Identifying Critical Attack Assets in Dependency Attack Graphs

Reginald E. Sawilla[1] and Xinming Ou[2,*]

[1] Defence Research and Development Canada, Ottawa, Canada
[2] Kansas State University, Manhattan, Kansas, USA

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 AssetRank, a generalization of Google's PageRank algorithm which ranks web pages in web graphs. AssetRank addresses the unique semantics of dependency attack graphs and incorporates vulnerability data from public databases to compute metrics for the graph vertices (representing attacker privileges and vulnerabilities) which reveal their importance in attacks against the system. The results of applying the algorithm on a number of network scenarios show that the numeric ranks computed are consistent with the intuitive importance that the privileges and vulnerabilities have to an attacker. The vertex ranks can be used to prioritize countermeasures, help a human reader to better comprehend security problems, and provide input to further security analysis tools.

Keywords: attack graph, security metric, PageRank, eigenvector.

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 [7,8,9]. 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

* This author was partially supported by the U.S. National Science Foundation under Grant No. 0716665, and the U.S. Department of Energy. Any opinions, findings and conclusions or recomendations expressed in this paper are those of the authors and do not necessarily reflect the views of the U.S. government agencies.

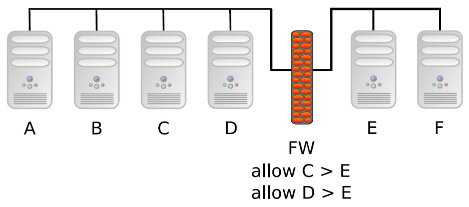


Fig. 1. An example network

which can distill the overwhelming amount of information into a list of priorities that will help them to secure the network, making efficient use of 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 such that the only allowed network access between the subnets is from C and 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. The resulting attack graph can be found in Appendix A. Even for a small network, the attack graph is barely readable. Assuming the attack graph can be read, it is still difficult for a human to capture the core security problems in the simple network. 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 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.

Beside the dependency relations represented in an attack graph, another important factor in determining the criticality of an identified security problem is the likelihood the attack path can lead to a successful exploit. For example, both hosts C and D can be exploited remotely by the attacker on host A. Assume that the vulnerability on host C is only theoretical and no one has successfully produced a proof-of-concept exploit, whereas the vulnerability on host D has a publicly available exploit that works most of the time. Obviously the vulnerability on D is more likely to be exploited than the vulnerability on C and so its elimination deserves prioritization.

In the past five years, significant resources have gone into standardizing the definition of the attributes of reported security vulnerabilities. Most notably, the Common Vulnerability Scoring System (CVSS)[1] is a standard for sharing the attributes of discovered security vulnerabilities among IT security professionals. It represents not just a single numeric score, but a metric vector that describes various aspects of a vulnerability such as its access vector, access complexity and exploitability. The CVSS metric vector is included in the National Vulnerability Database (NVD)[2] for every vulnerability reported in the NVD. The metrics provide crucial baseline information for automated security analysis. However, the metrics themselves can only give limited information without an understanding of the global security interactions in an enterprise environment. For example, further assume that the vulnerability on B is the same as the one on D. Since B does not have access into the right subnet, its vulnerability is less critical than the one on D. In the scenario just described, our algorithm gives first priority to the vulnerability on D, followed by the vulnerability on B, and then C. This prioritization is intuitive since D is easy to exploit and gives access to the right subnet; B is easy to exploit and gives access to D; and since only proof-of-concept code exists to exploit C, it warrants the lowest priority.

In summary, to determine the relative importance of security problems in a network, both the dependency relationships in the attack graph *and* the attributes of the security problems need to be considered. We present an approach which automatically digests the dependency relations in an attack graph as well as the baseline information of the vulnerability attributes to compute the relative importance of attacker assets (the graph vertices) as a numeric metric. The metric gauges the importance of a privilege or vulnerability to an attacker (and hence the defender). Our approach fuses attack graphs and baseline security metrics such as CVSS, to make both of them more useful in security analysis.

2 AssetRank for Attack Graphs

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 from one page to another. Google's PageRank algorithm [10] 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 generalization of the PageRank algorithm, which can handle the semantics of vertices and arcs of dependency attack graphs. Our first contribution allows AssetRank to treat the AND and OR vertices in a dependency attack graph correctly based on their logical meanings, whereas PageRank is only applied to OR vertex graphs. The second contribution is a generalization of PageRank's single system-wide damping factor to a per-vertex damping factor. This generalization allows AssetRank to accurately model the

[1] `http://www.first.org/cvss/`
[2] `http://nvd.nist.gov/cvss.cfm`

various likelihoods of an attacker's ability to obtain privileges through means not captured in the graph (out-of-band attacks). The third contribution is leveraging publicly available vulnerability information (*e.g.* CVSS) through parameters in AssetRank so that the importance of security problems is computed with respect to vulnerability attributes such as attack complexity and exploit availability. The fourth contribution is that our generalized ranking algorithm allows network defenders to obtain personalized AssetRanks to reflect the importance of attack assets with respect to the protection of specific critical network assets.

The AssetRank algorithm presented here could be applied to any graph whose arcs represent some type of dependency relation between vertices. In fact, web graphs are a special case of dependency graphs since a web page's functionality in part depends on the pages it links to.

