 \rightarrow s_3$ may have already become true in a previous state, say s_1. The reason the attacker can get to state s_3 is encoded in some state variables in s_2, but the arcs in the graph do not directly show where these conditions were first enabled. In a dependency attack graph, however, the dependency relations among various assets are directly represented by the arcs.

For example, Figure 3 is a simple dependency attack graph. The vertices $p_1, ..., p_5$ are assets to an attacker and e_1, e_2 are exploits an attacker can launch to gain privileges. The arcs from a vertex in a dependency attack graph can form one of two logical relations: OR or AND. An OR vertex represents conditions which may be enabled by any one of its out-neighbours. An AND vertex represents an exploit in the attack graph requiring all of the preconditions represented by its out-neighbours to be met. In our figures we use diamonds to symbolize OR vertices, ellipses to symbolize AND vertices, and boxes for sink vertices where there is no out-neighbour. The dependency attack graph in Figure 3 shows that attackers can gain privilege p_5 through one of two ways. They can launch exploit e_1 if all of the conditions p_1, p_2 and p_3 are true. Or they can launch exploit e_2 if conditions p_3 and p_4 are true. Each of the conditions $p_1, ..., p_4$ could be some other privilege the attackers need to gain first, or some configuration information such as the existence of a software vulnerability on a host.

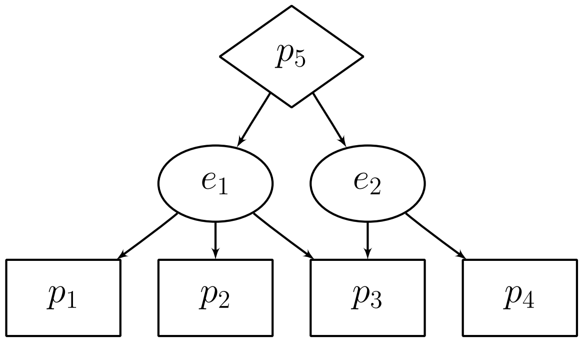

Figure 3: *Vertices and arcs in a dependency attack graph*

In this paper we have chosen to use dependency attack graphs. Our goal is to compute a numeric value representing the importance of each attack asset to an attacker and as such the semantics of dependency attack graphs are better suited for this purpose. Intuitively, the more a vertex is depended upon, the more important it is to an attacker. This is analogous to PageRank's use in the World Wide Web where the more the web depends upon a page (evidenced by links to it) the more important the page is.

3 AssetRank

Internet web pages are represented in a directed graph sometimes called a *web graph*. The vertices of the graph are web pages and the arcs are URL links. The PageRank algorithm [9] computes a page's rank not based on its content, but on the link structures of the web graph. Pages that are pointed to by many pages or by a few important pages have higher ranks than pages that are pointed to by a few unimportant pages. In this paper, we introduce AssetRank, a generalized PageRank algorithm, to handle the various semantics of vertices and arcs that may appear in an attack graph. Most importantly, the modifications allow AssetRank to treat the AND and OR vertices in an attack graph correctly based on their logical meanings. We also incorporate arc and vertex weights so that we can use the weights as input parameters to express the attacker's targets, the desirability of an attacking action, and other relevant information that could affect the ranking of vertices. The AssetRank algorithm could be applied to any graph whose arcs represent some type of dependency relation between vertices [12]. In fact, web graphs can be viewed as a special case of dependency graphs since a web page's functionality in part depends on the pages it links to. The original PageRank algorithm can be seen then as a special case of the AssetRank algorithm where all the vertices are OR vertices and all the vertices and arcs have the same weight.[1] In this section we introduce the AssetRank algorithm in the context of dependency attack graphs.

A dependency attack graph G can be represented as $G = (V, A, f, g, h)$ where V is a set of vertices; A is a set of arcs represented as (u, v), meaning that vertex u depends on vertex v; f is a mapping of non-negative weights to vertices where at least one vertex weight must be positive; g is a mapping of positive weights to arcs; and h is a mapping of vertices to their type (AND or OR). The *out-neighbourhood* of a vertex v, denoted $N^+(v)$, and *in-neighbourhood* of v, denoted $N^-(v)$, are the following two sets.

$$N^+(v) = \{w \in V : (v, w) \in A\} \tag{1}$$
$$N^-(v) = \{u \in V : (u, v) \in A\} \ . \tag{2}$$

The cardinality of a set X is denoted $|X|$ and its L1-norm is denoted $||X||_1$. Without loss of generality, we require the set of all vertex weights $f(V)$ to sum to 1, and the sum of arc weights of a vertex v to be

$$\sum_{w \in N^+(v)} g(v, w) = \begin{cases} 1, & \text{if } h(v) = \text{OR} \\ |N^+(v)|, & \text{if } h(v) = \text{AND} \end{cases} \tag{3}$$

except when v is a sink in which case the sum will be 0.

The intuition behind AssetRank is that each vertex is associated with a value that is a numeric representation of its importance. Part of this value comes from the vertex itself, and part of it comes from other vertices that depend upon it. We can imagine that a portion of a vertex's value "flows" to its out-neighbours, which are vertices it depends upon. Conversely, a vertex receives value from its in-neighbours, which are its dependents. The flow of value is depicted in Figure 4 where the colour of the vertex indicates its value (darker vertices have higher value). Vertices v_1 and v_2 in the left cluster have an average value. Since they both depend on v_3, some of their value is transferred to it and so v_3's value is increased. The vertex v_6 in the right cluster has a higher value than v_3 since one of its dependents, v_4, has a high value. If we imagine $v_1, ..., v_6$ to be an attacker's privileges, then privilege v_6 is more important to the attacker than v_3 because v_6 can enable v_4, which is very important to the attacker.

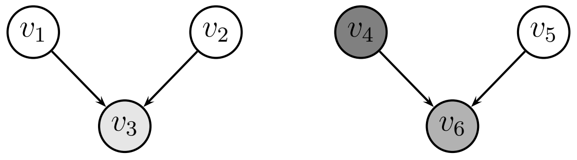

Figure 4: *Value flows to dependencies*

[1]We assume the current ranking algorithm used by Google is much more complicated than the original PageRank algorithm and may have features in common with our AssetRank algorithm.

