Merging Sub Evidence Graphs to an Integrated Evidence Graph

[*]Changwei Liu, [§]Anoop Singhal and [*]Duminda Wijesekera
cliu6@gmu.edu, anoop.singhal@nist.gov, dwijesek@gmu.edu
[*]Department of Computer Science, George Mason University, Fairfax VA 22030
[§]National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg MD 20899

ABSTRACT

Evidence graphs model network intrusion evidence and their dependencies, which helps network forensics analysts collate and visualize dependencies. In particular, probabilistic evidence graphs provide a way to link probabilities associated with different attack paths with available evidence. Existing work in evidence graphs assumes that all evidence is available in a single evidence graph. In this paper, we demonstrate how to merge multiple evidence graphs into a single evidence graph. We show this by providing algorithms and a possible scenario based on attacks on a fileserver and a database server in an example network environment. An integrated evidence graph, showing all attacks using *global* reasoning, is more useful to forensics analysts as compared to multiple evidence graphs that use *local* reasoning.

Keywords—Probabilistic evidence graphs; evidence and host probabilities; integrated probabilistic evidence graphs; Forensics

I. INTRODUCTION

Evidence graphs and attack graphs have been used in security analysis. Evidence graphs are generally used to help network forensics analysis in a multi-step, multi-host enterprise network by modeling intrusion evidence in the network. In such a graph, nodes represent host computers that interest forensics investigators and edges represent dependencies between evidence [1]. Evidence graphs help with forensic analysis because they are built from attacks that occurred in a specific network environment.

Attack graphs are used to analyze security vulnerabilities and their dependencies in enterprise networks. In particular, attack graphs with quantitative metrics, known as probabilistic attack graphs, estimate probabilities of attack success in networks [2]. While a good probabilistic attack graph can provide potential attack success probabilities to help forensics investigation and network configuration adjustment, an attack graph with inaccurate attack paths or attack success probabilities may mislead investigators or network administrators. As a solution to adjustment of an inaccurate probabilistic attack graph, [4] formally defines probabilistic evidence graphs and uses a mapping algorithm to map probabilistic evidence graphs to probabilistic attack graphs. The mapping algorithm stated in [4] works well in mapping an evidence graph to a small-scaled attack graph, but it does not show how to merge multiple evidence graphs into one integrated probabilistic evidence graph. The paper does not discuss how to map multiple evidence graphs to one large-scale attack graph. The reason for having multiple attack graphs is clear--namely evidence is collected from multiple systems, and investigators have to somehow join them together to get a complete view of the evidence trail left over by a distributed attack launched by multiple attackers. To the best of our knowledge, published literature does not address the issue of integrating evidence graphs, which is the main contribution of our paper.

The rest of this paper is organized as follows. Section II describes related work and definitions. Section III discusses how to integrate two probabilistic evidence graphs together. Section IV uses a possible scenario to show how to do integration of evidence graphs, and finally section V provides our conclusions.

II. RELATED WORK AND DEFINITIONS

Many papers define attack graphs and evidence graphs. [12] and [13] define state-based attack graphs where nodes are global states of the system and edges are state transitions. However, these state-based attack graphs create exponentially many states because the definition encodes nodes as a collection of Boolean variables to cover the entire network states. In order to solve this problem, a compact representation of attack graphs is defined in [3,14,15], where nodes represent exploits or conditions and edges represent attack dependencies. [1] defines an evidence graph as a graph in which nodes represent host computers involved in attacks and edges represent preprocessed forensics evidence that correlates those hosts.

NVD from NIST [9] standardizes vulnerability metrics that assign success probabilities to exposed individual vulnerabilities. They are used in attack graphs to compute success probabilities of attacks that exploit a series of vulnerabilities [3,16]. Similarly, evidence graphs also use quantitative metrics to estimate the admissibility of evidence and the certainty of the host being involved in a specific attack [1]. We use the following definitions to define evidence graphs and logical attack graphs [1,3].

Definition 1 (Evidence Graph) [1,4]: An evidence graph is a sextuple G=(N,E,N-Attr,E-Attr,L,T), where N is a set of nodes representing host computers, $E \subseteq (N_i \times N_j)$ is a set of directed edges consisting of a particular data item indicating of activities between source and target machines, N-Attr is a set of node attributes that include host ID, attack states, time stamp and host importance, and E-Attr is a set of edge attributes consisting of event description, evidence impact weight, attack relevancy and host importance. Functions $L:N \rightarrow 2^{N\text{-}Attr}$ and $T:E \rightarrow 2^{E\text{-}Attr}$ assign attribute-value pairs to a node and an edge respectively.

In this definition, attack states are one or many of attack source, target, stepping-stone, and affiliated host computers. Affiliated hosts have suspicious interactions with an attacker, one of victim hosts or stepping-stone hosts [4].

Definition 2 (Probabilistic Evidence Graph) [4]: In an evidence graph G=(N,E,N-Attr,E-Attr,L,T),

the probability assignment functions $p \in [0,1]$ for an evidence edge e and a victim host h are defined as follows.

a. **$p(e)=c \times w(e) \times r(e) \times h(e)$**, where w, r and h are the weight, relevancy and the importance [1] of the evidence edge e. Coefficient "c" indicates the categories of evidence, which include primary evidence, secondary evidence and hypothesis testing from expert knowledge. As an example, 1, 0.8 and 0.5 are respectively assigned to them in this paper.

b. **$p(h)=p[(\cup e_{out}) \cup (\cup e_{in})]$**, where $\cup e_{out}$ means all edges whose source computer is host h with a particular attack-related state in states mentioned in Definition 1, and $\cup e_{in}$ represents all edges whose target computer is h with the same state.

The weight of an evidence edge with a value between [0, 1] represents the impact of the evidence on the attack. Relevancy is the measure of the evidence impact on the attack success at this specific step, which consists of three values--an irrelevant true positive is 0, unable to verify is 0.5 and relevant true positive is 1. Lastly, the importance of the evidence with a value between [0,1] takes the bigger importance value of two hosts connected by the evidence edge.

There are two kinds of evidence used to construct evidence graphs. While primary evidence is explicit and direct, secondary evidence is implicit or circumstantial with a lower coefficient. In addition, in some circumstances where the evidence does not exit or is destroyed, a subject matter expert may insert an edge as the expert opinion. We assign it an even lower value because of its "non evidence" nature.

Definition 3 (Logical Attack graph) [3,4]: $A=(N_r,N_p,N_d,E,L,G)$ is a logical attack graph, where N_r, N_p and N_d are three sets of nodes in the graph that are called derivation, primitive and derived fact nodes respectively, $E \subseteq ((N_p \cup N_d) \times N_r) \cup (N_r \times N_d)$, L is a mapping from a node to its label, and $G \subseteq N_d$ is an attacker's final goal [3].

A tool named MulVAL[5] can be used to generate a logical attack graph, as illustrated in Figure 1. In this graph, a *primitive fact node* shown as a box represents a specific network configuration or

vulnerability information of a host; A *derivation node* shaped as an ellipse represents a successful application of an interaction rule on input facts including primitive facts and prior derived facts, which results in a *derived fact* node that is satisfied by these input facts. The derived fact is represented by a diamond.

