An Attack Graph-Based Probabilistic Security Metric

Lingyu Wang[1], Tania Islam[1], Tao Long[1], Anoop Singhal[2], and Sushil Jajodia[3]

[1] Concordia Institute for Information Systems Engineering
Concordia University
Montreal, QC H3G 1M8, Canada
{wang,t_is,ta_lon}@ciise.concordia.ca
[2] Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
anoop.singhal@nist.gov
[3] Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444, USA
jajodia@gmu.edu

Abstract. To protect critical resources in today's networked environments, it is desirable to quantify the likelihood of potential multi-step attacks that combine multiple vulnerabilities. This now becomes feasible due to a model of causal relationships between vulnerabilities, namely, attack graph. This paper proposes an attack graph-based probabilistic metric for network security and studies its efficient computation. We first define the basic metric and provide an intuitive and meaningful interpretation to the metric. We then study the definition in more complex attack graphs with cycles and extend the definition accordingly. We show that computing the metric directly from its definition is not efficient in many cases and propose heuristics to improve the efficiency of such computation.

1 Introduction

The traditional binary view of network security (that is, either secure or insecure) is becoming less and less suitable for today's increasingly complex networked environments. In practice, many vulnerabilities may still remain in a network after they are discovered, due to environmental factors (such as latency in releasing software patches or hardware upgrades), cost factors (such as money and administrative efforts required for deploying patches and upgrades), or mission factors (such as organizational preferences for availability and usability over security). To remove such residue vulnerabilities in the most cost-efficient way, we need to evaluate and measure the likelihood that attackers may compromise critical resources through cleverly combining multiple vulnerabilities.

The study of security metrics has recently drawn significant attention (a detailed review of related work is given in Section 5). However, existing network

metric standards typically focus on the measurement of individual vulnerabilities. For example, the Common Vulnerability Scoring System (CVSS) measures the potential impact and environmental metrics in terms of each individual vulnerability [13]. This is a major limitation, because the impact, damage, and relevance should be measured against potential compromises of critical resources, which typically require combining more than one vulnerability.

On the other hand, the causal relationships between vulnerabilities are well understood and usually encoded in the form of *attack graphs* [1, 24]. Attack graphs help to understand whether given critical resources can be compromised through multi-step attacks. However, as a qualitative model, attack graph still adopts a binary view towards security, that is, a network is either secure (critical resources are not reachable) or insecure. This is a limitation because it is usually desirable to find a relatively superior option among secure configurations.

Clearly, there is a gap between existing security metrics, which mostly focus on individual vulnerabilities, and qualitative models of vulnerabilities, which are usually limited to binary views of security. To fill this gap, we propose a probabilistic metric for measuring network security. The metric draws strength from both existing security metrics and the attack graph model. More specifically, we combine the measurements of individual vulnerabilities obtained from existing metrics into an overall score of the network. This combination is based on the causal relationships between vulnerabilities encoded in an attack graph. The key challenge lies in handling complex attack graphs with cycles. We first define the basic metric without considering cycles. We provide an intuitive interpretation of the metric. Based on such an interpretation, we extend the definition to attack graphs with cycles. Finally, we study the efficient computation of the metric. We show that computing the metric by using its definition is usually not efficient, and we provide heuristics for optimizing such computations.

The rest of the paper is organized as follows. Section 2 gives a motivating example. Section 3 defines the proposed metric and studies how to handle cycles in attack graphs. Section 4 presents heuristics for efficient computations of the metric. Section 5 reviews related work. Finally, Section 6 concludes the paper.

2 Attack Graph and Motivating Example

Attack graphs model how multiple vulnerabilities may be combined for advancing an intrusion. In an attack graph, security-related *conditions* represent the system state, and an *exploit* of vulnerabilities between connected hosts is modeled as a transition between system states. Figure 1 shows a toy example. The left side is the configuration of a network. Machine 1 is a file server behind the firewall that offers file transfer (ftp), secure shell (ssh), and remote shell (rsh)

services. Machine 2 is an internal database server that offers ftp and rsh services. The firewall allows ftp, ssh, and rsh traffic to both servers and blocks all other incoming traffic.