It is insufficient though to consider only the dependency relations in determining a vertex's value. For example, if v_4 represents the privilege of "execute arbitrary code as root on a database server," and v_1 is the privilege of "execute arbitrary code on a user machine," then v_4 is likely to be more important to both administrators and attackers than v_1. We use vertex weights as an *intrinsic value* to represent a vertex's inherent value. Thus, the value of a vertex consists of two parts: its intrinsic value, and the value that flows to it from its dependents. For the sake of simplicity we assume for the moment that all vertices in the graph are OR vertices. The following equation computes the rank x_v of a vertex v:

$$x_v = \delta \sum_{u \in N^-(v)} g(u,v) x_u + (1-\delta) f(v) \tag{4}$$

The variable δ is called the *damping factor* and its purpose is to set the contribution ratio of a vertex's in-neighbours' rank values versus its intrinsic value to the vertex's final rank value x_v. The intrinsic value of vertex v is given by $f(v)$; in the original PageRank algorithm, $f(v) = 1/|V|$. Each of v's dependents u contributes a portion of its value to v's value. In the original PageRank algorithm, the portion is equally distributed over all the out neighbours of a vertex, that is, $g(u,v) = 1/|N^+(u)|$. In the AssetRank algorithm, $g(u,v)$ could be any value in the range $(0,1]$. This is useful in practice since a vertex may not depend on all its out neighbours equally and the portion $g(u,v)$ can indicate how much vertex u depends on v compared with its other enablers. We assemble the x_v's into a vector X and the $g(u,v)$'s into a weighted adjacency matrix D such that $D_{vu} = g(u,v)$.[2] If $(u,v) \notin A$ then $g(u,v) = 0$. We also put all the vertices' intrinsic values into an intrinsic-value vector $IV = f(V)$. Then the ranks of all the vertices is the solution to the following linear system.

$$X = \delta D X + (1-\delta) IV \tag{5}$$

The above linear system can be solved using the Jacobi method by iterating the following sequence.

$$X_t = \delta D X_{t-1} + (1-\delta) IV \tag{6}$$

It has been shown [13] that for the type of transition matrix D we have discussed so far, such sequences always converge to the solution of the linear system (5) for any initial value X_0, provided that $0 \leq \delta < 1$. We use $X_0 = IV$.

[2]By abuse of notation we use u and v in D_{vu} to represent the column and row indices corresponding to the respective vertices.

Up to this point, we have limited our discussion to OR-vertex graphs where each vertex can be satisfied by any of its out-neighbours. Many graphs arise in practice (in particular, dependency attack graphs) which also have AND vertices. An AND vertex depends on *all* of its out-neighbours. For example, an OR-vertex would be used to represent an attacker gaining network access to a host by any one of five connected hosts. An AND vertex would be used to represent an attacker exploiting a vulnerability that requires the combination of account access and a vulnerable program.

Since any of an OR vertex's out-neighbours can enable it, they will split its value, as expressed in (3) and (4). As the number of out-neighbours increases, the importance of each out-neighbour decreases since the vertex can be satisfied by any one of them. This reduced dependency is not true of AND vertices. Since all the out-neighbours of an AND vertex are necessary to enable it, it is intuitively incorrect to lessen the amount of value flowed to each out-neighbour as their number grows. This can be seen best in an example.

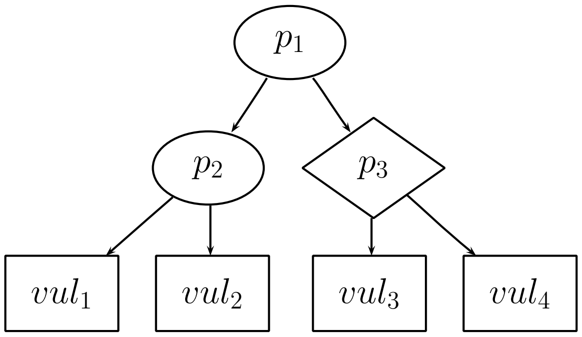

Figure 5: *An example AND/OR dependency graph*

In the dependency graph shown in Figure 5, attackers realizing the goal p_1 depend upon their ability to obtain both privileges p_2 and p_3. p_2 is an AND vertex and it requires two vulnerabilities vul_1 and vul_2. p_3 is an OR vertex and it requires only one of either vul_3 or vul_4. In this example we assume all the arcs have the same weight. If one were to rank the importance of the four vulnerabilities, it is logical that the ranks for vul_1 and vul_2 should be different than those of vul_3 and vul_4. Since vul_1 and vul_2 are both necessary for an attacker to achieve his goal p_1, they should be more important than either of vul_3 and vul_4. For example, if vul_3 is patched, vul_4 could still enable an attacker to obtain p_3. However, if we patch vul_1, this would break the attack chain to p_2 and p_1 will not be achievable. Thus in ranking the vertices, we would expect that values for vul_1 and vul_2 will be higher than vul_3 and vul_4. However, if we do not treat AND and OR vertices differently in D, after applying the computation given by (6) we would get identical values for all of $vul_1, ..., vul_4$. We apply AssetRank in a PageRank style so that all vertices are treated as OR vertices, with $\delta = 0.85$ and IV set up so that only the goal vertex p_1 has a non-zero intrinsic value. The sequence converges after four iterations and, not surprisingly, all the vulnerabilities have the same rank as presented in Table 1.

Table 1: *AssetRanks with only OR vertices*

Vertex	Rank $\times 10^2$
p_1	38.873
p_2	16.521
p_3	16.521
vul_1	7.021
vul_2	7.021
vul_3	7.021
vul_4	7.021

Hence, rather than splitting the value of an AND vertex we *replicate* it to its out-neighbours. Each out-neighbour of an AND vertex receives the full value from the vertex multiplied by δ. That is, for every outgoing edge (u, v) from an AND vertex u, the corresponding entry D_{vu} is 1. This is the basis for the restriction on AND-vertex arc weights given in Equation (3).

With this extension for AND vertices, the sequence (6) will no longer converge in general.[3] Since the columns in D corresponding to AND vertices may sum to a value greater than one, the L1-norm of X_t would increase indefinitely. However, the actual values of vertices are not important in ranking the vertex. What matters is their relative values with one another. Thus we normalize the vector X_t at each iteration and the computation becomes:

$$\text{Step 1: } X'_t = \delta D X_{t-1} + (1-\delta) IV \tag{7}$$

$$\text{Step 2: } X_t = \frac{1}{||X'_t||_1} X'_t \tag{8}$$

In our experience, the above sequence has always converged. However, we have not found a mathematical proof that it will always reach an equilibrium point. In the next section, we give an interpretation of the above sequence in the context of attack graphs and show that when the sequence converges, the resulting values indicate the importance of each vertex to an attacker. For safety, in our implementation we have set a maximum number of iterations so that the algorithm will terminate if the sequence does not converge within this limit. Table 2 displays the result of applying the above algorithm to the dependency graph in Figure 5. The same values for δ and IV are used and the computation converges after 38 iterations. The computation time is less than a second for all of the experiments presented in this paper.

The new algorithm gives the expected relative importance for the four vulnerabilities: vul_1 and vul_2 are more important than vul_3 and vul_4.

[3]The matrix D with only OR vertices can be converted to a stochastic matrix and Markovian-process theory guarantees an equilibrium point can be reached. After the introduction of AND vertices the matrix is no longer stochastic.