A dependency attack graph G is 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 positive weights to vertices; g is a mapping of non-negative weights to arcs; and h is a mapping of vertices to their type (AND, OR, or SINK). The *out-neighbourhood* of a vertex v is defined as $N^+(v) = \{w \in V : (v, w) \in A\}$, and *in-neighbourhood* of v is defined as $N^-(v) = \{u \in V : (u, v) \in A\}$. The cardinality of a set X is denoted $|X|$ and its L1-norm is denoted $||X||_1$. Without loss of generality, we require the vector of all vertex weights $f(V)$ to sum to 1.

AssetRank is computed by solving for the principal eigenvector X in the following equation.

$$\lambda X = (D\mathit{\Delta} + \gamma P e^T)X \tag{1}$$

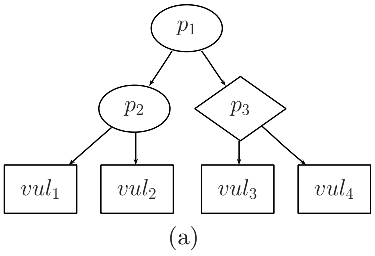
Where λ is the principal eigenvalue, X is the vector of AssetRanks (scaled to sum to 1), D is the transpose of the square adjacency matrix of a dependency attack graph G (an AND/OR directed graph), $\mathit{\Delta}$ is a diagonal matrix of vertex-specific arc-weight damping factors where each value is in the range $[0, 1]$, $\gamma \in (0, 1]$ is the vertex-weight damping factor, $P = f(V)$ is a personalization vector composed of the vertices' personalization values (that is, the vertex weights), and e is the all-ones vector.

Equation (1) reduces to the original PageRank if $\lambda = 1$, $\mathit{\Delta} = \delta I$ (where I is the identity matrix and δ is PageRank's damping factor), $\gamma = 1 - \delta$, and all vertices are required to be OR vertices.

2.1 AND Vertices

Dependency attack graphs contain both AND and OR vertices. An OR vertex can be satisfied by any of its out-neighbours, whereas an AND vertex depends on *all* of its out-neighbours. For example, the simple dependency attack graph in Figure 2(a) shows that attackers attaining the goal p_1 depend upon their ability to obtain both privileges p_2 and p_3. p_2 is an AND vertex[3] and it requires the

[3] In our figures, AND vertices are represented by ovals, OR vertices are represented by diamonds, and SINK vertices are represented by rectangles.



Vertex	AssetRank
p_1	0.1722
p_2	0.1680
p_3	0.1680
vul_1	0.1639
vul_2	0.1639
vul_3	0.0820
vul_4	0.0820

(b)

Fig. 2. AssetRank computation for an AND/OR graph

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.

Since any of an OR vertex's out-neighbours can enable it, the importance of each out-neighbour decreases as the number of out-neighbours increases 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 numbers grow.

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 the vertex's damping factor. That is, for every outgoing edge (u, v) from an AND vertex u, the corresponding matrix entry D_{vu}[4] is 1. We now have the following restrictions on the graph's arc weights.

$$\sum_{w \in N^+(v)} g(v,w) = \begin{cases} |N^+(v)|, & \text{if } h(v) = \text{AND} \\ 1, & \text{if } h(v) = \text{OR} \\ 0, & \text{if } h(v) = \text{SINK} \ . \end{cases} \tag{2}$$

A unique principal eigenvector X in Equation (1) exists (up to scalar multiplication) and follows from Perron's theorem (see, for example, [11]), and the fact that $D\Delta + \gamma P e^T$ is positive. Thus, convergence using the power method is guaranteed. The computation using the power method with the terms optimized to take advantage of the sparsity of $D\Delta$ follows.

$$\text{Step 1: } X'_t = D\Delta X_{t-1} + \gamma P; \qquad \text{Step 2: } X_t = \frac{1}{||X'_t||_1} X'_t \tag{3}$$

Figure 2(b) displays the result of applying the above algorithm to the graph in Figure 2(a). For this example, we use a single constant damping factor of $\Delta = 0.85I$ and P is such that only the goal vertex p_1 has a non-zero personalization value.

[4] As a shorthand notation we use u and v in D_{vu} to represent the column and row indices corresponding to the respective vertices.

AssetRank gives[5] the expected relative importance for the four vulnerabilities: vul_1 and vul_2 are twice as important as vul_3 and vul_4 since patching one of vul_1 or vul_2 has an equivalent effect in denying the goal p_1 as patching both vul_3 and vul_4.

2.2 Vertex-Specific Damping

In the case of PageRank applied to web pages, the system-wide damping factor δ gives the probability that surfers will stop surfing [12]. They could stop surfing for any number of reasons including having found the desired information or encountering a poor quality web page. The reality is that not all web pages have an equal likelihood to be the end point of a user's surfing. On some web pages almost all of the surfers will continue surfing (for example, search results) while on other pages, almost all of the surfers will stop surfing (for example, a local weather page).

An analogous situation exists for attack graphs. An "attack planner" will more likely stop traversing the attack graph if the vertex represents a privilege that can be easily obtained "out-of-band". For example, attackers requiring the ability to execute code on a user desktop could use out-of-band methods such as social engineering rather than purely technical exploits.[6]

In general, the damping factor measures the likelihood that an attack planner will continue traversing the graph. We improve the accuracy of the ranks by not assuming that the planners are equally likely to stop traversing the graph regardless of the vertex they are visiting. Rather than using a single damping factor, we introduce vertex-specific damping factors δ_v and assemble them into the diagonal damping matrix $\Delta = diag(\delta_1, \delta_2, \ldots, \delta_{|V|})$.