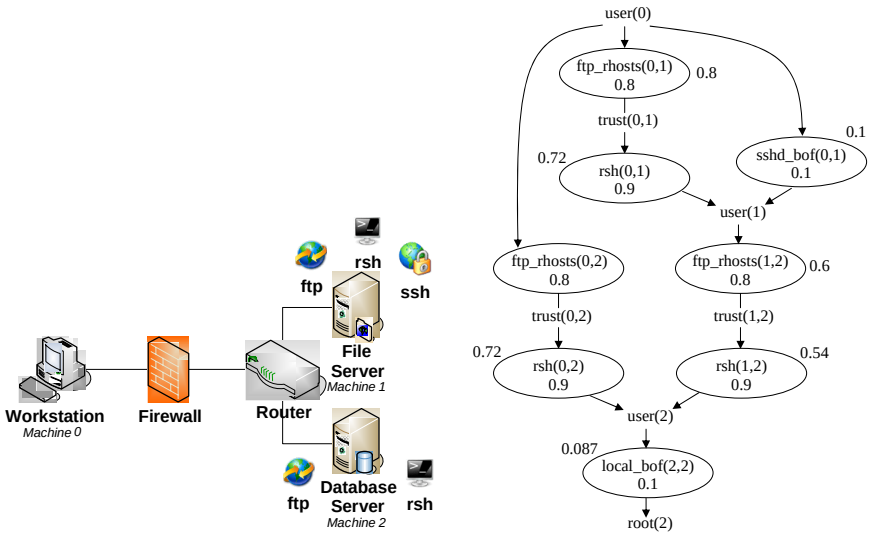

Fig. 1. An Example of Network Configuration and Attack Graph

The right-hand side of Figure 1 shows the attack graph (the numerical values are not part of the attack graph and will be explained shortly), which is a directed graph with two kinds of vertices, namely, exploits shown as predicates inside ovals and conditions shown in plaintexts. For example, $rsh(0,1)$ represents a remote shell login from machine 0 to machine 1, and $trust(0,1)$ means a trust relationship is established from machine 0 to machine 1. A directed edge from a condition to an exploit means executing the exploit requires the condition to be satisfied, and that from an exploit to a condition means executing the exploit will satisfy the condition. We formalize the attack graph in Definition 1.

Definition 1. *An attack graph G is a directed graph $G(E \cup C, R_r \cup R_i)$ where E is a set of exploits, C a set of conditions, and $R_r \subseteq C \times E$ and $R_i \subseteq E \times C$.*

The attack graph in Figure 1 depicts three *attack paths*. On the right, the attack path starts with an ssh buffer overflow exploit from machine 0 to machine 1, which gives the attacker the capability of executing arbitrary codes on machine 1 as a normal user. The attacker then exploits the ftp vulnerability on machine 2 to anonymously upload a list of trusted hosts. Such a trust relationship enables the attacker to remotely execute shell commands on machine 2 without providing a password. Consequently, a local buffer overflow exploit on machine

2 escalates the attacker's privilege to be the root of that machine. Details of the other two attack paths are similar and are omitted.

Informally, the numerical value inside each oval is a probability that indicates the relative likelihood of the corresponding exploit being executed by attackers when all the required conditions are already satisfied. This value thus only depends on each individual vulnerability, which is similar to many existing metrics, such as the CVSS [13]. On the other hand, we can clearly see the limitation of such metrics in assessing the impact, damage, or relevance of vulnerabilities, because such factors are rather determined by the combination of exploits. While we delay its definition and computation to later sections, the numerical value beside each oval represents the likelihood of reaching the corresponding exploit in this particular network. Clearly, a security administrator will be much happier to see the single score beside the last exploit ($local\_bof(2,2)$) than looking at all the eight values inside ovals and wondering how those values may be related to each other.

3 Defining the Metric

We introduce the metric in Section 3.1 for acyclic attack graphs. We discuss the need for conditions in the definition and provide an interpretation of the metric in Section 3.2. We then illustrate the issue raised by cycles in Section 3.3 and extend the definition to handle cycles in Section 3.4.

3.1 The Basic Definition

We first assume acyclic attack graphs and delay the discussion of cycles to Section 3.4. In this paper, we shall assume the events that an attacker can (and will) execute different exploits are independent and regard removing such an assumption as our future work. We only consider a fixed probability for measuring vulnerabilities, although other possibilities clearly exist (such as a probability distribution or a value varying in time).

We associate each exploit e and condition c with two probabilities, namely, $p(e)$ and $p(c)$ for the *individual score*, and $P(e)$ and $P(c)$ for the *cumulative score*. The individual score $p(e)$ stands for the intrinsic likelihood of an exploit e being executed, given that all the conditions required for executing e in the given attack graph are already satisfied. On the other hand, the cumulative score $P(e)$ and $P(c)$ measures the overall likelihood that an attacker can successfully reach and execute the exploit e (or satisfy the condition c) in the given attack graph (the individual score and cumulative score can also be interpreted as probabilities within a Bayesian network [8]).

For exploits, we assume the individual score is assigned based on expert knowledge about the vulnerability being exploited. For conditions, we assume in this paper that the individual score of every condition is always 1. Intuitively, a condition is either initially satisfied (for example, $user(0)$ in Figure 1), or immediately satisfied after a successful exploit (in practice, we can easily remove such assumptions by assigning less-than-1 individual scores to conditions). In Figure 1, we have assigned the individual scores (probabilities shown inside the ovals) based on simple facts, such as a buffer overflow attack requires more skills than executing a remote shell command. In practice, individual scores can be obtained by converting vulnerability scores provided by existing standards, such as the CVSS base score and temporal score [13], to probabilities.