Table 2: *AssetRanks with AND and OR vertices*

Vertex	Rank $\times 10^2$
p_1	17.219
p_2	16.801
p_3	16.801
vul_1	16.393
vul_2	16.393
vul_3	8.197
vul_4	8.197

4 Interpretation of AssetRank

In this section we describe a stochastic interpretation for the numeric value computed by AssetRank on dependency attack graphs. Stochastic interpretation has been used to give PageRank, which can be seen as a special case of AssetRank, a semantic meaning in a "random walk" model [9, 13]. A random walker surfs the web graph in the following manner: At each time interval, with probability δ it will follow one of the links in the current page with equal probability; with probability $1-\delta$ it will "get bored" and jump to one of the pages in the web graph with equal probability. Under this interpretation, the equilibrium point of sequence (6) will be the probability a random surfer is on a page.[4] This random-walk model cannot be applied to dependency attack graphs, primarily because it does not handle AND and OR vertices differently. In this section we give an interpretation of AssetRank that provides meaningful semantics in the context of dependency attack graphs.

Our interpretation is inspired by the model used by Bianchini *et al.* [13]. Imagine a potential attacker has the attack graph[5] and is planning how to attack the system. He does so by dispatching an army of "attack planning agents" whose task is to learn how to obtain the privileges represented by the vertices. Every agent behaves in the following manner: at each moment an agent considers only one vertex in the attack graph. We use $v_i(t)$ to denote the vertex agent i is contemplating at time t. If $v_i(t)$ is a sink vertex, agent i has finished his job and stops working. Otherwise he will, with probability δ, plan how to satisfy the requirements for $v_i(t)$ based on the attack graph; with probability $1-\delta$, he will decide to obtain the privilege $v_i(t)$ through other means not encoded in the attack graph (for example, through backdoors already installed in the system or social engineering). In the latter case, the agent has also finished his planning and stops working.

Let $v_i(t) = v$. With probability δ the agent uses the attack graph and follows the out-going

[4] An additional step is needed to convert the matrix D to a stochastic matrix by compensating for the lost value due to dangling pages [13].

[5] In reality an attack graph should never be leaked to an attacker; however, in evaluating security we need to assume that the attacker has the same information resources as the defenders.

arcs from v to satisfy its preconditions. Two cases need to be considered. If v is an OR vertex, the agent will choose one of its out-neighbours w with the following probability.

$$Pr[\, v_i(t+1) = w \mid v_i(t) = v \,] = g(v, w) \tag{9}$$

If v is an AND vertex, the agent must plan how to satisfy *all* the out-neighbours of v. Thus he must move along all the out-going arcs simultaneously. We model this by allowing the agent to replicate itself[6] with each replica moving to one of the out-neighbours independently. More precisely, at step $t+1$ agent i will become $r = |N^+(v)|$ agents $i_1, ..., i_r$, each of which is assigned one of the vertices in $N^+(v)$ so that every element in $N^+(v)$ is covered.

The potential attacker has an unlimited number of such agents at his disposal. Every time he dispatches an agent to a vertex in the attack graph, the agent will try to find a way to attack the system such that the goal represented by the starting vertex can be achieved. When the agent (and all his clones) finishes the job, an attack plan has been made. Each time he may find a different attack path due to the probabilistic choices he makes along the way. At each time interval, the potential attacker will dispatch new agents and the new agents will start from one of the graph vertices with the probability distribution specified by IV. The number of new agents is $(1-\delta)$ times the number of active agents currently in the system.

Let the vector $X_t = [X_t^1, ..., X_t^{|V|}]^T$ where X_t^v is a random variable representing the number of active agents planning an attack for vertex v at time t. $E(X_t^v)$ is the expected value of the random variable X_t^v. We use $E(X_t)$ to represent $[E(X_t^1), ..., E(X_t^{|V|})]^T$. Let $E(X_0) = IV$ which corresponds to the attacker dispatching the first agent according to the probability distribution given by IV. The following equation then holds for $t > 0$.

$$E(X_t) = \delta D E(X_{t-1}) + (1-\delta)||E(X_{t-1})||_1 IV \tag{10}$$

After normalization, this is exactly the same sequence as the sequence specified by Equations (7) and (8). Since an agent will replicate itself at an AND vertex, $||E(X_t)||_1$ grows indefinitely; however, the relative ratio of each $E(X_t^i)$ with respect to $E(X_t)$ may reach an equilibrium point. If the normalized value of $E(X_t)$ stabilizes as $t \to \infty$, the AssetRank value computed by the sequence specified in Equations (7) and (8) will represent the portion of active attack planning agents on each vertex in the attack graph.

Under this attack-planning-agents interpretation, a higher AssetRank value for a vertex indicates there will be a larger portion of planning agents discovering how to obtain the asset represented by the vertex. Thus AssetRank directly implies the importance of the privilege or vulnerability to a potential attacker. The arc weight $g(v, w)$ indicates the desirability

[6] Analogous to the UNIX `fork()` command.

of the attack step (v, w) with respect to achieving the capability v, since a higher $g(v, w)$ means a planning agent will be more likely to choose w as v's enabler. A vertex's intrinsic value represents the desirability of the privilege to an attacker. A higher intrinsic value indicates the attacker is more likely to dispatch a planning agent to determine how to achieve the goal. A higher $1 - \delta$ indicates the attacker is more likely to gain privileges "out of band" and thus does not need to follow the attack graph. $1 - \delta$ also indicates the rate at which the attacker dispatches new agents.

5 Experiments

In this section we present several experiments we conducted to study 1) whether the AssetRank algorithm gives ranking results consistent with the importance of an attack asset to a potential attacker; and 2) how to use the AssetRank value to better understand security threats conveyed in a dependency attack graph, and to choose appropriate mitigation measures.

In our experiments, we use the MulVAL attack-graph tool suite [4] to compute a dependency attack graph based upon a network description and a user query. For example, a user may ask if attackers can execute code of their choosing on any server. The attack graph is exported to a custom Microsoft Access database application. The database application uses SQL queries and VBA code to normalize the input data and compute the AssetRank values.

We make the assumption that the attacker prefers shorter attack paths, and we set the weights of out-going arcs from an OR vertex to indicate this preference.[7] In an AND/OR dependency attack graph, an *attack path* to satisfy a vertex w is a tree sub-graph rooted at w in which every non-leaf OR vertex has exactly one child from the original graph, every non-leaf AND vertex has all the out-neighbours in the original graph, and every leaf-vertex is a sink vertex in the original graph. The length of this attack path is the maximum depth of this tree. By performing a standard depth-first-search, we compute for every vertex in the graph the length of the shortest attack path to satisfy it. For an OR vertex v, we assign the weight for an out-going arc (v, w) as follows. If m_w is the length of the shortest attack path satisfying w, then the length of the shortest attack path satisfying v along the arc (v, w) is $m_w + 1$. The arc is assigned a weight of $1/(m_w + 1)^2$ where we chose the exponent to reflect the degree of bias against long attack paths. We then normalize the weights among all the out-neighbours from an OR vertex and the result is the $g(v, w)$ parameters in the AssetRank algorithm.