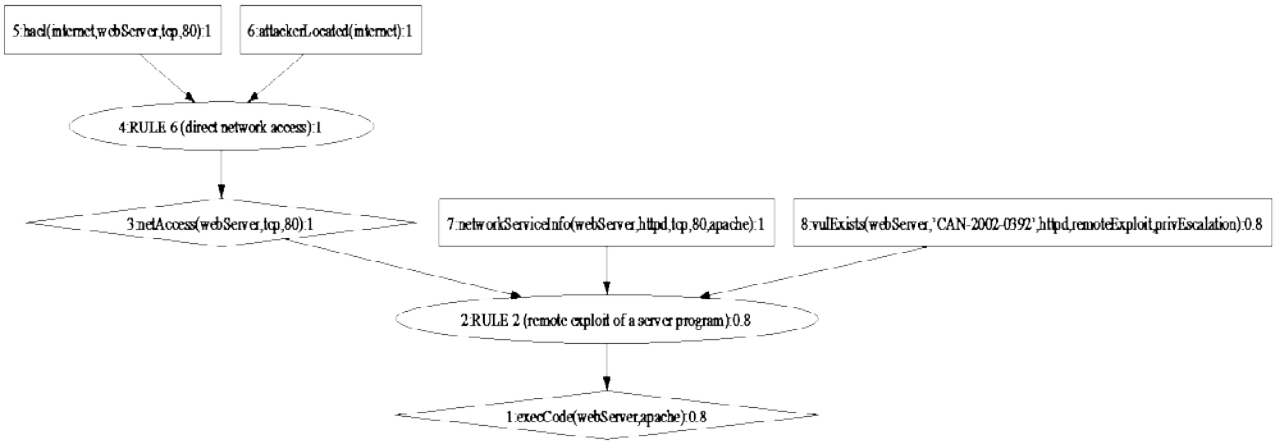

Figure 1: An Example Attack Graph

Definition 4(Cumulative Probability Function)[4]: Suppose $A=(N_r,N_p,N_d,E,L,G,p)$ is a probabilistic attack graph where the function $p:N_p \cup N_r \cup N_d \rightarrow [0,1]$ that assigns probabilities to nodes. If we use exploits to represent derivation nodes N_r and conditions to represent primitive facts N_p or derived facts N_d, the *cumulative probability function* P for exploits and conditions of a probabilistic attack graph $P:N_p \cup N_r \cup N_d \rightarrow [0,1]$ is computed from the probability assignment function $p:N_p \cup N_r \cup N_d \rightarrow [0,1]$ as follows.

1. **P(c)= p(c)** if the condition is a primitive fact.
2. **P(c)= p(c)**$\times \oplus$**P(e)** c is the derived fact node that is derived from derivation nodes(e) and primitive facts(c).
3. **P(e)=p(e)**$\times \Pi${**P(c)**} where $e \in N_r$, and c are conditions that include primitive facts and derived facts.

In clause 1, p(c) is always equal to 1, since a condition $c \in N_P$ or $c \in N_d$ is always satisfied as a primitive or derived fact. In clause 2, the probability law applies to $\oplus$P(e).

Now we define sub attack graphs and similar nodes that will be used in this paper. Notice that a sub evidence graph is an evidence graph, which has been defined in Definition 1.

Definition 5(Sub Logical Attack Graph): A logical attack graph $A'=(N'_r,N'_p,N'_d,E',L',G')$, is a sub logical attack graph of a complete logical attack graph $A=(N_r,N_p,N_d,E,L,G)$, iff

1) $N'_r,N'_p,N'_d \subseteq N_r,N_p,N_d$, and
2) $E'\subseteq E \wedge ((N_1, N_2)\in E' \rightarrow N_1, N_2\in N'_r\cup N'_p\cup N'_d$, and
3) $G'\subseteq G$, and
4) $L' \subseteq L$

Definition 6(Similar nodes): In a logical attack graph $A=(N_r,N_p,N_d,E,L,G)$ or an evidence graph $G=(N,E,N\text{-}Attr,E\text{-}Attr,L,T)$,

(1) If both N_{1d} and $N_{2d} \in A$, and satisfy the equalities of $N_{1d} == N_{2d}$, $N_{1r} == N_{2r}$, $N_{1p} == N_{2p}$, we say that N_{1d} and N_{2d} are similar, which can be represented as $N_{1d} \approx N_{2d}$.

(2) If both N_1 and $N_2 \in G$ (here G means an evidence Graph), the attack status of N_1 is equal to the attack status of N_2, and $E(N_1) == E(N_2)$, where $E(N_1)$ and $E(N_2)$ are the evidence edge whose source or destination host is N_1 and N_2 respectively, then we say N_1 and N_2 are similar , which can be represented as $N_1 \approx N_2$.

Because In order to generate an evidence graph, [1] suggests normalizing all evidence including primary evidence and secondary evidence to five components (1) id, (2) source, (3) destination, (4) content and (5) time stamp, which are used as edges to connect hosts in a time order forming an evidence graph. For refining purposes, a reasoning mechanism and hypothesis testing are used to find correlated hosts of an attack to construct a complete evidence graph. There are tools for an attack graph generation, of which MulVAL is used to generate a logical attack graph [9] in this paper.

Based on an evidence graph and its corresponding attack graph, the mapping algorithm described in [4] uses depth first searching algorithm to search for corresponding attack paths in both graphs, aiming to adjust any inaccurate attack path or attack success probability in the attack graph.

III. INTEGRATING SUB EVIDENCE GRAPHS

For every attack, investigators can build a probabilistic evidence graph as defined in Definition 2. However, it is necessary to merge different probabilistic evidence graphs together. For example, the same host that is attacked by using the same vulnerability might be attacked multiple times or involved in

different attack paths in the same network, or different hosts on a distributed network may be involved in the same attack in a network. In the former case, the host results should be assigned a higher probability than the situation where the host is only involved in a single attack. This is intuitive, because a host's being attacked many times implies that it can be attacked easily. More importantly, an integrated evidence graph includes all victim hosts and attack evidence in a network, which provides the information how evidence collected from different sites or nodes fits in recreating attacks in the same network. Therefore, the integrated evidence graph can provide a more precise global attack description on a network.

As mentioned above, for each attack, it is possible to use the method mentioned in [1] to generate an evidence graph and assign probabilities to corresponding host nodes and evidence edges. However, for an integrated probabilistic evidence graph that merges several sub probabilistic evidence graphs together, we need to ensure that the updated probabilities won't violate Definition 2, because an integrated probabilistic evidence graph is still a probabilistic evidence graph. In order to achieve this, we suggest using the following guidelines.

a) If a single host involved in different attacks has different attack status for each attack, we should use different nodes to represent different attacks on the same host. By doing so, the independence of a different evidence probability for a different attack on the same host can be reflected.

b) If the same victim host is exploited by using the same vulnerability, but is involved in different attack paths as a joint node or involved in the same attack at different times, we merge jointed attack paths together by giving the corresponding merged node an increased probability. According to clause 2 in Definition 2, the increased probability of the same node should be $p'(h)=p(h).G_1 + p(h).G_2 - p(h).G_1 \times p(h).G_2$. Here, p'(h) is the new probability of the merged node. $p(h).G_1$ and $p(h).G_2$ are the host probabilities of host h in graph G_1 and G_2 respectively. G_1 and G_2 represent evidence graphs for different attacks or the same attack at different times.

c) Similar hosts that are involved in similar attacks should be grouped together with an increased

probability computed using the equation $p'(h)= p(h).G_1 + p(h).G_2 - p(h).G_1 \times p(h).G_2$. For example, in an operational network, suppose two Windows workstations with IE6 Internet Explorer that has memory corruption vulnerability are taken as step-stones to attack a database server. We can group the two workstations together and assign an increased attack probability to the grouped workstation node.

Under the situation of either (b) or (c) where the host nodes are merged, the corresponding evidence edges should also be merged. Because of evidence's temporal nature and the evidence collection technique's varieties, the evidence collected at different times might not be exactly the same, even though investigators could use them to judge the attacks on the same host are the same or similar. As such, the probabilities for the evidence might not be the same. We use $p(e).G_1$ and $p(e).G_2$ to represent the probabilities of the evidence in evidence graphs G_1 and G_2 respectively, and increase the probability of the merged evidence edge to be $p'(e)=p(e).G_1+p(e).G_2-p(e).G_1 \times p(e).G_2$, where $p'(e)$ is the new merged evidence probability.