Unlike individual scores, the cumulative score takes into accounts the causal relationships between exploits and conditions. In an attack graph, such causal relationships may appear in two different forms. First, a conjunction exists between multiple conditions required for executing the same exploit. Second, a disjunction exists between multiple exploits that satisfy the same condition. The cumulative scores are defined in the two cases similar to the probability of the *intersection* and *union* of random events. That is, if the execution of e requires two conditions c_1 and c_2, then $P(e) = P(c_1) \cdot P(c_2) \cdot p(e)$; if a condition c can be satisfied by either e_1 or e_2 (or both), then $P(c) = p(c)(P(e_1)+P(e_2)-P(e_1)\cdot P(e_2))$. Definition 2 formalizes cumulative scores.

Definition 2. *Given an acyclic attack graph $G(E \cup C, R_r \cup R_i)$, and any individual score assignment function $p : E \cup C \rightarrow [0, 1]$, the cumulative score function $P : E \cup C \rightarrow [0, 1]$ is defined as*

- $P(e) = p(e) \cdot \prod_{c \in R_r(e)} P(c)$
- $P(c) = p(c)$, *if* $R_i(c) = \phi$; *otherwise,* $P(c) = p(c) \cdot \oplus_{e \in R_i(c)} P(e)$ *where the operator* $\oplus$ *is recursively defined as* $\oplus P(e) = P(e)$ *for any* $e \in E$ *and* $\oplus(S_1 \cup S_2) = \oplus S_1 + \oplus S_2 - \oplus S_1 \cdot \oplus S_2$ *for any disjoint and non-empty sets* $S_1 \subseteq E$ *and* $S_2 \subseteq E$.

In Figure 1, the cumulative scores of two exploits (shown as plaintexts besides corresponding exploits) can be calculated as follows.

1. $P(rsh(0, 1)) = P(trust(0, 1) \times p(rsh(0, 1)) = 0.8 \times 0.9 = 0.72$
2. $P(user(1)) = P(rsh(0, 1))+P(sshd\_bof(0, 1))-P(rsh(0, 1))\times P(sshd\_bof(0, 1)) = 0.72 + 0.1 - 0.72 \times 0.1 = 0.748$

3.2 The Need for Conditions and an Interpretation of the Metric

From the above example, the score of conditions may seem rather unnecessary (as a matter of fact, we do not show the score of conditions in Figure 1). However, the attack graph shown in Figure 1 is a special case where all the causal

relationships between exploits happen to be disjunction only. In general, more complicated relationships may arise between exploits, and the cumulative score of conditions will be helpful in such cases. For example, Figure 2 shows the calculation of cumulative scores when a conjunctive, disjunctive, and hybrid relationship exists between exploits, respectively. It would be cumbersome to explicitly deal with such different relationships in defining our metric. However, as long as we include conditions as an intermediate between exploits, we can safely ignore the difference between those cases.

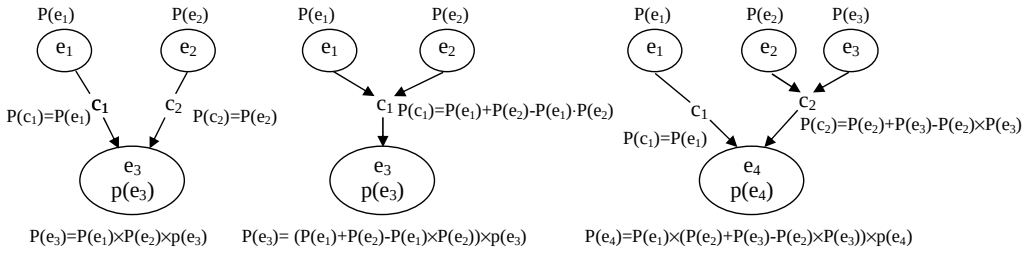

Fig. 2. Examples Showing the Need for Cumulative Scores of Conditions

Using probabilities for a security metric has been criticized as violating a basic design principle, that is, the value assignment should be specific and unambiguous rather than abstract and meaningless [22]. However, there is a simple interpretation for our metric. That is, the individual score $p(e)$ is the probability that any attacker can, and will execute e during an attack, given that all the preconditions are already satisfied. Equivalently, among all attackers that attempt to compromise the given network during any given time period, $p(e)$ is the fraction of attackers that can, and will execute e.

This interpretation of individual scores considers two factors in determining the individual score $p(e)$, namely, whether an attacker has the skills and resources to execute e and whether he/she will choose to do so. For example, a vulnerability that cannot be exploited remotely, or the one that requires a valid user account will likely have a lower score due to the first factor (that is, fewer attackers *can* exploit the vulnerability), whereas a vulnerability that can be easily detected, or the one less exposed to the public will likely have a lower score due to the second factor (that is, fewer attackers *will* exploit the vulnerability).

The interpretation of individual scores also provides a natural semantics to the cumulative scores. That is, $P(e)$ or $P(c)$ stands for the likelihood, or the fraction of, attackers who will successfully exploit e or satisfy c in the given network. The cumulative score of a given goal condition thus indicates the likelihood that a corresponding resource will be compromised during an attack, or

equivalently, among all attackers attacking the given network over a given time period, the average fraction of attackers who will successfully compromise the resource. Such a likelihood or fraction is clearly relevant in analyzing the security of a network or in hardening the network for better security.