We first demonstrate the results of applying AssetRank to the attack graph for the example network in Figure 1. In this example we assign equal intrinsic value to all the vertices in the

[7]The weights could also be used to assume other preferences such as the desirability of simple or complex attacks. This notion is discussed in more detail in Section 6.

attack graph, indicating that the attacker is interested in all the assets equally. In a MulVAL attack graph, each vertex is associated uniquely with a logical sentence in the form of a predicate applied to a number of parameters, describing an attack asset. Table 3 shows the AssetRank values for some of the interesting MulVAL-generated attack graph vertices. For this example, AssetRank took 39 iterations to converge.

Table 3: *AssetRanks for the Figure 1 network*

Vertex	Rank$\times 10^2$
execCode(c,serviceaccount)	2.857
execCode(d,serviceaccount)	2.857
execCode(e,serviceaccount)	1.619
execCode(b,serviceaccount)	1.587
execCode(f,serviceaccount)	0.343
hacl(a,d,tcp,80)	3.279
hacl(a,c,tcp,80)	3.279
hacl(a,b,tcp,80)	2.354
hacl(d,e,tcp,80)	1.715
hacl(c,e,tcp,80)	1.715
hacl(e,f,tcp,80)	1.619
hacl(c,d,tcp,80)	0.965
hacl(b,d,tcp,80)	0.965
hacl(d,c,tcp,80)	0.965
hacl(b,c,tcp,80)	0.965
hacl(d,b,tcp,80)	0.862
hacl(c,b,tcp,80)	0.862
...	...
vulExists(c,...remoteExploit,privEscalation)	3.372
vulExists(d,...remoteExploit,privEscalation)	3.372
vulExists(e,...remoteExploit,privEscalation)	2.203
vulExists(b,...remoteExploit,privEscalation)	2.174
vulExists(f,...remoteExploit,privEscalation)	0.999

In Table 3, we group vertices with the same predicate together making it easier to compare the relative importance of privileges within the same category. For example, the vertex "execCode(d,serviceaccount)"[8] has a value of 0.02857. Another privilege in the same execCode category, "execCode(b,serviceaccount)", only has a value of 0.01587. Intuitively this is correct because while both machines B and D are accessible directly by the attacker from A, machine D could enable the attacker to directly penetrate deeper into the right subnet while machine B can only do so indirectly through C or D. Thus the compromise on D

[8]Meaning the attacker can have code-execution privilege as user "serviceaccount" on machine "d". MulVAL is implemented in Datalog which requires that the first letter of a constant be lowercase; however, we will use uppercase for machine names outside of predicates.

is more serious. For the same reason, a software vulnerability on machine C or D is more valuable to an attacker than one on machine B, as shown by the ranking in the "vulExists" category. The AssetRank values in this example indicate a software vulnerability or a machine compromise that can enable more penetrations is more valuable to the attacker, which is what we expected. Machines whose compromise and vulnerabilities are ranked higher should be given priority consideration for security hardening, such as patching and installing Intrusion Detection System (IDS) devices.

In MulVAL, a tuple "hacl(H1, H2, Protocol, Port)" means "machine H1 can initiate a network conversation to machine H2 through Protocol and Port." Host Access Control List (HACL) tuples are high-level abstractions of the effects of network traffic-control devices such as firewalls, routers, and switches, whose settings a system administrator can modify. The ranking of the HACL predicates demonstrates the effectiveness of AssetRank. The removal of the first two HACL predicates requires the attacker to launch a more complicated attack since then he must take the extra step of exploiting B. The removal of the first three HACL predicates eliminates the entire network attack. If any of those three HACL predicates are not removed then the attacker has an entry to the network and the best we can do is save the two machines in the right subnet. Removing the fourth and fifth HACL predicates will do precisely that. Finally, if the first five HACL predicates are left intact the only machine that can be saved is F which can be accessed after exploiting E. AssetRank has predictably ranked the HACL predicate from E to F as the next most important. We see that the AssetRank values correctly reflect the importance of network routes to an attacker, and thus identify the routes whose removal would be most effective in protecting machines in the network. All machines in this example were given equal intrinsic value which corresponds to the goal of protecting as many machines as possible. The AssetRank values for the HACL tuples are consistent with this goal.

The second experiment is adapted from an example in a technical report by Lippmann *et al.* [14]. In this example, we demonstrate how AssetRank may be used to suggest courses of action. Figure 6 presents a network with a web server, database server and user desktop. Each of the two servers has a remotely exploitable vulnerability resulting in a privilege escalation. The user desktop has a browser vulnerability which is exploited if the user is lured to a maliciously crafted website. Attackers are located on the Internet and in this scenario we assume their most important goal is gaining access to the database server.

Since the attackers' main interest is the database server, we give the attack graph goal vertex "execCode(databaseServer, oracle_user)" an intrinsic value equal to 15% of the total and the remaining 85% is distributed evenly among the remaining vertices to reflect the attackers' side interest in attaining any privilege.

Figure 7 displays the attack graph for this scenario. The vertices have been coloured according to their AssetRank values with colours ranging from red to blue. Red vertices have a high AssetRank value and blue vertices have a low AssetRank value. Table 4 shows the

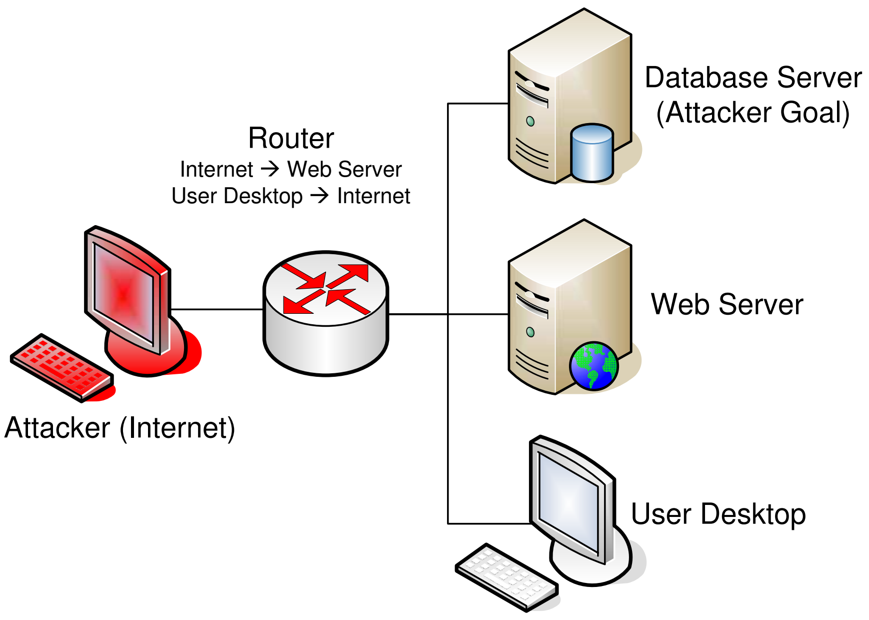

Figure 6: *Experiment 2, Scenario 1*

Table 4: *AssetRanks for the Figure 6 network*

Name	Rank $\times 10^2$
execCode(userDesktop,joeAccount)	3.357
execCode(databaseServer,oracle_user)	2.764
execCode(webServer,system)	2.575
hacl(userDesktop,attackerHost,tcp,80)	4.659
hacl(attackerHost,webServer,tcp,80)	3.708
hacl(userDesktop,databaseServer,tcp,1521)	2.575
hacl(webServer,databaseServer,tcp,1521)	2.575
hacl(userDesktop,webServer,tcp,80)	1.285
...	...
vulExists(userDesktop,browser_vulid,...)	4.039
vulExists(databaseServer,oracle_vulid,...)	3.499
vulExists(webServer,iis_vulid,...)	3.327