2.3 Personalization Vector

It is insufficient to consider only the dependency relations and damping factors in determining a vertex's value. Network defenders place a higher priority on defending critical servers than non-critical PCs. Similarly, some machines are more valuable than others to attackers. We use vertex weights as a *personalization value* to represent a vertex's inherent value to network attackers or defenders. Network defenders may identify the assets they desire to deny the attacker by assigning them a personalization value that reflects their importance to the defender's operations. The remaining attack assets are assigned a value of 0 which then causes the computed AssetRank values to reflect their importance only in so far as they are likely to be used by an attacker to obtain the attack assets identified as critical.

[5] All of the experiments in this paper required a computation time of less than one second on a typical desktop PC and converged in 78 iterations or less. The complexity of the power method depends upon the complexity of matrix multiplication and the number of iterations required. The complexity of naive matrix multiplication is $O(n^3)$. Speed improvements for PageRank computation can also speed up AssetRank computation as long as they do not require the principal eigenvalue to be 1.

[6] The attack graphs we use in this paper include only technical exploits.

3 Parameter Assignment

Attack graph dependencies and attack asset attribute information (such as CVSS metrics obtained from the NVD database) supply the three key components D, Δ, and P of the AssetRank matrix $A = D\Delta + \gamma Pe^T$. In this section we explain how to obtain and set these values. In Section 4 we will demonstrate their effect on the asset ranks. The parameter γ sets the influence of the personalization vector which has the effect of opting to favour attack assets closer to the goal versus favouring attack assets closer to the attacker.

3.1 Dependency Matrix (*D*)

To model attacker preferences we assign a *success likelihood* $s(v)$ to every vertex. The success likelihood has a slightly different meaning for the three types of vertices: AND, OR, and SINK.

The SINK vertices represent the ground facts that MulVAL uses when deriving attack paths. The ground facts include the existence of vulnerable software, network routes and the services running on each machine. Every ground fact is assigned a success likelihood. To simplify the demonstration in this paper we assign the success likelihood 1 to all non-vulnerability SINK vertices. That is, we assume that if a service exists, it is always up, and that network paths are stable.[7]

CVSS is a standard for specifying vulnerability attributes. Two attributes that are particularly useful in prioritizing attack assets are the base metric of Access Complexity (AC) and the temporal metric of Exploitability (E). For the AC metric, vulnerabilities are assigned a value of high, medium, or low, to indicate the existence of specialized access conditions such as a race condition or configuration setting. When considering the E metric, vulnerabilities are assigned a value of unproven, proof-of-concept, functional, or high, to indicate the current state of exploit maturity. If one attack path in the attack graph depends upon an unproven vulnerability and another attack path depends upon a vulnerability with functional exploit code, the attack assets in the latter attack path (all vulnerabilities and network routes) are more likely to be involved in an attack and so they are more valuable to attackers. Consequently, they also deserve a higher degree of attention by network defenders. In our experiments we assign the following success likelihoods $s(v)$ to each vulnerability vertex v to indicate the probability that an attacker will successfully exploit the vulnerability: Unproven (1%), Proof-Of-Concept (40%), Functional (80%), High (99%).

MulVAL attack graphs also contain `rule` vertices. These are AND vertices that specify how a privilege may be obtained. The parameter $s(v)$ for AND vertices models the preference of attackers for different attack strategies. For example, two of the rules describe how network access may be obtained. In the

[7] Users could assume mobile devices are present intermittently and hence assign a success likelihood to network routes for mobile devices that represent the likelihood that the device will be connected to the network.

first case, direct network access to a host is obtained if an attacker has a machine and a network route exists from that machine to the intended host. In the second case, multi-hop network access to a host is obtained if an attacker can execute code of his choosing on a victim machine and a network route exists from that machine to the intended host. Since an attack is complicated by multi-hop access, we assume that the attacker prefers direct routes so we assign a preference score of 1.0 to the direct route and 0.5 to the indirect route. In a similar manner, other rules may be assigned a preference score indicating attackers' preferences. These rule preferences would be set by experts to model different types of attackers (for example, script kiddies or black-hat criminals).

Finally, MulVAL attack graphs contain derived attack assets.For example, MulVAL-generated attack graphs include `execCode(machine,account)` vertices stating that an attacker could obtain the ability to execute arbitrary code on `machine` at the privilege of `account`. However, the `execCode` attack asset might be obtained through a choice of multiple routes in the attack graph. These multiple routes are represented by multiple outgoing arcs from the `execCode` vertex, an OR vertex. Not all of these routes are equally difficult to obtain and we make the assumption that attackers prefer easier methods of obtaining the derived attack asset. For example, attackers would favour routes that may be exploited with reliable tools.

The OR vertices discussed at the beginning of this section.

Attack paths will contain several ground facts (SINK vertices), rules (AND vertices), and derived attack assets (OR vertices). Weights of the out-going arcs are computed by percolating the success likelihoods throughout the graph by setting $g(u, v) = m(v)$ where

$$m(v) = \begin{cases} s(v), & \text{if } h(v) = \text{SINK} \\ s(v) \prod_{w \in N^+(v)} m(w), & \text{if } h(v) = \text{AND} \\ \max_{w \in N^+(v)} m(w), & \text{if } h(v) = \text{OR} \end{cases} \tag{4}$$

In words, the arc weight from vertex u to v is the success likelihood of v if v is a SINK vertex, the attacker's preference for the attack type multiplied by the product of all of the paths required for v if v is an AND vertex, and the easiest path from v if v is an OR vertex. Finally, the arc weights are normalized according to Equation (2).