3.3 Difficulties with Cycles

One complication in defining cumulative scores lies in the effect of cycles in attack graphs. Different types of cycles naturally exist in attack graphs, and they create different difficulties. Namely, some cycles can be completely removed; some cycles can be safely broken, some cycles, however, can neither be removed or broken. Figure 3 shows an example for each type of such cycles.

First, the left-hand side of Figure 3 shows a cycle that can be completely removed because none of the exploits or conditions inside the cycle can ever be reached by attackers. More specifically, executing the exploit e_1 requires both c_1 and c_3 to be satisfied. However, c_3 can only be satisfied by the execution of e_2, which again requires e_1 to be executed first. Therefore, neither e_1 nor e_2 can ever be successfully executed, and thus conditions c_2 and c_3 can never be satisfied. Such a *removable* cycle can be completely ignored during calculating the cumulative scores. In another word, all exploits and conditions inside the cycle have a cumulative score of zero (notice that c_4 thus automatically receives a cumulative score of zero by the definition given in Section 3.1).

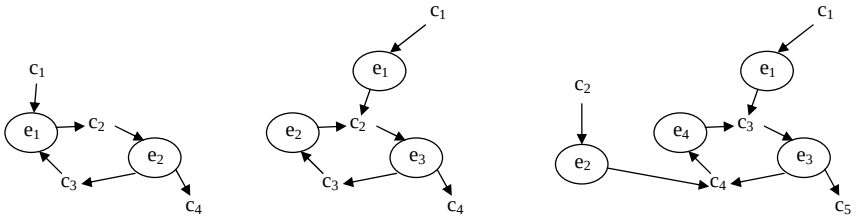

Fig. 3. Cycles in Attack Graphs

Second, the middle case of Figure 3 shows that some cycles cannot be removed because the exploits and conditions inside the cycle can indeed by reached. The condition c_2 can be satisfied by either e_1 or e_2. If c_2 is first satisfied by e_1, then both e_2 and e_3 can be successfully executed. Ignoring such a cycle will thus cause incorrect definition of the metric. Fortunately, this cycle can be easily broken by removing the directed edge from e_2 to c_2. Intuitively, c_2 is only satisfiable by e_1 even though later on it may be satisfied *again* by e_2 (we shall provide a clearer interpretation shortly). After we break the cycle in this way, the cumulative scores can then be easily calculated.

Third, the right-hand side of Figure 3 shows a cycle that can be neither removed nor broken in the aforementioned manner. Both e_1 and e_2 can lead to exploits in the cycle to be executed. There are thus two different ways for breaking the cycle among which we can only choose one. That is, we can either remove the edge from e_4 to c_3 by assuming c_3 is satisfied by e_1, or remove the edge from e_3 to c_4 by considering c_4 to be satisfied by e_2. However, there is no clear reason to prefer any of the two choices over the other. Moreover, removing both edges is clearly not a valid solution (the graph will be separated into two disjoint components). This example shows that removing or breaking a cycle is not a valid solution for all cases.

3.4 Extending the Definition to Handle Cycles

To find a general and meaningful solution, we need to revisit the aforementioned interpretation of the proposed metric. That is, an individual score represents the fraction of attackers who can (and will) execute an exploit or satisfy a condition (in the rest of the paper, we shall refer to both as *reaching*), given that all preconditions are already satisfied; a cumulative score indicates the fraction of attackers who will reach an exploit or a condition. However, when cycles are present in an attack graph, an attacker may reach an exploit or a condition more than once. Clearly, extra caution must be taken in calculating cumulative scores to avoid counting any attacker twice.

Without the loss of generality, we consider the calculation of $P(e_4)$ on the right-hand side of Figure 3. We illustrate those events using Venn diagrams in Figure 4 (the shaded areas can be ignored for now). Notice that the figure depicts cumulative scores for e_1 and e_2 as they are not part of the cycle. Referring to the right-hand side of Figure 3, we shall first calculate $P(e_4)$ by following the cycle clockwise from c_4, through e_4, c_3, e_3, and finally to c_4 again, as follows (each step simply follows the basic definition given in Section 3.1).

1. By abusing notations, denote $P(e_2)$ the set of attackers reaching e_2, represented as an oval in Figure 4.
2. Among the attackers in $P(e_2)$, those who can execute e_4 form the intersection $P(e_2) \cap p(e_4)$.
3. The union of two sets of attackers, $P(e_2) \cap p(e_4) \cup P(e_1)$, will reach c_3.
4. The intersection of the above set with $p(e_3)$, that is, $(P(e_2) \cap p(e_4) \cup P(e_1)) \cap p(e_3)$ will reach e_3.
5. Among those who reach e_3, the attackers originally coming from the set $P(e_2)$ should not be counted again towards satisfying c_4. In another word, only those in $(P(e_2) \cap p(e_4) \cup P(e_1)) \cap p(e_3) \setminus P(e_2) = P(e_1) \cap p(e_3) \setminus P(e_2)$ should be counted.