AssetRank values in some of the vertex categories.[9] In this example we would like to focus on the categories that represent configuration options, since these will be conditions a system administrator can change to mitigate the threats. The "hacl" and "vulExists" predicates are two of these categories. The highest-ranked HACL is the desktop machine's ability to access the Internet ("attackerHost"), followed by the accessibility from Internet to the web server. It may seem counter-intuitive that out-bound network access, typically deemed less harmful, is ranked higher than the inbound access to the web server. This is because the

[9]The algorithm required 44 iterations to converge.

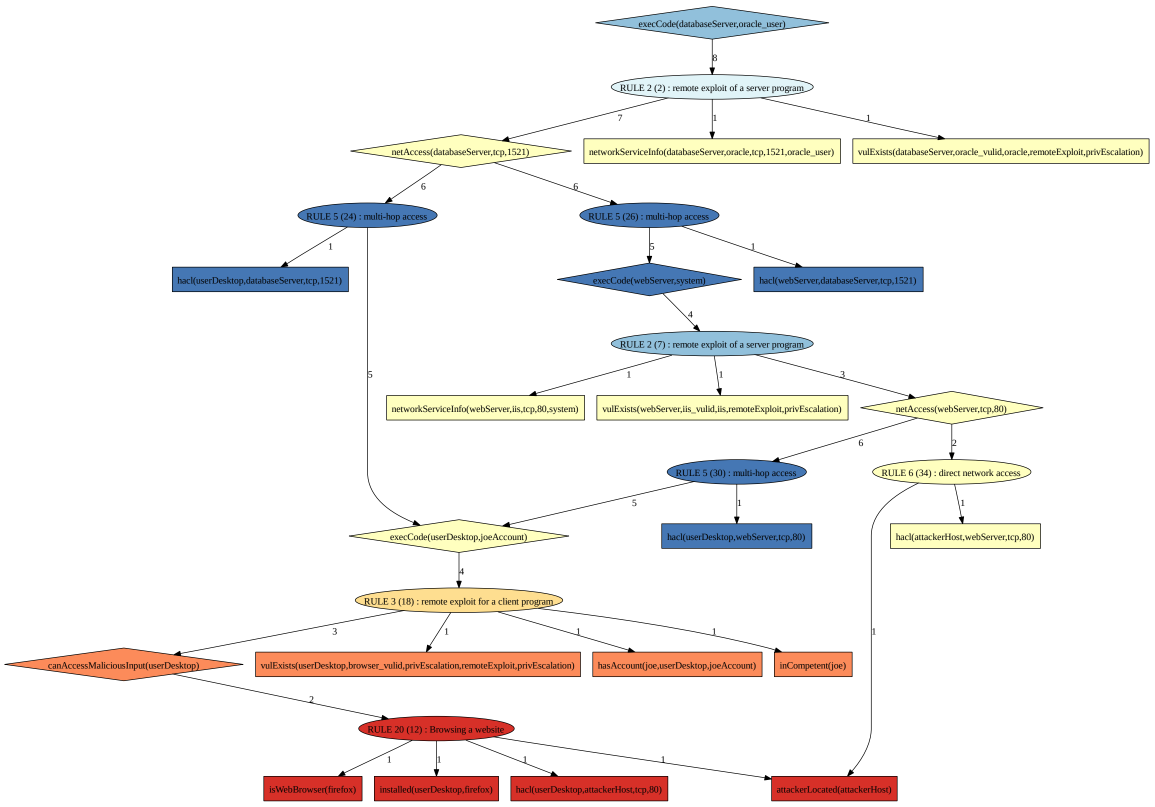
execCode(databaseServer,oracle_user)
8
RULE 2 (2) : remote exploit of a server program
7
1
1
netAccess(databaseServer,tcp,1521)
networkServiceInfo(databaseServer,oracle,tcp,1521,oracle_user)
vulExists(databaseServer,oracle_vulid,oracle,remoteExploit,privEscalation)
6
6
RULE 5 (24) : multi-hop access
RULE 5 (26) : multi-hop access
1
5
1
hacl(userDesktop,databaseServer,tcp,1521)
execCode(webServer,system)
hacl(webServer,databaseServer,tcp,1521)
4
RULE 2 (7) : remote exploit of a server program
5
1
1
3
networkServiceInfo(webServer,iis,tcp,80,system)
vulExists(webServer,iis_vulid,iis,remoteExploit,privEscalation)
netAccess(webServer,tcp,80)
6
2
RULE 5 (30) : multi-hop access
RULE 6 (34) : direct network access
5
1
1
execCode(userDesktop,joeAccount)
hacl(userDesktop,webServer,tcp,80)
hacl(attackerHost,webServer,tcp,80)
4
RULE 3 (18) : remote exploit for a client program
3
1
1
1
1
canAccessMaliciousInput(userDesktop)
vulExists(userDesktop,browser_vulid,privEscalation,remoteExploit,privEscalation)
hasAccount(joe,userDesktop,joeAccount)
inCompetent(joe)
2
RULE 20 (12) : Browsing a website
1
1
1
1
isWebBrowser(firefox)
installed(userDesktop,firefox)
hacl(userDesktop,attackerHost,tcp,80)
attackerLocated(attackerHost)

Figure 7: *Attack graph for the Figure 6 network*

out-bound access, along with the user's browser vulnerability, makes it possible for the user to fall victim to a malicious website that exploits the browser vulnerability, and as a result, attackers can gain a flexible foothold *inside* the corporate network. The desktop gives two paths to the database server — it can exploit the server directly or it can launch a two stage attack through the web server. By exploiting the desktop, attackers gain everything they would by exploiting the web server plus additional capabilities.

It may be tempting to think that the best course of action is blocking the top-ranked HACL predicates. In reality, however, neither of the top two HACL predicates can be removed due to the business needs of user desktop access to the Internet and web server availability from the Internet. We then consider the next two most important HACL predicates: the desktop and web server's ability to directly access the database server. This time it is possible to remove the route from the desktop machine to the database server, since an ordinary user does not need to have direct access to the database server. We can then propose the course of action of putting the desktop machine into a separate subnet and blocking access from the subnet to the database server in the router configuration.