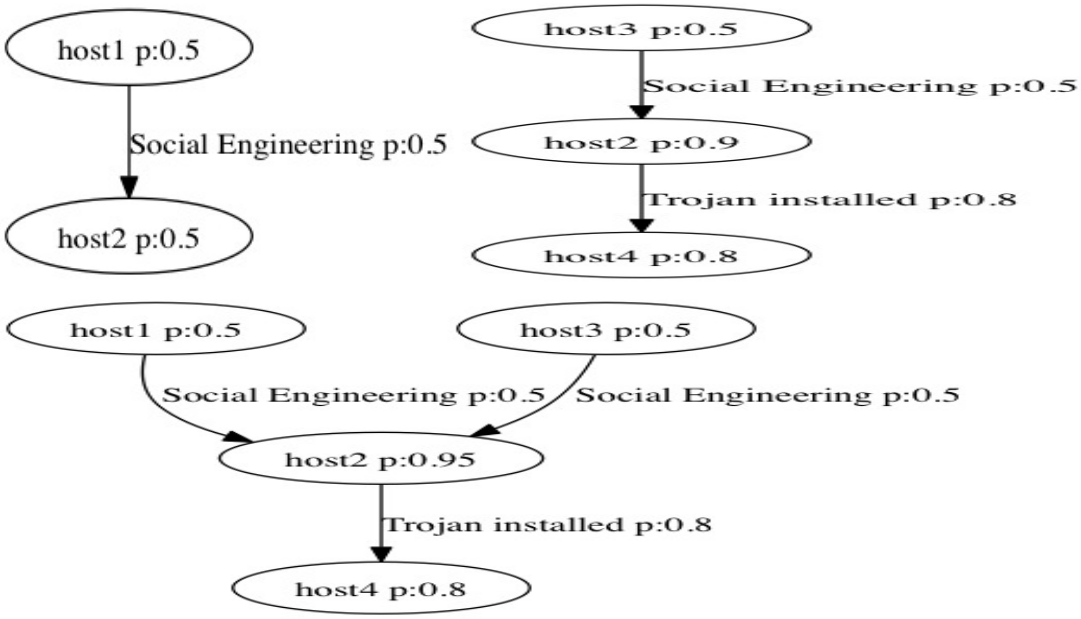

Figure 2: Evidence Graphs Merging

Figure 2 is an example of two separate evidence graphs and their integrated evidence graph. In the two evidence graphs, suppose that host 2 is attacked by exploiting the same vulnerability from host 1 and host 3. We merge two evidence graphs together to an integrated evidence graph (the lower one in Figure 2),

where both host 2 from the two graphs are merged to a single host node (also marked as host 2) with an increased probability p' (h_2) =0.5+0.9−0.5×0.9 =0.95. Here p(h_2) =0.5 is host 2's probability in the left evidence graph, and p(h_2) =0.5+0.8−0.5×0.8=0.9 is host 2' probability in the right evidence graph. All other evidence and host probabilities remain the same.

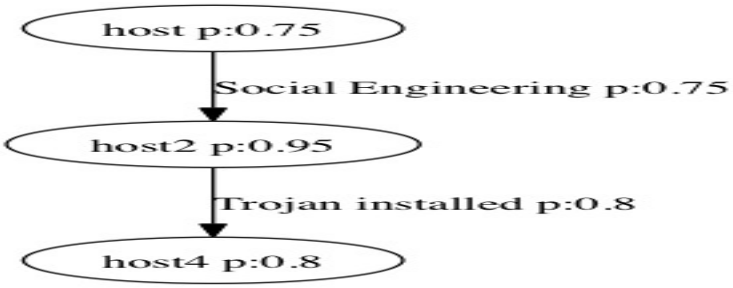

Figure 3: Grouped similar hosts in evidence graphs

In Figure 2, if host 1 ≈ host 3, we can merge them together to create a merged node with an increased probability based on their previous probabilities, which is p' (h_2) =p(h_1)+p(h_3)−p(h_1) ×p(h_3)=2×0.5−0.5×0.5 =0.75. If the evidence of host 1 and host 3 is not the same, we simply move all the evidence to the new merged host, and keep their previous probabilities. If the similar nodes' evidence is the same, we also merge the evidence together and assign it an increased evidence probability p' (e) = p(e_1)+p(e_3)−p(e_1) ×p(e_3)=2× 0.5−0.5×0.5 =0.75 as shown in Figure 3. In the p' (e) equality, e_1 represents the evidence of attacking host 2 from host 1 in the left sub evidence graph, and e_3 means the evidence of attacking host 2 from host 3 in the right sub evidence graph. By merging similar hosts, network administrator can get a more intuitive and succinct view of host configuration vulnerabilities in the network.

Using the principals mentioned above, we propose to directly merge sub evidence graphs into an integrated evidence graph, provided that the sub evidence graphs are well constructed with no tainted or missing evidence. If not, we propose to use the attack graph of the same network as a reference to integrate sub evidence graphs in forming an integrated one. For the latter case, the attack graph should include all vulnerabilities that would cause the attacks in the sub evidence

graphs. By doing so, the attack graph can be used to implement the incomplete evidence graph during the integration process, because the attack graph has vulnerability information and conditions that correspond to missing evidence of the same attack. This happens when the evidence is destroyed or the vulnerabilities used to generate the attack graph are not completely known at the time of evidence collection.

A. Merging Evidence graphs without using an attack graph

Suppose we want to merge two evidence graphs together. For this merging, starting from the victim host node, we use the depth first search algorithm [10] to traverse all host nodes in the first evidence graph or second graph and merge them to a third evidence graph forming an integrated evidence graph. Our algorithm uses a coloring scheme to keep track of nodes in both sub evidence graphs. All nodes are initialized white and colored gray when being considered but with children not yet fully examined. And, a host is colored black after all its children are fully examined. Having all nodes from both evidence graphs colored black, the merging is done and the third evidence graph is the integrated evidence graph. The following is the specific algorithm.

Algorithm 1: Merging probabilistic evidence graphs
Input: Two probabilistic evidence graphs G_i, =(N_i, E_i, N_i-Attr, E_i-Attr, L_i, T_i) with victim nodes v_i for i=1,2
Output: One probabilistic evidence graph G = (N, E, N-Attr, E-Attr, L, T)

Begin:
1. **for** each node n_1 and n_2 in G_1 and G_2
2. **do** color[n_1] ← WHITE, color[n_2] ← WHITE
3. $\pi[n_1]$ ← NIL, $\pi[n_2]$ ← NIL
4. **for** each node h ∈ V[G_1] or h ∈ V[G_2]
5. **do if** color[h] == white
6. **then** DFS-VISIT (h)

End

DFS-VISIT (h)
7. Color[h] ← GRAY
8. MERGING(π[h],h,G)
9. **for** each v ∈ Adj[h]
10. **do if** Color[v] == WHITE
11. **then** π[v] ← h
12. DFS-VISIT(v)
13. Color[h] ← BLACK
14. **return**

MERGING (π[h],h,G)

```
15. for all nodes u in G
16.   if u is identical to h
17.      then p'(u) = p(u)+p(h) –p(u) ×p(h)
18.           p'(u.e) = p(u.e)+p(h.e) –p(u.e) ×p(h.e)
19.   else add h to G  as u '
20.        p(u ') =  p(h)
21.        for all nodes n in G
22.            do if   n is identical to π[h]
23.                 then   π[u '] ← n
24.               else  π[u '] ← NIL
25.               if   π[u '] ≠ NIL
26.                 then   E(π[u '], u ') ← E(π[h], h)
27.                        P(E(π[u '], u ')) ← p(E(π[h], h))
28. return
```

Algorithm 1 integrates two sub evidence graphs G_1 and G_2 to form an integrated evidence graph G. This algorithm checks evidence graphs G_1 and G_2 to see if there is an identical host node in G. We say hosts are "identical", if these hosts, which are even from different attack paths in a network, are exploited by using the same vulnerability--this can be judged using the evidence left behind. If a host node in G is found identical to a node in G_1 or G_2, we keep the one in G and increase its probability as we mentioned before. If there is no such an identical host found in G, the node and its corresponding evidence from G_1 or G_2 should be added to G.