6. Finally, the set of attackers that can reach e_4 is $(P(e_1) \cap p(e_3) \setminus P(e_2) \cup P(e_2)) \cap p(e_4) = (P(e_1) \cap p(e_3) \cup P(e_2)) \cap p(e_4)$

The shaded area in the left-hand side of Figure 4 indicates the final result of $P(e_4)$. The right-hand side of Figure 4 corresponds to $P(e_3)$, which can be calculated similarly. In the above calculation, we essentially break the cycle in the second to the last step, by disregarding those attackers who reach c_4 for the second time. This is reasonable because in measuring the fraction of attackers (or the likelihood of an attacker) reaching c_4, we should only count the fraction of distinct attackers. In another word, although an attacker can repeat an exploit for many times, this should not affect the metric.

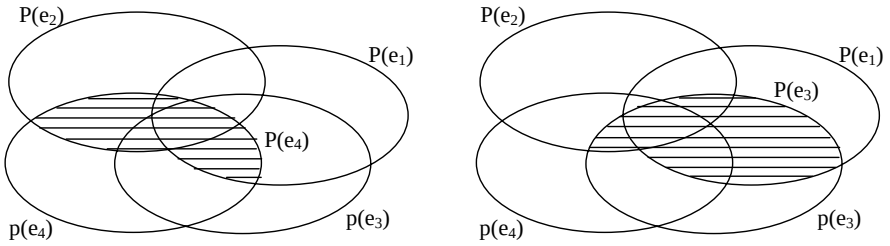

Fig. 4. Calculating Cumulative Scores in the Presence of Cycles

There is, however, an alternative interpretation of the final result $P(e_4) = (P(e_1) \cap p(e_3) \cup P(e_2)) \cap P(e_4)$. Intuitively, instead of breaking the cycle when some attackers reach c_4 for the second time, we prevent them from ever leaving e_4. More precisely, when calculating a cumulative score of an exploit (or a condition), we remove all outgoing edges from that exploit (or condition). After removing those edges, some other exploits or conditions may need to be removed if they can no longer be reached. The attack graph will needs to be updated to remove all unreachable vertices. For example, In the left-hand side of Figure 3, to calculate $P(e_1)$, we will remove the edge (e_1, c_2), (c_2, e_2), (e_2, c_3), (c_3, e_1), and finally the exploit e_1 itself (this can be interpreted as $P(e_1) = 0$).

Definition 3 formalizes cumulative scores for general attack graphs. In the definition, the first case corresponds to the exception where e is inside a removable cycle, so its cumulative score is defined (not computed) as zero. In the second case, the cumulative score is defined in $A(G, e)$ instead of G so to ensure that e is not inside any cycle and its cumulative score can thus be calculated based on Definition 2 (however, $A(G, e)$ is not guaranteed to be an acyclic attack graph so Definition 2 does not directly apply).

Definition 3. *Given an attack graph $G(E \cup C, R_r \cup R_i)$, and any individual score assignment function $p : E \cup C \to [0, 1]$, we denote $A(G, e)$ (or $A(G, c)$)*

an attack graph obtained by removing from G all the outgoing edges at e (or c) and consequently removing all unreachable exploits and conditions from G. The cumulative score function $P : E \cup C \to [0, 1]$ is defined as

- *If e (or c) does not appear in $A(G, e)$ (or $A(G, c)$), then $P(e) = 0$ (or $P(c) = 0$).*
- *Otherwise, $P(e)$ (or $P(c)$) is equal to its value calculated in $A(G, e)$ (or $A(G, c)$) based on Definition 2.*

Definition 3 satisfies two desirable properties as stated in Proposition 1. The first property guarantees that the cumulative score is defined for all exploits and conditions in the given attack graph G. The second property ensures that the extended definition is still consistent with the aforementioned interpretation of the metric.

Proposition 1. *By Definition 3, for any exploit e (the result applies to a condition c in a similar way),*

- *$P(e)$ can be uniquely determined, and*
- *$P(e)$ represents the likelihood of an attacker (or fraction of attackers) reaching e for the first time in the given attack graph G.*

Proof: We only discuss the case of an exploit e since a condition c is similar. First, e may be unreachable in G because it is inside a breakable cycle, such as the left-hand side of Figure 3. In this case, removing all outgoing edges from e will essentially cause the cycle to be completely removed. We interpret this as $P(e) = 0$, which indicates that no attacker can reach e in G.

Suppose e is reachable. We prove the first claim through induction on the number k of exploits that can be reached before reaching e (we ignore those exploits that are not on any attack path that contains e so any exploit can either be reached before e, or after it). Clearly, for $k = 0$, $P(e) = p(e)$. Suppose the claim holds for any $k - 1$. We consider an exploit e before which k others can be reached. Before any of those k exploits, at most $k - 1$ others can be reached (otherwise, there would be more than k exploits reachable before e), and hence the claim holds for the k exploits that can be reached before e. For exploits that can only be reached after reaching e at least once (notice those exploits could be within a cycle containing e), they will not appear in $A(G, e)$. The claim thus holds for e, because e is no longer in any cycle and the claim already holds for all the k remaining exploits in $A(G, e)$ (except e itself).