The "vulExists" category shows that the vulnerability on the desktop machine is more important than the one on the database server, which is more important than the one on the web server. We can see that this assessment is correct since the compromise of a desktop machine is more valuable to attackers, as discussed above. Since the target is the database server, and the web server is not necessary in this scenario, the web server vulnerability is less important than the database server vulnerability.

While it is tempting to simply recommend that all organizations keep their software fully patched, this is not realistic in an enterprise environment for a number of reasons. First, it is extremely expensive and time consuming to roll out a patch in a large network; furthermore, patches may not be immediately available and, once they are released, they must be tested against the organization's baseline before being deployed. Second, business needs might favour uptime over security and so patches are not applied as soon as they are released (and tested) but are applied on a patch cycle. Third, patches are a reactive security measure and organizations are often better protected if they can make architectural changes that mitigate the consequences of an attacker exploiting a vulnerability. In this scenario we assume that it is not realistic to keep the desktop machines fully patched and that the vendors have not yet provided workable patches for the server software.

Thus the only course of action available is the architectural change to the network topology described above. We would like to evaluate how effective this proposed change will be in mitigating the threats. This can be done by simulating the new configuration in MulVAL and computing a new attack graph based on the suggested changes.

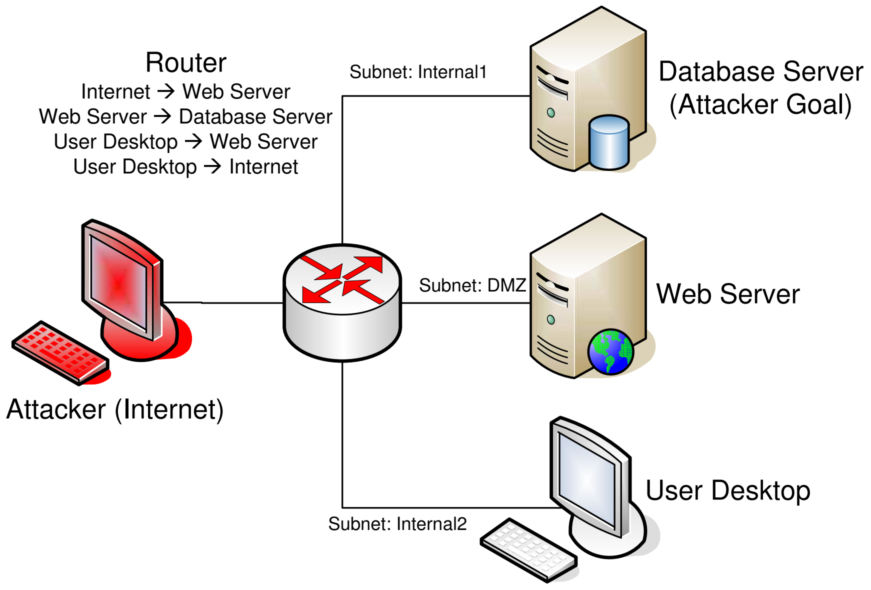

Figure 8: *Experiment 2, Scenario 2*

Figure 8 illustrates the new network configuration. The database server has been placed into the subnet Internal1 and is only accessible from the web server. The user has been placed into the subnet Internal2 and she has access to the web server and the Internet. The web server is in the subnet DMZ and is still accessible from the Internet. The MulVAL-generated attack graph on the new scenario is coloured according to the AssetRank values in Figure 9. The intrinsic value and arc weights are assigned as before. Table 5 gives the new AssetRank values for the new attack graph. The values indicate that in the changed configuration, the vulnerability and privileges of the web server become the most valuable assets for the attacker. The reason is that the attacker must now compromise the web server to reach the database server. The new configuration is better since now only a single entry point is presented to the attacker. If the web server vulnerability is patched, the only attack path to the database server is eliminated. Furthermore, the system administrator likely monitors the web server much more closely than the individual desktop machines.

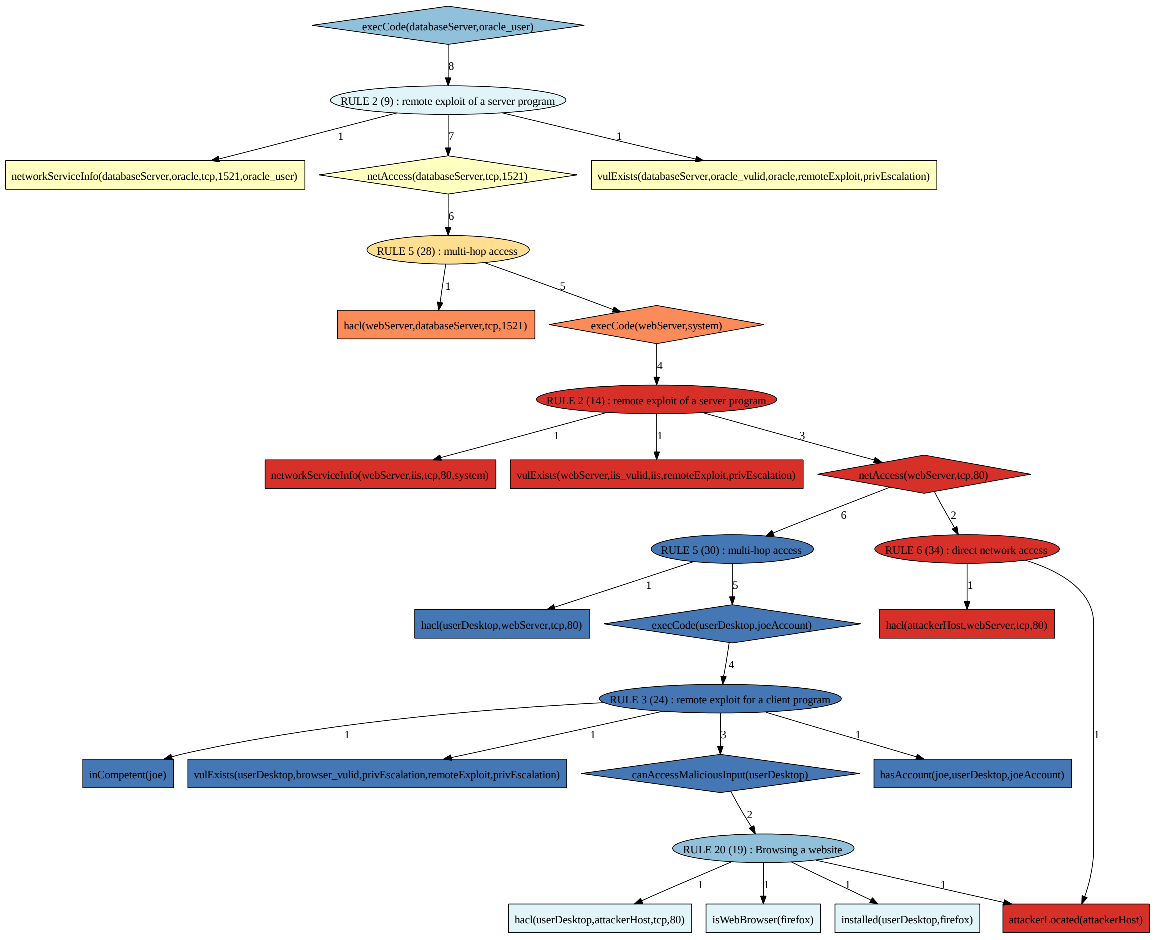
execCode(databaseServer,oracle_user)
RULE 2 (9) : remote exploit of a server program
networkServiceInfo(databaseServer,oracle,tcp,1521,oracle_user)
netAccess(databaseServer,tcp,1521)
vulExists(databaseServer,oracle_vulid,oracle,remoteExploit,privEscalation)
RULE 5 (28) : multi-hop access
hacl(webServer,databaseServer,tcp,1521)
execCode(webServer,system)
RULE 2 (14) : remote exploit of a server program
networkServiceInfo(webServer,iis,tcp,80,system)
vulExists(webServer,iis_vulid,iis,remoteExploit,privEscalation)
netAccess(webServer,tcp,80)
RULE 5 (30) : multi-hop access
RULE 6 (34) : direct network access
hacl(userDesktop,webServer,tcp,80)
execCode(userDesktop,joeAccount)
hacl(attackerHost,webServer,tcp,80)
RULE 3 (24) : remote exploit for a client program
inCompetent(joe)
vulExists(userDesktop,browser_vulid,privEscalation,remoteExploit,privEscalation)
canAccessMaliciousInput(userDesktop)
hasAccount(joe,userDesktop,joeAccount)
RULE 20 (19) : Browsing a website
hacl(userDesktop,attackerHost,tcp,80)
isWebBrowser(firefox)
installed(userDesktop,firefox)
attackerLocated(attackerHost)