The following is a detailed explanation on our merging algorithm. To begin with, lines 1-3 paint every node in G_1 and G_2 white and set the parent of each node as NIL. Lines 4—6 traverse every host node in graphs G_1 and G_2. During this traversal, if the node color is white (line 5), line 6 calls DFS-VISIT(h) to search for one of its undiscovered child nodes by using depth first search (here DFS stands for depth first search). In Function DFS-VISIT (h), line 7 paints host h Gray, implying that this node is discovered but its children have not been fully examined. Line 8 calls function MERGING[π[h],h,G)] to merge the discovered host node h in either G_1 or G_2 to the integrated evidence graph G. The reason why we need π[h] here is that we need to add the evidence edges between π[h] and h to G later. Line 9 to line 12 iteratively search for an undiscovered child node (v) of host node (h), and assign node v's parent as node h. Line 13

paints the node h black after all its children have been discovered. The algorithm terminates when all nodes from evidence graph G_1 and G_2 are painted black.

In MERGING function, lines 15-18 decide if h (the node passed from G_1 or G_2) is identical to any node in G. If one node u is found to be identical to node h, the host u's probability is increased to $p'(u) = p(u)+p(h)–p(u)\times p(h)$, and the corresponding evidence probability is increased to $p'(h.e) = p(u.e)+p(h.e) – p(u.e)\times p(h.e)$ (lines 17-18), because the identical node h and its corresponding evidence are merged into node u in G. Here, p'(u) is the merged probability of u in G. p(u) and p(h) are the probabilities of u and h in evidence graph G and G_1/G_2 respectively before the merging. Correspondingly, p'(u.e) is the probability of merged evidence whose source or destination host is u. p(u.e) and p(h.e) are the probabilities of evidence whose source computer or destination computer is u or h in G and G_1/G_2 respectively before merging. If there is no such a node in G that is identical h, the host node h and its attack probability from evidence graph G_1 or G_2 are added to evidence graph G with a new name u' (lines 19-20). Lines 20-23 find the new added node u''s parent node π[u '] in G, which should be identical to π[h] in G_1/G_2 (This is possible since π[u '] has been found or added before the new node u' is added). Otherwise, π[u '] is equal to NIL. Once the parent node of u ' has been decided, lines 24-25 copy the evidence edges between π[h] and h in G_1 or G_2 to π[u '] and u ', which then are integrated into G. Also, the evidence edges' probabilities are copied to G in line 27.

B. Mapping sub evidence graphs to form the integrated evidence graph by referring to attack graphs

Under some circumstances, for example, when anti-forensics technologies are used, some evidence could be missing. Therefore, the constructed evidence graph might have un-connected nodes because of missing evidence edges. In this case, merging evidence graphs becomes problematic. As a solution, we use an attack graph of the whole enterprise to fill in the missing evidence by mapping sub evidence graphs to the attack graph to find the corresponding attack path. As a byproduct, we enhance the attack graph using the mapping process.

a)Processing a large-scaled attack graph

Runtime overhead has been a problem to a large scale attack graph. It is impractical to map evidence graphs to a large-scaled attack graph, because such a mapping algorithm [4] basically uses depth-first search algorithm, which has a running time O($V + E$)[10], where V is the number of nodes, and E is the number of edges in the attack graph. Because an evidence graph is usually much smaller than an attack graph, which only has a polynomial size, the running time of the algorithm is mainly determined by the attack graph's size that can be exponential. In order to solve this huge complexity problem of the mapping algorithm, we use the following methods to reduce the attack graph complexity.

(1). Use the method suggested in [6] to group hosts with similar vulnerabilities together to reduce the complexity of the attack graph. For example, in an enterprise network, there could be many computers connected to a database server. It is very possible that those computers have similar vulnerabilities. Therefore we can group similar machine nodes together and assign the grouped machine node a higher probability using the equation $p(h)= p(h_1 \cup h_2)=p(h_1)+p(h_2)-p(h_1)\times p(h_2)$, where h is the grouped machine node, h_1 and h_2 represent host 1 and host 2 that have similar vulnerabilities. This would reduce the size of the attack graph.

(2). The probability calculated by Definition 4 represents the attack success probability and reachability. If we take it as a rank to drop those hosts that have lower attack success probabilities [8], the size of the attack graph would be reduced. We use a value 0.02 as the minimum reachability in this paper.

(3). Drop off all paths where the destination computer won't expand further to the desired victim hosts and all hosts that are not involved in the desired attack paths. In addition, if there are multiple exploits at the same host, neglect those exploits for which Common Vulnerability Scoring System (CVSS) is smaller than a preselected threshold value. By using the above strategies, we can generate a compact sub attack graph of a complete attack graph with much smaller complexity.

c. Using the Attack Graph to merge sub evidence graphs

This section describes Algorithm 2, which is designed to use an attack graph to merge evidence graphs when they miss some evidence. In this algorithm, we still use the coloring mechanism to mark those nodes that have been discovered. In sub evidence graphs, gray colored nodes mean they have been visited, but not all children have been visited. A node is marked black when all its children nodes have been visited.

Algorithm 2: Integrating evidence graphs by referring to attack graph

Input: Two probabilistic evidence graphs G_i, =(N_i, E_i, N_i-Attr, E_i-Attr, L_i, T_i) with victim nodes v_i for i=1,2,

A probabilistic attack graph: A=(N_r, N_p, N_d, E, L, G, P). N_p has vulnerability information. N_r is an exploit, and N_d is the host state after exploiting the vulnerability.

Output: A merged evidence graph G = (N, E, N-Attr, E-Attr, L, T)

```
Begin:
1.  for each node n1 and n2 in G1 and G2
2.        do color[n1] ← WHITE, color[n2] ← WHITE
3.           π[n1] ← NIL, π[n2] ← NIL
4.  for each node h ∈ V[G1] or h ∈ V[G2]
5.        do if color[h] == white
6.              then DFS-VISIT (h)
End

DFS-VISIT (h)
7.    color[h] ← GRAY
8.    MERGING(π[h],h,A)
9.    for each v ∈ Adj[h]
10.         do if color[v] == WHITE
11.              then π[v] ← h
12.                   DFS-VISIT(v)
13.   color[u] ← BLACK
14.   return

MERGING (π[h],h,G,A)
15.  for all nodes u in G
16.       do if u is identical to h
17.            then p'(u) = p(u)+p(h) –p(u) ×p(h)
18.                 p'(u.e) = p(u.e)+p(h.e) –p(u.e) ×p(h.e)
19.          else add h to G as u '
20.               p(u ') = p(h)
21.               for all nodes n in G
22.                    do if n is identical to π[h]
23.                         then π[u '] ← n
24.                       else π[u '] ← NIL
25.                       if π[u '] ≠ NIL and E(π[h], h) ≠ NIL
26.                          then E(π[u '], u ') ← E(π[h], h)
27.                               P(E(π[u '], u ')) ← p(E(π[h], h))
28.                       else
29.                          for each derived node m in A
```