For the second claim, it suffices to show that any attacker can reach e for the first time in G iff it can do so in $A(G, e)$. The if part is true because any valid attack path in $A(G, e)$ will also exist in G. The only-if part holds because when

we update the attack graph, only those exploits that can only be reached after reaching e are removed, so their removal will not affect any attack path from reaching e in $A(G, e)$, if such a path exists in G.

□

4 Computing the Metric

Cumulative scores can certainly be computed directly by Definition 3. That is, for each exploit e (similarly for each condition), we first compute $A(G, e)$ and then compute $P(e)$ on $A(G, e)$. However, this naive approach is inefficient because the removal of outgoing edges (and subsequent updates of attack graphs) is only necessary for vertices inside cycles. More specifically, if a vertex v is not part of a cycle, then on each attack path including v, all vertices can be divided into two sets, namely, the predecessors and successors of v (on the other hand, if v is inside a cycle, then every vertex in the cycle is both a predecessor and a successor of v). Removing outgoing edges from v will cause all successors of v to be removed as well, whereas all the predecessors of v are not affected. Therefore, in Definition 3, if v is not part of a cycle, then calculating the cumulative score of v in $A(G, e)$ will give the same result as calculating it in G.

For vertices not inside a cycle, we use a modified breadth-first search (BFS) to calculate cumulative scores. A normal BFS follows the outgoing edges of a vertex only when it is reached for the first time (cycles are always broken at a vertex with the maximum shortest-path distance from some initially satisfied conditions). However, such a normal BFS is not suitable for calculating our metric. In Definition 3, the cumulative score of any vertex can be calculated only when the cumulative scores of all its predecessors have already been calculated. We thus modify the BFS such that a vertex *receives* a cumulative score from its predecessor, whenever the former is reached from the latter. The outgoing edges of a vertex are followed only when the vertex is reached for the last time (this can be implemented using a counter based on the in-degree of each vertex). At this time, the vertex will have received cumulative scores from all its predecessors, and its own cumulative score can thus be calculated.

The above procedure, however, will terminate upon reaching one or more cycles. Referring to the right-hand side of Figure 3, the search will reach c_3 from e_1 (or, reach c_4 from e_2) and then stop there, because the predecessor e_3 of c_4 (or, the predecessor e_4 of c_3) has not yet been, and will never be, reached. Notice that this is true no matter a cycle is removable (such as in the left-hand side of Figure 3) or not. This termination of the procedure is actually desirable, because it signals that one or more cycles have been reached. It also indicates *entry points* of the cycle, that is, those vertices that have at least one of their pre-

decessors reached, such as c_4 and c_3 in the right-hand side of Figure 3. Upon the termination of the main procedure, a sub-procedure will mark vertices inside the encountered cycle(s), and then calculate their cumulative scores by Definition 3.

One subtlety is that the cumulative score of all entry points of a cycle must be calculated strictly by applying Definition 3. This may not seem to be necessary, since once the calculation is finished for one vertex in the cycle, it is possible to continue without considering the cycle. For example, in the right-hand side of Figure 3, once we calculate $P(c_3)$ by applying Definition 3, $P(e_3)$ can be calculated as $P(e_3) = P(c_3) \cdot p(e_3)$, and $P(c_4) = P(e_3) + P(e_2) - P(e_3) \cdot P(e_2)$. However, while $P(e_3)$ can indeed be calculated this way, the calculation of $P(c_4)$ is incorrect. Figure 5 shows $A(G, c_3)$, $A(G, e_3)$, and $A(G, c_4)$ obtained by Definition 3. Clearly, $P(e_3)$ can be calculated from $P(c_3)$ as $P(e_3) = P(c_3) \cdot p(e_3)$, but the calculation of $P(c_4)$ cannot be based on $P(e_3)$. Actually, it can be observed that $P(c_3)$ and $P(e_3)$ both depend on the individual score $p(e_4)$, whereas $P(c_4)$ has nothing to do with $p(e_4)$.