3.2 Damping Matrix (Δ)

In Section 2.2 we introduced vertex-specific damping factors. This extension allows the modeling of out-of-band attacks for derived attack assets (OR vertices). For example, the ability to execute code on a victim's machine can be gained by obtaining the victim's login credentials through social engineering — a non-technical attack that is not captured in the attack graph. If attackers gain the attack asset v by means outside the graph, they will not require the dependencies of v captured in the attack graph so those dependencies are less valuable to the attacker and so deserve less attention from network defenders.

For MulVAL attack graphs, specifying a damping factor is only sensible for OR vertices (derived attack assets). The damping factor has no effect on SINK vertices because they have no out-going arcs. Also, AND vertices are fundamentally required in the attack graph and cannot be obtained out-of-band so the damping factor for AND vertices is set to 1 (no damping).

The success likelihood of obtaining a derived asset out-of-band for an OR vertex v is denoted $s(v)$. An example of an out-of-band attack is an attacker obtaining a user's login credentials through social engineering. The success likelihood depends upon the level of awareness and training of the user. A network defender can specify the success likelihood based upon the type of user account. For example, root users could be assigned a low likelihood score such as 20% while standard users could be assigned a score of 80%. Security experts will be relied upon to provide metrics for out-of-band attacks.

The degree to which attackers will use out-of-band attacks depends upon both the projected success of the out-of-band attack and the difficulty of obtaining the attack asset by using the means specified in the attack graph. If the attack asset may be obtained with certainty using the attack graph then the attacker will use those means. Also, if out-of-band attacks are impossible or are certain to fail, the attacker will not exit the graph to attempt the out-of-band means but will use the means in the attack graph to obtain the privilege. The following equation captures these requirements. For an OR vertex v with an out-of-band success likelihood $s(v)$, the damping factor δ_v is given by

$$\delta_v = (1 - s(v)) + s(v)m(v) \ . \tag{5}$$

The damping matrix is a diagonal matrix constructed from the vertex-specific damping factors by setting $\Delta = diag(\delta_1, \delta_2, \ldots, \delta_{|V|})$.

3.3 Personalization Vector (*P*)

The personalization vector P represents the network defender's desire to deny an attack asset to attackers. If a defender is only interested in denying a single goal vertex g then its personalization value $f(g)$ is set to 1 and all other vertices are set to 0.[8] If the defender desires to deny several vertices (for example, the `execCode` privilege on all servers) then the values will be set for the vertices in a manner that represents the defenders (conversely, the attackers) interest in those vertices. It is expected that the defender will set the personalization values based upon the organization's operational priorities.

[8] Technically, the non-goal vertices are set to an arbitrarily small $\epsilon > 0$ and the goal is set to $1 - (|V| - 1)\epsilon$. This ensures that the AssetRank matrix $A = D\Delta + \gamma Pe^T$ is positive, a condition that guarantees the existence of a unique positive eigenvector according to Perron's theorem.

4 Experiments

In this section we present several experiments we conducted to study 1) AssetRank's efficacy in giving results consistent with the importance of an attack asset to a potential attacker; and 2) how the AssetRank metric may be used to better understand security threats conveyed in a dependency attack graph, as well as in choosing appropriate mitigation measures.

In our experiments, we use the MulVAL attack-graph tool suite 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 Python module. The Python module normalizes the input data, computes the AssetRank values, and visualizes the attack graph using the graph visualization software Graphviz [13].

4.1 Experiment 1

The first experiment demonstrates the effect of arc weights and vertex-specific damping factors on a small network. Figure 3 shows the network for experiments 1a and 1b. The attacker has access to both PC1 and PC2. User1 is on PC1 which has vulnerability Vul1 and User2 is on PC2 which has vulnerability Vul2. PC1 and PC2 have access to the goal machine but not to each other.

In experiment 1a we assume that Vul1 has functional exploit tools available and Vul2 has only proof-of-concept code available. Hence, we assign success likelihood metrics of 0.8 and 0.4, respectively. A uniform damping factor of 0.99 is applied to all vertices. We expect that Vul1 will have a higher rank metric than Vul2 since the attacker is more likely to prefer it. Figure 4 shows the attack graph coloured according to the assets' AssetRank values. The vertex colours range from blue to red with blue indicating vertices with relatively lower ranks and red indicating vertices with higher ranks. Our algorithm computes a value of 0.0579 for Vul1 and a value of 0.0289 for Vul2 which is consistent with the higher value that Vul1 has to the attacker.