30. **do if** m is identical to π[u '] or u '
31. **then** color[π[u ']] ← BLACK, color[u '] ← BLACK
32. E(π[u '], u ') ← Path between two marked nodes in A
33. **return**

Algorithm 2 enhances Algorithm 1 by looking up the missing evidence in the attack graph A. Line 1 to Line 27 are exactly the same as algorithm 1. Line 28 to Line 32 aim to find the missing evidence between a new added node u' and its parent node π[u '] in the attack graph A (if the evidence does not exit in evidence graphs G_1 and G_2 , we search for it in A). To be specific, line 22 to line 24 search for the evidence edges between π[h] and h in G_1 or G_2, which correspond to evidence edges between π[u '] and u ' in G. Once they are found, line 25 to line 27 add them as the evidence edges between π[u '] and u'. If such evidence edges do not exist, the evidence is missing in both the sub evidence graphs G_1 and G_2. If so, then, lines 29 to 31 traverse all derived nodes in attack graph A to find the corresponding nodes for π[u '] and u' from G, and mark them Black. Line 32 adds the found attack paths between the two marked nodes in the attack graph A to the integrated attack graph G as the evidence edges between π[u '] and u '.

To sum up, if two evidence graphs are so well constructed that there are no missing evidence edges or nodes, algorithm 1 should be used. Otherwise, algorithm 2 is used.

D. Complexity Analysis

In both algorithms, all nodes and edges in evidence graph 1 and 2 will be traversed. We use “n” and “e” to represent all nodes and edges in both evidence graphs.

In algorithm 1, lines 1 to 3 have an O(n) run time, since the three lines only go through every single node once. Lines 4 to 6 traverse every node to do a depth first search (“DFS-VISIT”), which has an O(n) run time. In this “DFS-VISIT” function, line 8 calls the “MERGING” function, where we can see there are two “for” loops, which take a $O(n^2)$ run time. Lines 9 to 13 run depth first search to traverse every single edge and node in both evidence graphs to find the current node’s next un-marked child node, which takes O(n+e) time as proved in [10]. If we put all these run times together, we can see algorithm 1 has a

complexity $O(n+n\times(n\times e + n\times n^{2)}) = O(n^2\times e + n^4)$. For most evidence graphs, e is polynomial to n since there are usually at most three edges between two nodes, so the run time is polynomial.

Algorithm 2 is similar to algorithm 1, except that algorithm 2 has to traverse all derived nodes in the corresponding attack graph so that the missing evidence in the evidence graphs can be found. This is done by a "for" loop between lines 29 to 31. Suppose we use "m" to represent node number in the attack graph, the complexity of algorithm 2 is $O(n + n\times(n\times e + n\times n^2\times m)) = O(n^2\times e + n^4 \times m)$, which is determined by m, since e is only polynomial to n.

Complexity has been a hindrance of using attack graph. In particular, for a state-based attack graph, the complexity can be $O(2^n)$ ("n" means node number) , because every node is a collection of Boolean variables encoding the entire network state at an attack stage and attacker's actions are modeled as state-transition[13]. A logical attack graph has dramatically reduced the complexity down to a polynomial time. [3] has proved that a logical attack graph for a network with N machines has a size at most $O(N^2)$. However, in a large-scaled network with a big N, this complexity can still be a problem. By using the methods mentioned in section B a)-- grouping, removing unrelated attack paths, dropping machines with small reachability, we could greatly shrink the number of N. Besides, because, in reality, an evidence graph is much smaller than an attack graph, if we take hosts involved in evidence graphs as references to generate an attack graph, the size of the attack graph can be even smaller. Therefore, the complexity of algorithm 2 is polynomial.

IV. AN EXAMPLE OF AN ATTACK SCENARIO

In this section, we describe an attack scenario by using a small example network shown in Figure 4. In Figure 4, the external firewall controls network access from the Internet to the enterprise network, where the Apache Tomcat webserver that uses Firefox 3.6.17 hosts a webpage allowing Internet users' visit via port 8080. The internal firewall controls the access to MySQL database server and fileserver. The webserver and workstation can access the database server via the default port 3306, and the webserver is

allowed to access fileserver through the NFS protocol. Suppose there are many workstations with IE6 installed in this network, which can get access to internal database server.

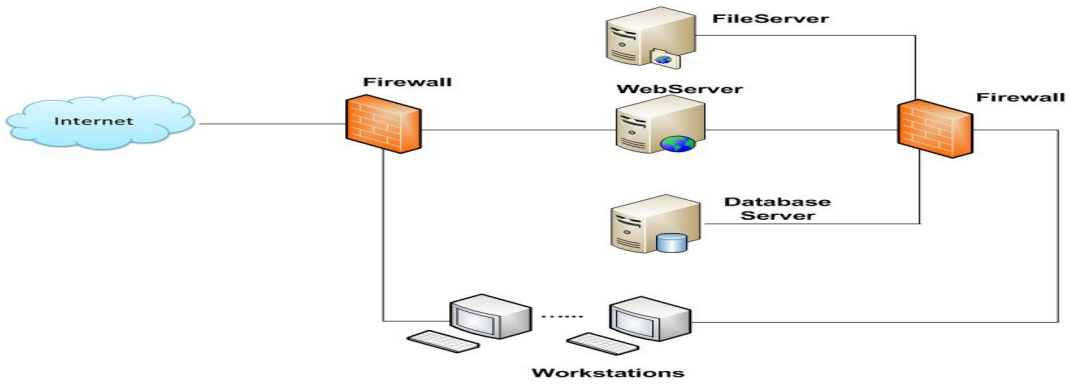

Figure 4: An Example Network

We assume that the objective of the attacker is to gain access to database tables in the database server and to comprise the fileserver. For the attack plan to the database server, we assume a SQL injection attack that exploits a java servlet code on the webserver, which did not sanitize input values: *theStatement.executeQuery("select * from profiles where name='Alice' AND password='"+passWord+"'");*. This exploit corresponds to CWE89 in NVD [12]. For the attack plan on the fileserver, we assume vulnerabilities in the NFS service daemons (*CVE-2003-0252*) to compromise the file server, which can be done by controlling the webserver using its vulnerability on Firefox 3.6.17 (*CVE-2011-2365*). We assume that workstation ran Windows XP SP3 operating system with IE6 had the vulnerability (*CVE-2009-1918* [9]) that enabled executing any code on this machine. We assume that an attacker can use social engineering to trick the workstation user to click on a malicious web link, which can enable the attacker's machine to control the workstation that could directly access the database.

A. Merging Evidence Graphs without Referring to Attack graph

For this example scenario, the attack roles and vulnerabilities on all computers are stated in Table 1.

a) An un-sanitized string "anything'or'1'='1" into the password field of the Webpage will cause a SQL injection attack to a database table "profiles". This corresponds to vulnerability "CWE-89".

b) Vulnerability “CVE-2009-1918” can be exploited by tricking a workstation user to click on a link sent from the attacker’s machine. By using this vulnerability, Internet Explorer "Aurora" Memory Corruption attack can be launched from the attacker’s machine to the Windows workstation that used the IE6 browser.

c) A “CVE-2011-2365” vulnerability on Firefox 3.6.17 can be used to hijack the Web server. This memory corruption will allow the attacker to execute arbitrary code on Web Server.

d) After having hijacked the workstation and web server, the attacker can remotely control workstation to access database server or attack fileserver by using vulnerabilities in the file server.

f) A “CVE-2003-0252” vulnerability can be exploited to comprise the file server, which will allow the attacker to execute arbitrary code via certain RPC requests to mountd.