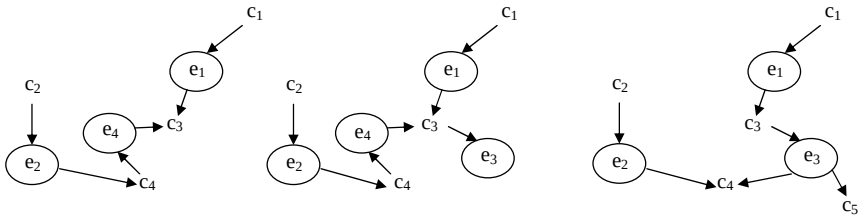

Fig. 5. Computing the Metric for Vertices in Cycles

It is certainly a viable solution to always calculate cumulative scores for all vertices in a cycle by applying Definition 3. However, for those vertices that are not entry points of a cycle, it will be more efficient to calculate their cumulative scores without considering the cycle. In the above example, $P(e_3)$ can be safely calculated from $P(c_3)$. More generally, for any vertex in a cycle that has only one incoming edge, the cumulative score can be safely calculated (after calculation of entry points is finished) as if the vertex is not in the cycle. The reason is as follows. The only incoming edge of such a vertex v must be part of the cycle. Let w be the predecessor of v. When we calculate $P(w)$, we will remove v together with all its successors to obtain $A(G, w)$. When we calculate $P(v)$, we also remove all predecessors to obtain $A(G, v)$. The only difference between the two cases is thus v itself. That is, $P(v) = P(w) \cdot p(v)$ is true. However, if v has more than one incoming edge (and hence multiple predecessors), then to calculate $P(w)$, v may not need to be removed. For example, we do not remove c_4 while calculating $P(e_3)$ in the middle case of Figure 5.

Figure 6 shows an algorithm for calculating the metric in a given attack graph with all individual scores assigned. Lines 1-2 assign the cumulative score of initially satisfied conditions as 1. The main loop between line 3 and line 9 calculates the cumulative score for all other vertices. Each loop is separated into three phases. First, lines 4-5 employs the aforementioned modified BFS to calculate cumulative scores until one or more cycle is encountered and the search terminates. Second, lines 6-7 applies Definition 3 to calculate cumulative scores for vertices in cycles that have more than one incoming edges. Finally, lines 8-9 calculate cumulative scores for other vertices in the cycle in a simpler way (as if they are not in any cycle). After all vertices in the encountered cycles are processed, the main loop will repeat lines 3-9, until all vertices are processed.

Input: An attack graph G with individual scores assigned to all vertices
Output: A set of cumulative scores for all vertices of G
Method:
1. **For** each initially satisfied condition c
2. **Let** $P(c) = 1$ and mark c as processed
3. **While** there exist unprocessed vertices
4. **While** there exists an unprocessed vertex v whose predecessors are all processed
5. **Calculate** $P(v)$ and mark v as processed
6. **For** each vertex v' in a cycle that has more than one incoming edge
7. **Calculate** $P(v')$ and mark v' as processed
8. **For** each unprocessed vertex v'' in the cycles
9. **Calculate** $P(v'')$ and mark v'' as processed
10. **Return** the set of all calculated cumulative scores

Fig. 6. Algorithm for Calculating the Metric

5 Related Work

General reviews of security metrics are given in [11, 2]. The NIST's efforts on standardizing security metrics are in the Technology Assessment: Methods for Measuring the Level of Computer Security [14] and more recently in the Security Metrics Guide for Information Technology Systems [25]. The latter describe the current state of practice of security metrics, such as that required by the Federal Information Security Management Act (FISMA). Another overview of many aspects of network security metrics is given in [9].

Dacier et al. give intuitive properties that should be satisfied by any security metric [5, 6, 16]. The difficulty of attacks are measured in terms of time and efforts spent by attackers. Based on an exponential distribution for an attacker's success rate over time, they use a Markov model and the MTTF (Mean Time to Failure) to measure the security of a network. They discuss simple cases of combining individual measures but do not study the general case. Standardization

efforts for vulnerability assessment include the Common Vulnerability Scoring System (CVSS) [13], although these generally treat vulnerabilities in isolation, without considering attack interdependencies on target networks. More recently, we propose an attack resistance metric based on attack graph in [31, 30]. In this paper, we adopt a probabilistic approach and tackle challenging issues, such as cycles in attack graphs, that are not addressed elsewhere.

The work by Balzarotti et al. [3] focuses on computing the minimum efforts required for executing each exploit. Based the exploitability concept, a qualitative measure of risk is given in [4]. Another approach measures the relative risk of different configurations using the *weakest attacker* model, that is the least conditions under which an attack is possible [20]. Yet another series of work measures how likely a software is vulnerable to attacks using a metrics called *attack surface* [10, 17–19, 12]. These work allow a partial order to be established on different network configurations based on their relative security. However, the treatment of many aspects of security is still qualitative in nature.

Our work on minimum-cost network hardening is one of the first efforts toward the quantitative study of network security [15, 29]. This work quantifies the cost of removing vulnerabilities in hardening a network, but it does not consider other hardening options, such as modifying the connectivity. It also has the limitation of adopting a qualitative view of damages (all the given critical resources are equally important) and of attack resistance (attacks on critical resources are either impossible or trivial).

To generate attack graphs, topological vulnerability analysis enumerates potential multi-step intrusions based on prior knowledge about vulnerabilities and their relationships [5, 7, 16, 21, 33, 26]. Based on whether a search starts from the initial state or the final state, such analyses can be forward [21, 26] or backward [23, 24]. To avoid the exponential explosion in the number of explicit attack sequences, a compact representation of attack graphs was proposed based on the *monotonicity assumption* saying an attacker never needs to relinquish any obtained capability [1]. On the attack response front, attack graphs have been used for the correlation of attacks, the hypotheses of alerts missed by IDSs, and the prediction of possible future attacks [27, 28, 32].