In experiment 1b we assign both Vul1 and Vul2 a success likelihood of 1.0. However, we assume that it is 80% likely that PC1 will be compromised by ways

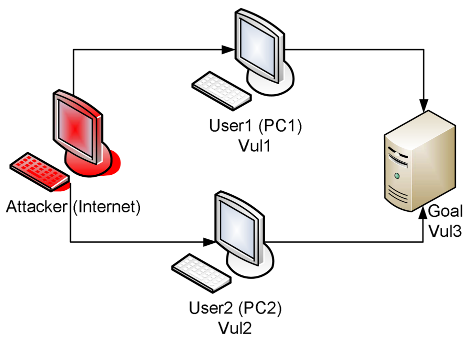


Fig. 3. Scenario for experiments 1a and 1b

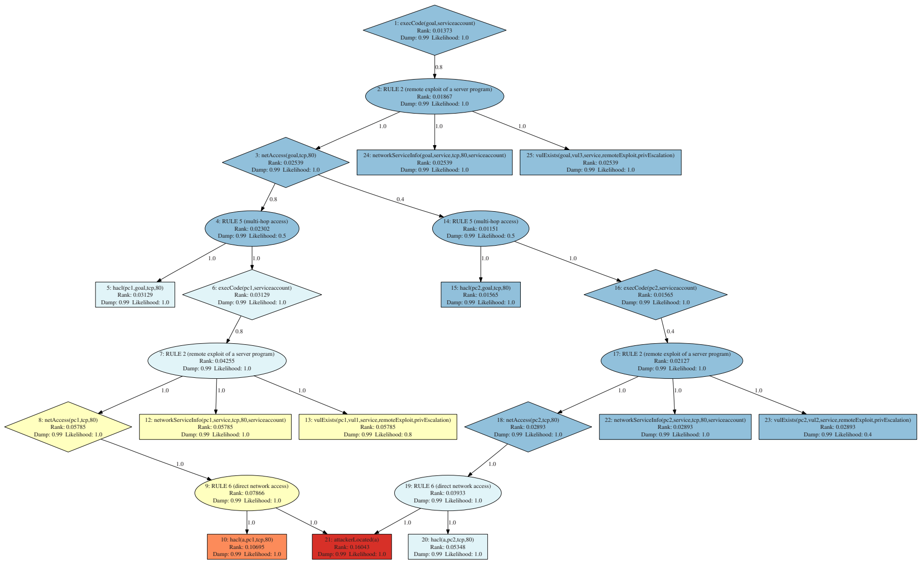

1: execCode(goal,serviceaccount) Rank: 0.01373 Damp: 0.99 Likelihood: 1.0
0.8
2: RULE 2 (remote exploit of a server program) Rank: 0.01867 Damp: 0.99 Likelihood: 1.0
1.0
3: netAccess(goal,tcp,80) Rank: 0.02539 Damp: 0.99 Likelihood: 1.0
24: networkServiceInfo(goal,service,tcp,80,serviceaccount) Rank: 0.02539 Damp: 0.99 Likelihood: 1.0
25: vulExists(goal,vul3,service,remoteExploit,privEscalation) Rank: 0.02539 Damp: 0.99 Likelihood: 1.0
0.8
0.4
4: RULE 5 (multi-hop access) Rank: 0.02302 Damp: 0.99 Likelihood: 0.5
14: RULE 5 (multi-hop access) Rank: 0.01151 Damp: 0.99 Likelihood: 0.5
5: hacl(pc1,goal,tcp,80) Rank: 0.03129 Damp: 0.99 Likelihood: 1.0
6: execCode(pc1,serviceaccount) Rank: 0.03129 Damp: 0.99 Likelihood: 1.0
15: hacl(pc2,goal,tcp,80) Rank: 0.01565 Damp: 0.99 Likelihood: 1.0
16: execCode(pc2,serviceaccount) Rank: 0.01565 Damp: 0.99 Likelihood: 1.0
7: RULE 2 (remote exploit of a server program) Rank: 0.04255 Damp: 0.99 Likelihood: 1.0
17: RULE 2 (remote exploit of a server program) Rank: 0.02127 Damp: 0.99 Likelihood: 1.0
8: netAccess(pc1,tcp,80) Rank: 0.05785 Damp: 0.99 Likelihood: 1.0
12: networkServiceInfo(pc1,service,tcp,80,serviceaccount) Rank: 0.05785 Damp: 0.99 Likelihood: 1.0
13: vulExists(pc1,vul1,service,remoteExploit,privEscalation) Rank: 0.05785 Damp: 0.99 Likelihood: 0.8
18: netAccess(pc2,tcp,80) Rank: 0.02893 Damp: 0.99 Likelihood: 1.0
22: networkServiceInfo(pc2,service,tcp,80,serviceaccount) Rank: 0.02893 Damp: 0.99 Likelihood: 1.0
23: vulExists(pc2,vul2,service,remoteExploit,privEscalation) Rank: 0.02893 Damp: 0.99 Likelihood: 0.4
9: RULE 6 (direct network access) Rank: 0.07866 Damp: 0.99 Likelihood: 1.0
19: RULE 6 (direct network access) Rank: 0.03933 Damp: 0.99 Likelihood: 1.0
10: hacl(a,pc1,tcp,80) Rank: 0.10695 Damp: 0.99 Likelihood: 1.0
21: attackerLocated(a) Rank: 0.16043 Damp: 0.99 Likelihood: 1.0
20: hacl(a,pc2,tcp,80) Rank: 0.05348 Damp: 0.99 Likelihood: 1.0

Fig. 4. Attack graph for the Experiment 1a scenario

not shown by the attack graph (for example, obtaining User1's log-in credentials through social-engineering), and PC2 is 40% likely to be compromised in such ways. Perhaps User2 has received more training and so is more security-vigilant than User1. We expect that Vul1 will be ranked lower than Vul2 since the attacker has a lower dependence upon it. Due to space constraints, we are not able to show the attack graph but Vul2 has an AssetRank of 0.0414 and Vul1 has an AssetRank of 0.0310. This ranking is intuitively correct since attackers have a greater chance of obtaining PC1 without exploiting its vulnerability, so Vul1 is less important to them.