TABLE 1:ROLE CONFIGURATION IN THE ATTACKS

Attacker	129.174.128.148	
Stepping Stone	Workstation	*CVE-2009-1918*
Stepping Stone/Affiliated	Web server	CWE89, *CVE-2011-2365*
Victim	Database server, File server	CVE-2003-0252

We can model evidence as a vector (id, source, destination, content, time stamp) and use Graphviz [11] to generate two evidence graphs (Figure 5 and 6) by using time dependencies and network configurations. Respectively, they are attacks on database server and file server. In both graphs, solid edges represent primary evidence and second evidence that have coefficient 1 and 0.8 respectively. Dotted edges represent expert knowledge, which has a coefficient 0.5. The probabilities for the evidence edges and hosts are calculated by using Definition 2.

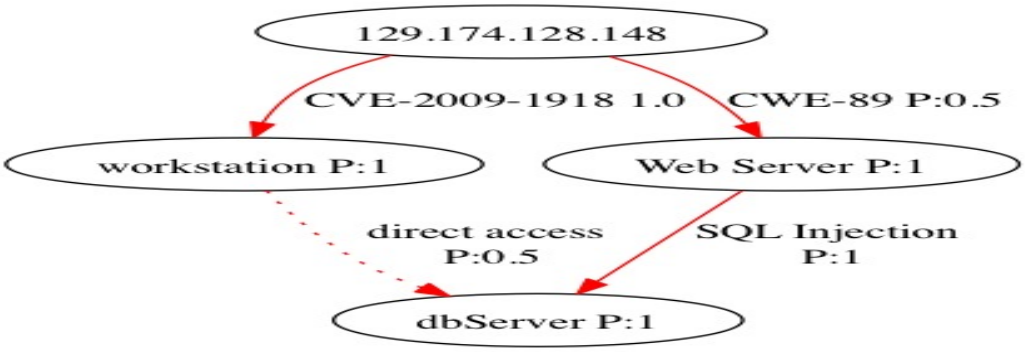

Figure 5: Evidence graph for attack based on dbServer

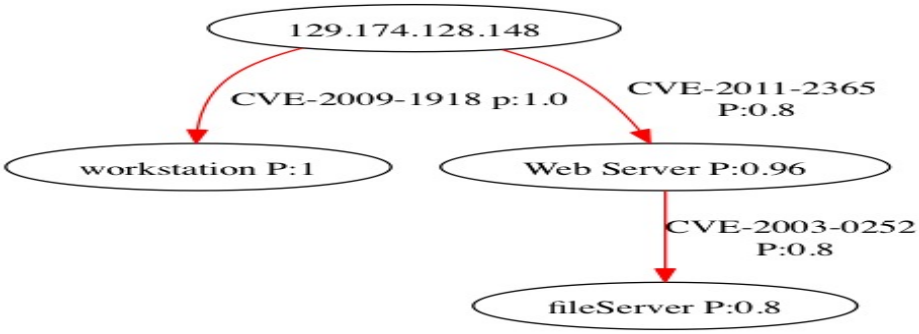

Figure 6: Evidence graph for attack based on fileserver

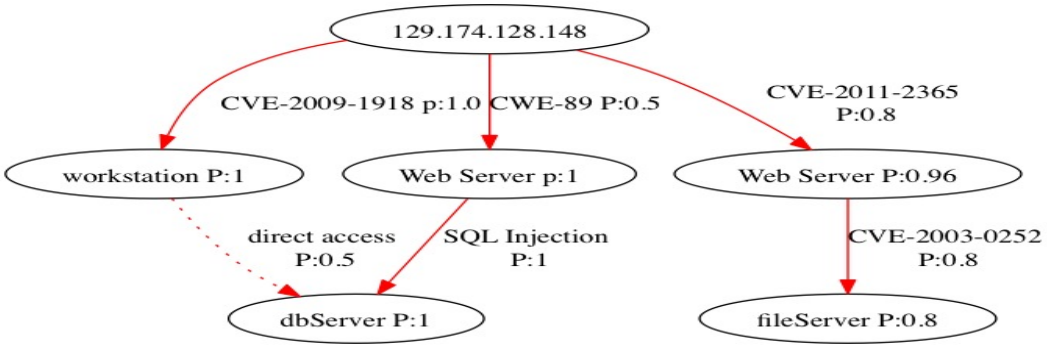

Figure 7: Combined Evidence Graph from Figure 5 and Figure 6

We applied algorithm 1 to both evidence graphs and got the integrated evidence graph as Figure 7, where the probabilities on the database sever and fileserver are not affected since the attacks on them are independent in the two evidence graphs. Because the workstation is involved in both attacks where the attacks to it are the same, the workstation in the two evidence graphs in Figure 5 and 6 are merged into a single node in the integrated evidence graph in Figure 7. And the attack probability on workstation is increased to $p'(h)=p(h.G_1)+p(h.G_2)-p(h.G_1)\times p(h.G_2) = 1+1-1\times1=1$. Here, $p(h.G_1)$ and $p(h.G_2)$ represent the probabilities of p(h) in evidence graph G_1 and G_2 respectively.