6 Conclusion

While removing all vulnerabilities is usually impractical, leaving vulnerabilities unattended may cause significant damages to critical resources in a networked environment. It is thus critical to understand and measure the likelihood of sophisticated attacks combining multiple vulnerabilities for reaching the attack goal. We have proposed an attack graph-based probabilistic metric for this pur-

pose. We have tackled challenging issues, such as cycles in attack graphs. We showed that the definition of the metric has an intuitive and meaningful interpretation, which will be helpful in real world decision making. Future work will implement a practical tool to measure security risk of enterprise networks.

Acknowledgements This material is based upon work supported by National Institute of Standards and Technology Computer Security Division; by Homeland Security Advanced Research Projects Agency under the contract FA8750-05-C-0212 administered by the Air Force Research Laboratory/Rome; by Army Research Office under grant W911NF-05-1-0374, by Federal Aviation Administration under the contract DTFAWA-04-P-00278/0001, by the National Science Foundation under grants CT-0627493, IIS-0242237 and IIS-0430402, by Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035, and by Fonds de recherche sur la nature et les technologies. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring organizations. The authors thank the anonymous reviewers for their valuable comments.

References

1. P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, 2002.
2. A. C. S. Associates. Workshop on. In *Information Security System Scoring and Ranking*, 2001.
3. D. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 1st Workshop on Quality of Protection*, 2005.
4. P. Balzarotti, M. Monga, and S. Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 2nd ACM workshop on Quality of protection*, 2005.
5. M. Dacier. Towards quantitative evaluation of computer security. Ph.D. Thesis, Institut National Polytechnique de Toulouse, 1994.
6. M. Dacier, Y. Deswarte, and M. Kaaniche. Quantitative assessment of operational security: Models and tools. Technical Report 96493, 1996.
7. D. Farmer and E. Spafford. The COPS security checker system. In *USENIX Summer*, pages 165–170, 1990.
8. M. Frigault and L. Wang. Measuring network security using bayesian network-based attack graphs. In *Proceedings of The 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*, 2008.
9. K. Hoo. Metrics of network security. White Paper, 2004.
10. M. Howard, J. Pincus, and J. Wing. Measuring relative attack surfaces. In *Workshop on Advanced Developments in Software and Systems Security*, 2003.
11. A. Jaquith. *Security Merics: Replacing Fear Uncertainity and Doubt*. Addison Wesley, 2007.
12. K. Manadhata, J. Wing, M. Flynn, and M. McQueen. Measuring the attack surfaces of two ftp daemons. In *Quality of Protection Workshop*, 2006.

13. P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy Magazine*, 4(6):85–89, 2006.
14. National Institute of Standards and Technology. Technology assessment: Methods for measuring the level of computer security. NIST Special Publication 500-133, 1985.
15. S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency grpahs. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, 2003.
16. R. Ortalo, Y. Deswarte, and M. Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Trans. Software Eng.*, 25(5):633–650, 1999.
17. J. W. P. Manadhata. Measuring a system's attack surface. Technical Report CMU-CS-04-102, 2004.
18. J. W. P. Manadhata. An attack surface metric. Technical Report CMU-CS-05-155, 2005.
19. J. W. P. Manadhata. An attack surface metric. In *First Workshop on Security Metrics (MetriCon)*, 2006.
20. J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38, New York, NY, USA, 2006. ACM Press.
21. C. Phillips and L. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the New Security Paradigms Workshop (NSPW'98)*, 1998.
22. M. Reiter and S. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information and System Security*, 2(2):138–158, 5 1999.
23. R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Research on Security and Privacy (S&P'00)*, pages 156–165, 2000.
24. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02)*, 2002.
25. M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo. Security metrics guide for information technology systems. NIST Special Publication 800-55, 2003.
26. L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX'01)*, 2001.
27. L. Wang, A. Liu, and S. Jajodia. An efficient and unified approach to correlating, hypothesizing, and predicting intrusion alerts. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS 2005)*, pages 247–266, 2005.
28. L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, 2006.
29. L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 11 2006.
30. L. Wang, A. Singhal, and S. Jajodia. Measuring network security using attack graphs. In *Proceedings of the 3rd ACM workshop on Quality of protection (QoP'07)*, New York, NY, USA, 2007. ACM Press.
31. L. Wang, A. Singhal, and S. Jajodia. Measuring the overall security of network configurations using attack graphs. In *Proceedings of 21th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2007)*, 2007.
32. L. Wang, C. Yao, A. Singhal, and S. Jajodia. Interactive analysis of attack graphs using relational queries. In *Proceedings of 20th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2006)*, pages 119–132, 2006.
33. D. Zerkle and K. Levitt. Netkuang - a multi-host configuration vulnerability checker. In *Proceedings of the 6th USENIX Unix Security Symposium (USENIX'96)*, 1996.