4.2 Experiment 2

We now demonstrate the results of applying AssetRank to the attack graph for the example network in Figure 1. In the first scenario, we assume all the vulnerabilities have the same exploitability difficulty level, represented by identical success likelihood metrics.

The attack graph is not shown due to space constraints but a portion of the resulting ranking is shown in Table 1(a).[9] The ranking is consistent with the intuitive importance of the various attacker assets. Namely, vulnerabilities on

[9] In MulVAL, a tuple `vulExists(Host, VulID, Account, AccessVector, Consequence)` means "machine Host has the vulnerability VulID in software running as Account that is exploitable via AccessVector with the result Consequence." A tuple `hacl(H1, H2, Protocol, Port)` means "machine H1 can reach machine H2 through Protocol and Port."

Table 1. AssetRanks for Experiment 2

(a) Experiment 2a

Attack Asset	Rank
vulExists(c,vulid2, . . .)	0.0323
vulExists(d,vulid1, . . .)	0.0323
vulExists(e,vulid4, . . .)	0.0274
vulExists(f,vulid5, . . .)	0.0219
vulExists(b,vulid1, . . .)	0.0174
hacl(e,f,tcp,80)	0.0267
hacl(a,d,tcp,80)	0.0240
hacl(a,c,tcp,80)	0.0240
hacl(d,e,tcp,80)	0.0167
hacl(c,e,tcp,80)	0.0167
hacl(a,b,tcp,80)	0.0129

(b) Experiment 2b

Attack Asset	Rank
vulExists(d,vulid1, . . .)	0.0453
vulExists(e,vulid4, . . .)	0.0303
vulExists(f,vulid5, . . .)	0.0229
vulExists(b,vulid1, . . .)	0.0188
vulExists(c,vulid2, . . .)	0.0127
hacl(a,d,tcp,80)	0.0406
hacl(d,e,tcp,80)	0.0304
hacl(e,f,tcp,80)	0.0287
hacl(a,b,tcp,80)	0.0168
hacl(a,c,tcp,80)	0.0097
hacl(c,e,tcp,80)	0.0076

C and D are more important than the one on B, since these two machines are stepping stones into the right subnet. Likewise, the attacker's reachability to C and D is ranked higher than that to B.

Now suppose the vulnerability vulid2 on machine C is very difficult to exploit, and the other vulnerabilities are easy to exploit. We therefore assign the metric 0.2 to vulid2 and the other vulnerabilities a metric of 0.8. The result of the new configuration is given in Table 1(b).

What is remarkable in the new ranking is that the vulnerability on machine C is ranked much lower than before, since it is hard to exploit. Now machine D becomes much more valuable to the attacker since it is likely to be the only feasible stepping stone into the right subnet, which is manifested by the boosted values on both the vulnerabilities and reachability relations involving D. Note that the vulnerability on machine B is the same as the one on machine D. But since B cannot directly help the attacker penetrate deeper into the network, its vulnerability's rank is lower than that of D.

4.3 Experiment 3

To study how AssetRank works in a more complicated realistic setting, we tested it on a network scenario adapted from a real control-system network, shown in Figure 5. In this network, an enterprise network is protected by a firewall from the Internet. Only machines in the DMZ subnet can be directly accessed from the Internet zone. The machines in the CORP internal subnet can freely access the Internet. Only one machine in the network, the Citrix server, can access the control-system subnet (the Energy Management System, or EMS) which is protected by another firewall, and it may only access the Data Historian. Assuming the attacker is on the Internet and wants to obtain privileges on the Communications Servers in the EMS subnet, there are two obvious entry ways for him: the web server and the VPN server, both of which can be directly accessed from the Internet.

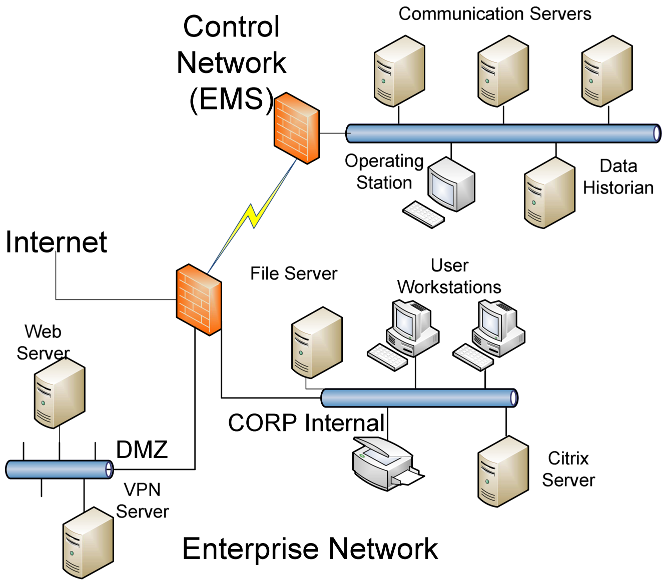


Fig. 5. A realistic network scenario for Experiment 3

We introduced hypothetical vulnerabilities into this scenario and assigned metrics for them based on our understanding of typical security problems in this type of network.[10] Due to space constraints we cannot show the attack graph; however, the ranking identifies the two most critical vulnerabilities in the network. One is a remote buffer overflow vulnerability on the web server, which would allow a remote attacker to gain code execution privilege in the DMZ subnet. The other is a browser vulnerability on the user workstation. Since outbound traffic from the CORP Internal zone is not restricted, an unsuspecting user may browse to a malicious website and compromise his machine. This compromise will yield privileges on the internal network to the attacker. There are many other vulnerabilities in the network and there are other ways to penetrate into the system (for example, through the VPN server). But the two critical problems identified by the AssetRank algorithm are consistent with a human's conclusion after spending an extensive amount of time studying the information revealed by the complicated 129 vertex attack graph with 185 dependencies.