B. Integrate evidence graphs by referring to the attack graph

a) Reducing Attack Graph Complexity

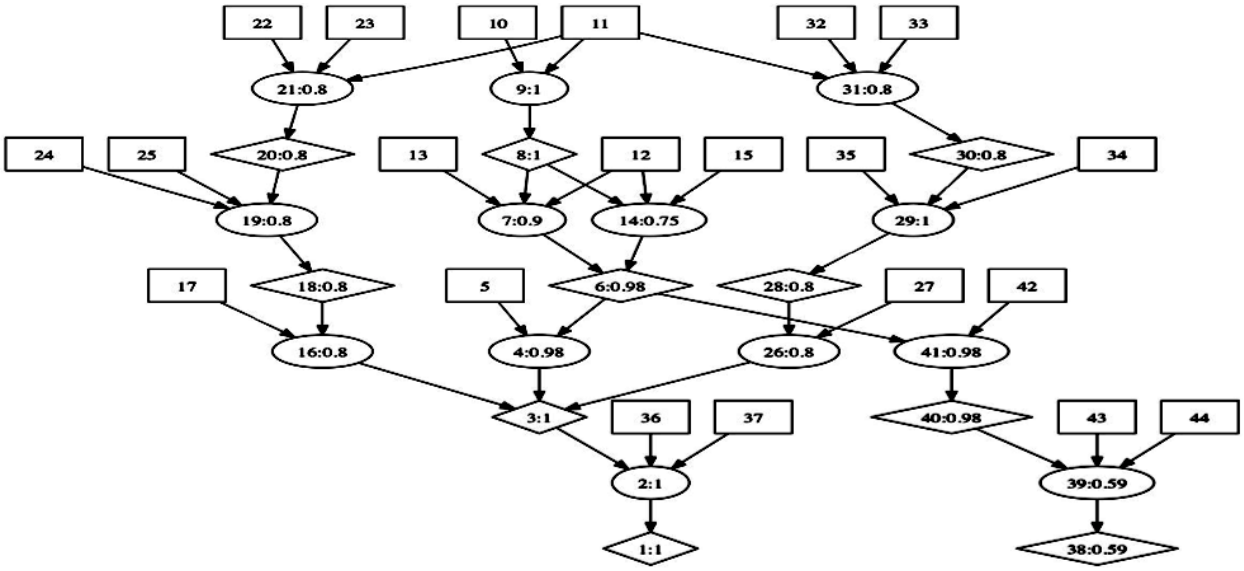

Figure 8: Attack graph with two workstations

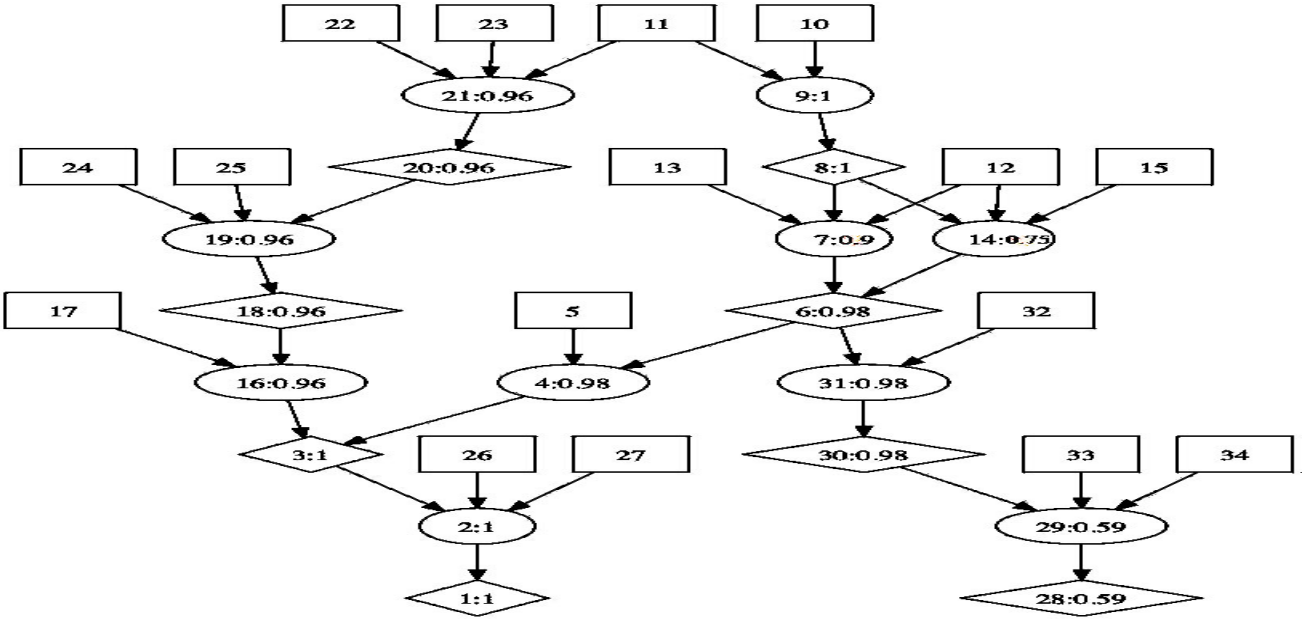

Figure 9: Attack graph with two workstations

In order to have a succinct visual effect, we only took two workstations out of many in Figure 3 for graph generation. With all vulnerabilities information on webserver, database server, fileserver and the two workstations, we generated an attack graph using MulVAL[5], the tool used for logical attack graph generation. This attack graph is in Figure 8 where we can see there are two exactly same attack paths on the left side and right side. If we merge two workstations to one, we can get figure 9 that obviously has a

smaller complexity than figure 8, since there are less host nodes in Figure 9. We can see that all other nodes keep their previous attack success probabilities, except the following grouped ones:

1) p'(21) = p(21)+p(31)-p(21) ×p(31) = 0.8+0.8-0.8×0.8=0.96
2) p'(20) = p(20)+p(30)-p(20) ×p(30) = 0.8+0.8-0.8×0.8=0.96
3) p'(19) = p(19)+p(29)-p(19) ×p(29) = 0.8+0.8-0.8×0.8=0.96
4) p'(18) = p(18)+p(28)-p(18) ×p(28) = 0.8+0.8-0.8×0.8=0.96.

Our study is based on a lab environment, which is small. In an operational network, the attack graph could be large if every single host has a separate attack path. By grouping the host computers that have same configuration and vulnerability information together, we can have a much smaller attack graph.

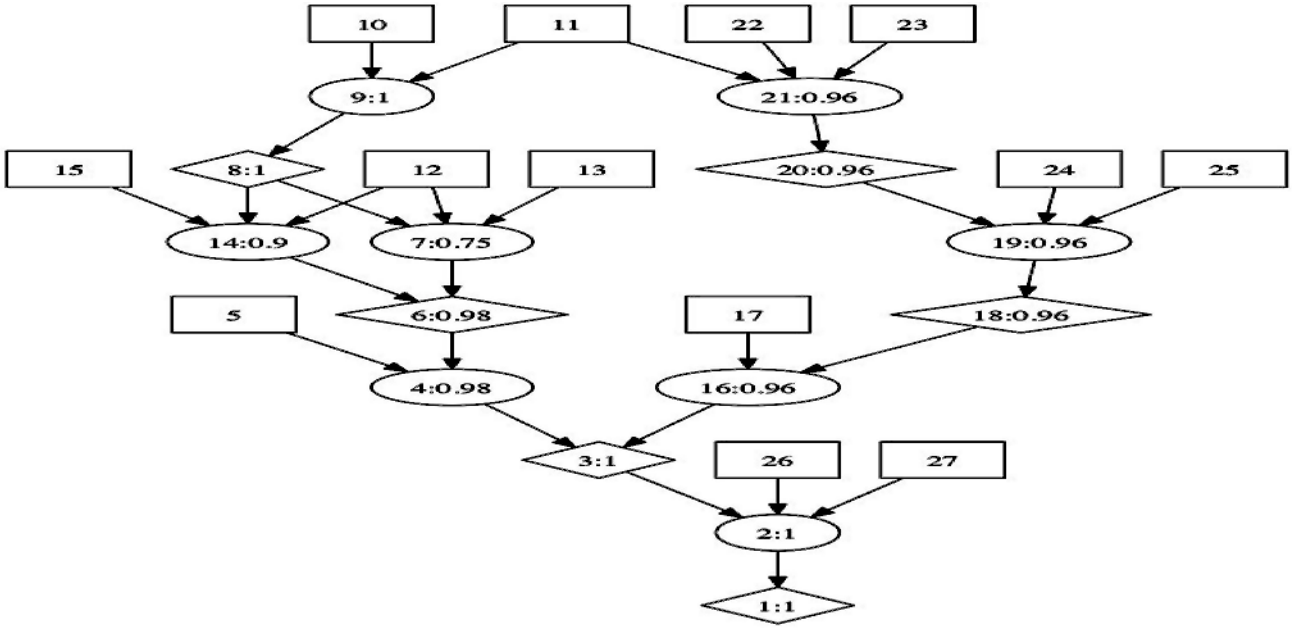

Figure 10: A sub attack graph of Figure 9

We also generated a sub attack graph of a complete attack graph. Suppose that the fileserver has not been attacked in our experimental network. Consequently, the evidence graph won't have a corresponding victim host--fileserver. Having removed the attack goal toward fileserver in the input file to MulVAL, we obtained the Figure 9's sub attack graph (Figure 10), which obviously has a smaller size than Figure 9, the complete attack graph. Our experiment did not have a path with a final host attack success probability less than 0.02, but we modified the MulVAL program to remove all nodes falling under, in order to get a smaller attack graph.

b) Integrating evidence graphs

Having generated a compact attack graph shown in Figure 9, which includes victim hosts in the evidence graphs, we used Algorithm 2 to test integrating evidence graphs by referring to the attack graph. In order to have missing evidence in evidence graphs, we assume the evidence from web server to fileserver in Figure 6 has not been found as shown in Figure 11.