Figure 9: *Attack graph for the Figure 8 network*

6 Discussion

From the experiments in the previous section, we can make several observations. First, the value given by the AssetRank algorithm is consistent with the logical importance of a

Table 5: *AssetRanks for the Figure 8 network*

Name	Rank $\times 10^2$
execCode(webServer,system)	4.535
execCode(databaseServer,oracle_user)	2.839
execCode(userDesktop,joeAccount)	1.566
hacl(attackerHost,webServer,tcp,80)	5.513
hacl(webServer,databaseServer,tcp,1521)	4.535
hacl(userDesktop,attackerHost,tcp,80)	3.431
hacl(userDesktop,webServer,tcp,80)	1.566
...	...
vulExists(webServer,iis_vulid,...)	5.297
vulExists(databaseServer,oracle_vulid,...)	3.717
vulExists(userDesktop,browser_vulid,...)	2.531

privilege or misconfiguration. For example, in Table 3 we saw that the highest-ranked vertices for the “execCode” category are those for machines C and D. Eliminating execCode entries for those two machines is critical to the goal of protecting the largest number of machines. The logical correctness of the ranks is especially evident in the ordering of the HACL predicates.

Second, simply applying AssetRank to an attack graph is not sufficient in itself to produce the best course of action. Input relating to business needs and costs is also required to arrive at the same course of action that a human would select. As shown in the second experiment, the two highest-ranked “hacl” tuples are those that must be there to meet business needs. This indicates that AssetRank can also be used to aid an administrator in the reverse task of ensuring legitimate access to servers. For example, AssetRank can reveal the most important communication paths, software, and hardware used to deliver the business’s priorities. Security hardening costs must also be factored in. For example, the highest-ranked “vulExists” tuple represents a vulnerability that cannot be easily patched due to management burdens. If the system does not integrate business needs and implementation costs, the administrator must manually determine the course of action while using the AssetRank values as a guide.

As a standalone tool, a very useful aspect of AssetRank in the context of attack graphs is to assist in prioritizing further analysis and understanding of the threats. One can imagine using AssetRank to incrementally show the vertices in an attack graph, with the highest ranked vertices shown first followed by the lower-ranked ones. In the limited number of experiments we conducted, the highest-ranked vertices were always the most important. Administrators can work through the ranked attack graph addressing the threats in order of their criticality. Since the full attack graph is often too cumbersome for a user to understand, this type of incremental analysis should be useful in practice.

Providing a well-founded semantic model is important to guarantee that computed numeric metrics are consistent with the assessments of real world experts. Having said that, every model has limitations and cannot completely capture reality. AssetRank shares a limitation with PageRank that arises due to an assumption made in its stochastic interpretation. The attack planning agent in our model, like the random walker in the PageRank model, is Markovian. This trait means the agents are memoryless and base their decisions solely on the current vertex and the weight of the out-going arcs. Hence, planning agents do not take into account the vertices they have previously traversed when they decide how to obtain future privileges. This assumption is valid if the attack graph does not contain cycles; however, when cycles exist, attack planners will not purposely avoid looping. Since the process is Markovian, agents will not recognize if they are traversing a path which they have already investigated. The result is that the vertices involved in a cycle accumulate disproportional rank values, much the same way that cycles in web page communities will make the PageRank of the involved web pages disproportionately high. We mitigate this effect by using the arc weights to reflect the shortest path to satisfy a vertex. The vertices that produce loops in an agent's traversal inevitably have a longer satisfaction path, and thus will be less desirable to attackers. Due to this approach our AssetRank algorithm is still able to give appropriate rankings, even when the attack graph contains cycles.

Arc weights are a flexible instrument that allow the user to take attacker preferences into account. In our paper we used the weights to favour short attack paths over long ones. Alternatively, the metric can be used to denote other attack characteristics.

- Stealthiness of an attack — allows the inclusion of IDSs in the model by giving a high penalty for attacks leaving evidence (log entries or system crashes for example) or detectable attacks over links monitored by an IDS.
- Resources required — gives the ability to penalize resource consuming attacks (for example, attacks that require password cracking or large bandwidth).
- Complexity — attacks executable by amateur attackers with well developed tools could be given a higher priority than theoretical attacks or attacks that only have proof-of-concept code available.

7 Related Work

Mehta *et al.* apply the Google PageRank algorithm to model-checking-based attack graphs [15]. Aside from the generalization of PageRank presented in this paper, the key difference from their work is that AssetRank is applied to dependency attack graphs which have very different semantics from the state enumeration attack graphs generated by a model checker. First, a vertex in a dependency attack graph describes a privilege attackers use or a vulnerability they exploit to accomplish an attack. Hence ranking a vertex in a dependency attack graph directly gives a metric for the privilege or vulnerability. Ranking a vertex in a state enumeration attack graph does not provide this semantics since a vertex represents the state of the entire system including all configuration settings and attacker privileges. Second, the source vertices of our attack graphs are the attackers' goals as opposed to the source vertex being the network initial state, as is the case in the work of Mehta *et al.* Since the attackers' goals are the source vertices, value flows from them and the computed rank of each vertex is in terms of how much attackers *need* the attack asset to achieve their goals. Thus our rank is a direct indicator of the main attack enablers and where security hardening should be performed. The rank computed in Mehta *et al.*'s work represents the probability a random attacker (similar to the random walker in the PageRank model) is in a specific state, in particular, a state where he has achieved his goal.