5 Related Work

Attack graphs have been proposed and studied extensively to analyze the security of enterprise networks. There are basically two types of attack graphs. In the

[10] In real applications, this information will automatically be furnished by data collection agents installed on the machines and the CVSS metrics provided by the NVD.

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 [14], and the attack graph in Swiler and Phillips' work [15]. 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 work in this paper applies the extended PageRank algorithm, AssetRank, to distill and prioritize the information presented in a dependency attack graph.

Mehta *et al.* apply the PageRank algorithm to state enumeration attack graphs [16]. Aside from the generalizations 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 our source vertices are the attackers' goals, 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. But the probability a random attacker is in the goal state may decrease as the number of attack paths increases — simply because there are more states to split the distribution. As a result, contrary to what was proposed in their paper, this rank cannot serve as a metric for the system's overall vulnerability.

Recent years have seen a number of efforts that apply numeric security metrics to attack graphs. For example, Wang *et al.* studied how to combine individual security metrics to compute an overall security metric using attack graphs [17]. Dewri *et al.* proposed configuration optimization methods that are based on attack graphs, numeric cost functions, and genetic algorithms [18]. The goal of our work is different. We aim to use standardized security metrics and a unified algorithmic framework to rank and prioritize the security problems revealed by an attack graph.

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 [6,19]. 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. In the experiment in Section 4.2, 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 incorporate standardized security metrics such as CVSS, and compute the relative importance of each attack asset based on both the metrics and the attack graph. A binary result from the minimal-cut algorithm does not provide this capability, which we believe is important in realistic security management.

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 [7,8,9,20]. 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 coloring the visualization based on the computed ranks.

6 Conclusion

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 adds the ability to model various types of attackers. We have shown how to incorporate vulnerability attribute information into the arc weights. Similarly, users could compute attack asset ranks derived from metrics regarding attack noisiness, attack path length, or resource utilization. We have also shown how to model the existence of out-of-band attacks into vertex-specific damping weights. We incorporated personalization values to allow network defenders to specify the assets they most desire to deny attackers and thus obtain a personalized attack asset ranking based upon their operational priorities.

The numeric value computed by AssetRank is a direct indicator of how important the attack asset represented by a vertex is to a potential attacker. The algorithm was empirically verified through numerous experiments conducted on several example networks. The rank metric will be valuable to users of attack graphs in better understanding the security risks, in fusing publicly available attack asset attribute data, in determining appropriate mitigation measures, and as input to further attack graph analysis tools.

Acknowledgements

The authors thank Craig Burrell for many valuable discussions. We also thank the numerous people who provided helpful comments on the paper.

References

1. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of 9th ACM Conference on Computer and Communications Security, Washington, DC (November 2002)
2. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: 22nd Annual Computer Security Applications Conference (ACSAC), Miami Beach, Florida (December 2006)
3. Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: 19th Annual Computer Security Applications Conference (ACSAC) (December 2003)
4. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: 13th ACM Conference on Computer and Communications Security (CCS), pp. 336–345 (2006)
5. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: NSPW 1998: Proceedings of the 1998 workshop on New security paradigms, pp. 71–79. ACM Press, New York (1998)
6. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 254–265 (2002)
7. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: VizSEC/DMSEC 2004: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, pp. 109–118. ACM Press, New York (2004)
8. Noel, S., Jacobs, M., Kalapa, P., Jajodia, S.: Multiple coordinated views for network attack graphs. In: IEEE Workshop on Visualization for Computer Security (VizSEC 2005) (2005)
9. Lippmann, R., Williams, L., Ingols, K.: An interactive attack graph cascade and reachability display. In: IEEE Workshop on Visualization for Computer Security (VizSEC 2007) (2007)
10. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998)
11. Meyer, C.D.: Matrix analysis and applied linear algebra. Society for Industrial and Applied Mathematics. Philadelphia, PA, USA (2000)
12. Bianchini, M., Gori, M., Scarselli, F.: Inside PageRank. ACM Trans. Inter. Tech. 5(1), 92–128 (2005)
13. Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G.: Graphviz-Open Source Graph Drawing Tools. Graph Drawing, 483–485 (2001)
14. Sheyner, O.: Scenario Graphs and Attack Graphs. Ph.D thesis, Carnegie Mellon (April 2004)
15. Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DARPA Information Survivability Conference and Exposition (DISCEX II 2001), June 2001, vol. 2 (2001)

16. Mehta, V., Bartzis, C., Zhu, H., Clarke, E., Wing, J.: Ranking attack graphs. In: Proceedings of Recent Advances in Intrusion Detection (RAID) (September 2006)
17. Wang, L., Singhal, A., Jajodia, S.: Measuring network security using attack graphs. In: Third Workshop on Quality of Protection (QoP) (2007)
18. Dewri, R., Poolsappasit, N., Ray, I., Whitley, D.: Optimal security hardening using multi-objective optimization on attack tree models of networks. In: 14th ACM Conference on Computer and Communications Security (CCS) (2007)
19. Jha, S., Sheyner, O., Wing, J.M.: Two formal analyses of attack graphs. In: Proceedings of the 15th IEEE Computer Security Foundations Workshop, Nova Scotia, Canada, June 2002, pp. 49–63 (2002)
20. Homer, J., Varikuti, A., Ou, X., McQueen, M.A.: Improving attack graph visualization through data reduction and attack grouping. In: The 5th International Workshop on Visualization for Cyber Security (VizSEC) (2008)

Appendix

A Full Attack Graph for Example in Fig. 1.