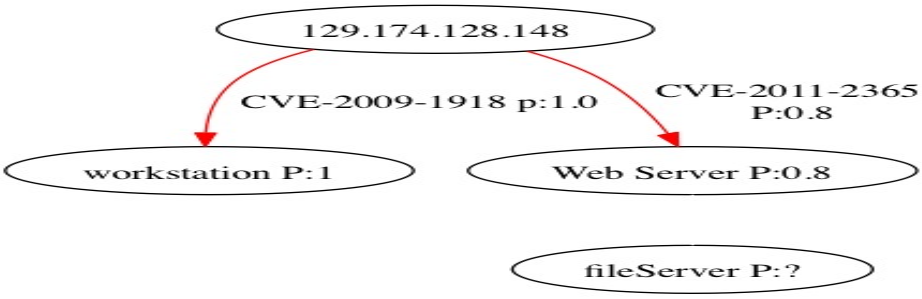

Figure11：Evidence graph 6 with missing evidence

We Applied algorithm 2 to two evidence graphs (Figure 5 and Figure 11) and the corresponding attack graph (Figure 9), and got our result as Figure 12. The upper graph is the integrated evidence graph where all nodes and edges colored in red are integrated from both evidence graphs in Figure 5 and Figure 11. The black attack path between “Web Server” and “fileserver” is added by copying the corresponding attack path (colored in red) from the attack graph, the lower graph in Figure 12 where node 6 and 28 correspond to “Web Server” and “fileserver” respectively. This new added attack path between the “Web Server” and the “fileserver” means the webserver can be used as a step stone to comprise file server. This is done by using the vulnerability of RPC requests to mountd that does not contain newlines, which is named “CVE-2003-0252” in NVD[9].

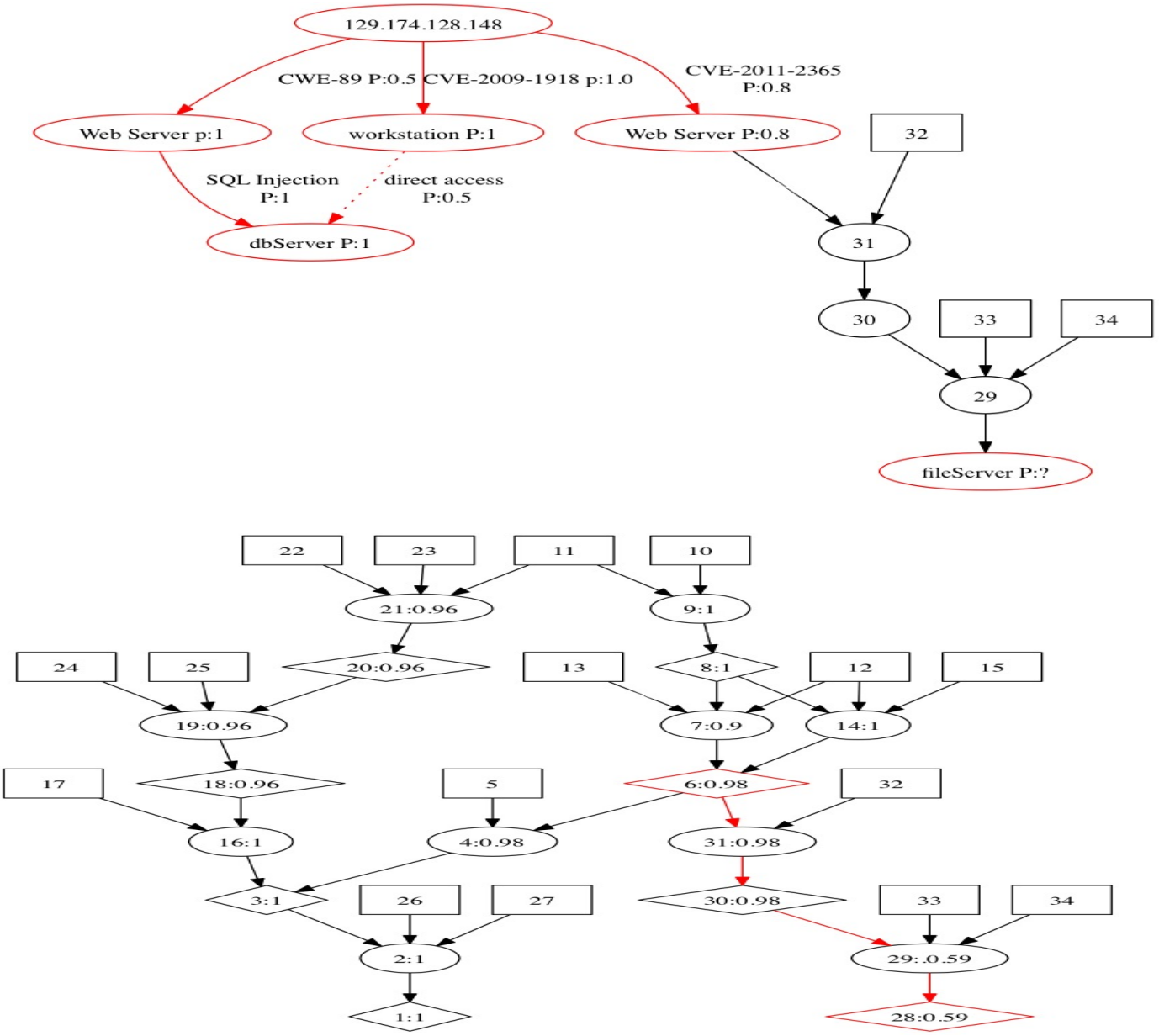

Figure 12: Mapping evidence graphs to attack graph

VII. Conclusion

In this paper, we have developed two algorithms to integrate multiple evidence graphs to a single evidence graph. Our first algorithm merges probabilistic evidence graphs directly without paying any attention to infrastructural vulnerabilities. Our second algorithm merges multiple evidence graphs in the presence of an attack graph. This algorithm is used under the circumstance where the evidence is tampered or missing. Because our second algorithm could result in a long runtime if the attack graph has a big complexity, we showed some approximations that would shrink the attack graph in order to reduce the

runtime. We used an example attack scenario to show that both algorithms work well to get an integrated evidence graph.

DISCLAIMER

This paper is not subject to copyright in the United States. Commercial products are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

REFERENCES

1. W. Wang, T. E. Daniels, A graph based approach toward network forensics analysis, ACM Transactions on Information and Systems Security 12 (1) (2008).

2. A. Singhal,X. Ou,"Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs", NIST InterAgency Report 7788, September 2011.

3. 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).

4. C. Liu, A. Singhal, D. Wijesekera, Mapping Evidence Graphs to Attack Graphs, IEEE International Workshop on Information Forensics and Security, December, 2012

5. MulVAL V1.1, Jan 30, 2012, http://people.cis.ksu.edu/~xou/mulval/.

6. J. Homer, A. Varikuti, X. Ou, M. A. McQueen, Improving attack graph visualization through data reduction and attack grouping. 5th International Workshop on Visualization for Cyber Security (VizSEC 2008), Cambridge, MA, U.S.A., September 2008.

7. TightVNC Software,

http://www.tightvnc.com/.

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

9. National Vulnerability Database, http://nvd.nist.gov/.

10. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, MIT University Press, Cambridge, 2001.

11. http://graphviz.org/

12. S. Jha, O. Sheyner, and J.M. Wing. Two formal analysis of attack graph. In Proceedings of the 15th Computer Security Foundation Workshop (CSFW'02), 2002.

13. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M.Wing. Automated generation and analysis of attack graphs. In Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02), pages 273–284, 2002.

14. 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), pages 217–224, 2002.

15. Kyle Ingols ,Richard Lippmann and Keith Piwowarski. "Practical Attack Graph Generation for Network Defense", Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual, pages 121-130.

16. L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08), 2008.