It is important to distinguish between the probability that an attacker is in a state *currently* and the probability he can reach the state *eventually*. The rank computed in Mehta *et al.*'s work is the former, not the latter. In reality, an attacker with a target in mind will always be able to reach the goal states in the attack graph. Indeed, in their model the probability that the random attacker has already reached a goal state at some point will go to one as time goes to infinity. To demonstrate why the probability a random attacker is in a goal state currently cannot serve as a meaningful security metric, we implemented their algorithm as described in the paper [15] and applied it to two very simple state enumeration attack graphs. The first one only has one attack path $p_1 \rightarrow g$. The second one has the same attack path plus an extra attack path $p_1 \rightarrow p_2 \rightarrow g$. g is the goal state for the attacker. The rank for g in the first graph is 0.500 and its rank in the second graph is 0.397. Hence, in an attack graph with more paths to achieve the goal, the probability a random attacker is at the goal state is actually lower, due to the fact that the attacker must spend more time in the other states. Clearly, this rank cannot serve as a metric for the system's overall vulnerability, as claimed in their paper, since the second case is obviously more vulnerable than the first one due to the additional attack path. In both PageRank and AssetRank, it is the relative rank values, not the actual values that are significant. The relative rank values in Mehta *et al.*'s work compare the likelihood a random attacker is currently in a state. Unlike our work, they do not indicate the relative importance of configuration settings and privileges to attackers in achieving their goal.

There have been various forms of attack graph analysis proposed in the past. The ranking scheme described in this paper is complementary to those works and could be used in combination with existing approaches. One of the factors that has been deemed useful for attack graphs is finding a minimal set of critical configuration settings that enable potential attacks since these could serve as a hint on how to eliminate the attacks. Approaches to find the minimal set have been proposed for both dependency attack graphs [3] and state-enumeration attack graphs [6, 8]. Business needs usually do not permit the elimination of all security risks so the AssetRank values could be used alongside minimal-cut algorithms to selectively eliminate risk. Our first experiment shows that the highest ranked vertices (compromise/vulnerability on host C and D) happen to be a minimal set that will cut the attack graph in two parts. AssetRank can also indicate the relative importance of each attack asset, which a binary result from the minimal-cut algorithm does not provide. This is illustrated in the second experiment, where although none of the attack assets on the web server and user desktop alone could completely cut the attack paths, their importance values are different and this is also useful in understanding the security threats and determining the best courses of action to counteract them.

It has been recognized that the complexity of attack graphs often prevents them from being useful in practice and methodologies have been proposed to better visualize them [7]. The ranks computed by our algorithm could be used in combination with the techniques in those works to help further the visualization process, for example by incrementally displaying vertices in an attack graph.

8 Conclusion and Future Work

In this paper we proposed the AssetRank algorithm, a generalization of the PageRank algorithm, that can be applied to rank the importance of a vertex in a dependency attack graph. The model adds the ability to reason on heterogeneous graphs containing both AND and OR vertices. It also incorporates intrinsic values to reflect an attacker's goal and arc weights to specify the desirability of an attack step. The numeric value computed by AssetRank is a direct indicator of how important the privilege or vulnerability represented by a vertex is to a potential attacker. The algorithm was presented theoretically through an attack-planning agent model, and empirically verified through numerous experiments conducted on several example networks. The rank value will be valuable to users of attack graphs in better understanding the security risks, in determining appropriate mitigation measures, and as input to further attack graph analysis tools.

In the future, we would like to explore how to incorporate business priorities and implementation costs into the rank value, so that the resulting ranks can be used immediately by a system administrator to generate a course of action or automatically implement security hardening measures. We would also like to conduct experiments on operational networks to better understand the advantages and limitations of our proposed algorithm, along with ways of improving it. Finally, we wish to see how the values for arc and vertex weights, representing diverse preferences, can be combined and, similarly, how AssetRanks in various contexts may be combined.

References

[1] Ammann, Paul, Wijesekera, Duminda, and Kaushik, Saket (2002), Scalable, Graph-Based Network Vulnerability Analysis, In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC.

[2] Ingols, Kyle, Lippmann, Richard, and Piwowarski, Keith (2006), Practical Attack Graph Generation for Network Defense, In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida.

[3] Noel, Steven, Jajodia, Sushil, O'Berry, Brian, and Jacobs, Michael (2003), Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs, In *19th Annual Computer Security Applications Conference (ACSAC)*.

[4] Ou, Xinming, Boyer, Wayne F., and McQueen, Miles A. (2006), A scalable approach to attack graph generation, In *13th ACM Conference on Computer and Communications Security (CCS)*, pp. 336–345.

[5] Phillips, Cynthia and Swiler, Laura Painton (1998), A graph-based system for network-vulnerability analysis, In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, ACM Press.

[6] Sheyner, Oleg, Haines, Joshua, Jha, Somesh, Lippmann, Richard, and Wing, Jeannette M. (2002), Automated generation and analysis of attack graphs, In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 254–265.

[7] Noel, Steven and Jajodia, Sushil (2004), Managing attack graph complexity through visual hierarchical aggregation, In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 109–118, New York, NY, USA: ACM Press.

[8] Jha, Somesh, Sheyner, Oleg, and Wing, Jeannette M. (2002), Two Formal Analyses of Attack Graphs, In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pp. 49–63, Nova Scotia, Canada.

[9] Page, Lawrence, Brin, Sergey, Motwani, Rajeev, and Winograd, Terry (1998), The PageRank Citation Ranking: Bringing Order to the Web, Technical Report Stanford Digital Library Technologies Project.

[10] Sheyner, Oleg (2004), Scenario Graphs and Attack Graphs, Ph.D. thesis, Carnegie Mellon.

[11] Swiler, Laura P., Phillips, Cynthia, Ellis, David, and Chakerian, Stefan (2001), Computer-Attack Graph Generation Tool, In *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, Vol. 2.

[12] Sawilla, Reginald (2006), Abstracting PageRank to dynamic asset valuation, (DRDC Ottawa TM 2006-243) Defence R&D Canada – Ottawa.

[13] Bianchini, Monica, Gori, Marco, and Scarselli, Franco (2005), Inside PageRank, *ACM Trans. Inter. Tech.*, 5(1), 92–128.

[14] Lippmann, Richard, Ingols, Kyle, Scott, Chris, Piwowarski, Keith, Kratkiewicz, Kendra, Artz, Michael, and Cunningham, Robert (2005), Evaluating and Strengthening Enterprise Network Security Using Attack Graphs, (Technical Report ESC-TR-2005-064) MIT Lincoln Laboratory.

[15] Mehta, Vaibhav, Bartzis, Constantinos, Zhu, Haifeng, Clarke, Edmund, and Wing, Jeannette (2006), Ranking Attack Graphs, In *Proceedings of Recent Advances in Intrusion Detection (RAID)*.