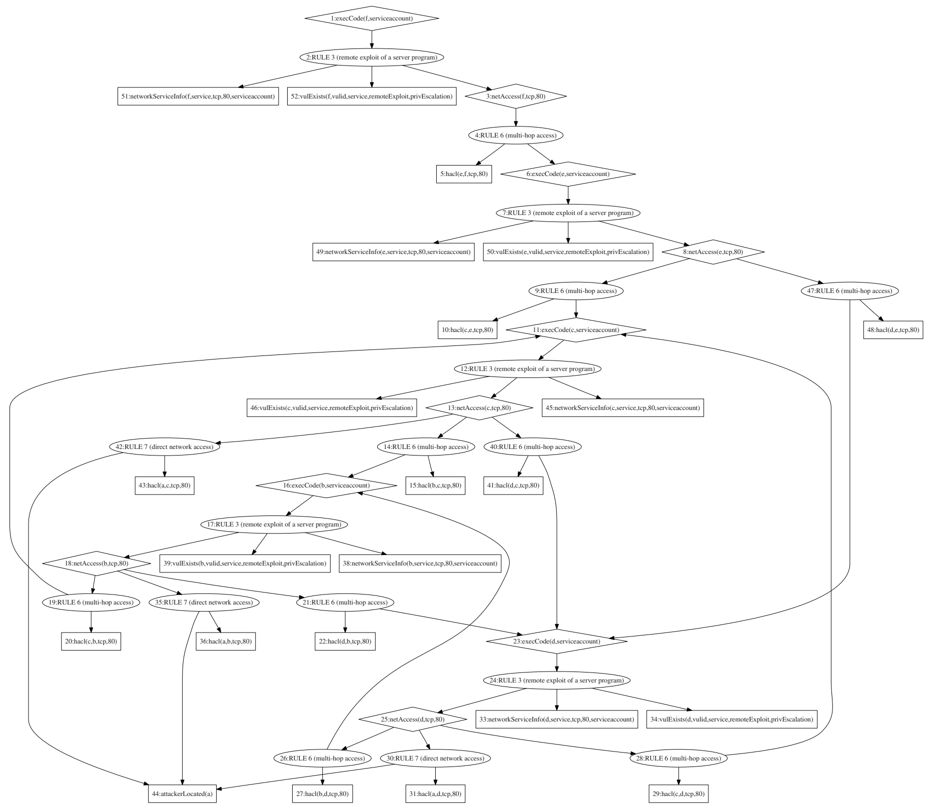

1:execCode(f,serviceaccount)
2:RULE 3 (remote exploit of a server program)
51:networkServiceInfo(f,service,tcp,80,serviceaccount)
52:vulExists(f,vulid,service,remoteExploit,privEscalation)
3:netAccess(f,tcp,80)
4:RULE 6 (multi-hop access)
5:hacl(e,f,tcp,80)
6:execCode(e,serviceaccount)
7:RULE 3 (remote exploit of a server program)
49:networkServiceInfo(e,service,tcp,80,serviceaccount)
50:vulExists(e,vulid,service,remoteExploit,privEscalation)
8:netAccess(e,tcp,80)
9:RULE 6 (multi-hop access)
47:RULE 6 (multi-hop access)
10:hacl(c,e,tcp,80)
11:execCode(c,serviceaccount)
48:hacl(d,e,tcp,80)
12:RULE 3 (remote exploit of a server program)
46:vulExists(c,vulid,service,remoteExploit,privEscalation)
13:netAccess(c,tcp,80)
45:networkServiceInfo(c,service,tcp,80,serviceaccount)
42:RULE 7 (direct network access)
14:RULE 6 (multi-hop access)
40:RULE 6 (multi-hop access)
43:hacl(a,c,tcp,80)
16:execCode(b,serviceaccount)
15:hacl(b,c,tcp,80)
41:hacl(d,c,tcp,80)
17:RULE 3 (remote exploit of a server program)
18:netAccess(b,tcp,80)
39:vulExists(b,vulid,service,remoteExploit,privEscalation)
38:networkServiceInfo(b,service,tcp,80,serviceaccount)
19:RULE 6 (multi-hop access)
35:RULE 7 (direct network access)
21:RULE 6 (multi-hop access)
20:hacl(c,b,tcp,80)
36:hacl(a,b,tcp,80)
22:hacl(d,b,tcp,80)
23:execCode(d,serviceaccount)
24:RULE 3 (remote exploit of a server program)
25:netAccess(d,tcp,80)
33:networkServiceInfo(d,service,tcp,80,serviceaccount)
34:vulExists(d,vulid,service,remoteExploit,privEscalation)
26:RULE 6 (multi-hop access)
30:RULE 7 (direct network access)
28:RULE 6 (multi-hop access)
44:attackerLocated(a)
27:hacl(b,d,tcp,80)
31:hacl(a,d,tcp,80)
29:hacl(c,d,tcp,80)

Fig. 6. Attack graph for the network in Figure 1