Chapter 1

MERGING SUB EVIDENCE GRAPHS TO AN INTEGRATED EVIDENCE GRAPH FOR NETWORK FORENSICS ANALYSIS

Changwei Liu #[1] , Anoop Singhal *[2], Duminda Wijesekera #[3]

#*Department of Computer Science, George Mason University, Fairfax VA 22030 USA* [1]cliu6@gmu.edu, [3]dwijesek@gmu.edu

**National Institute of Standards and Technology, Gaithersburg MD 20899 USA* [2]anoop.singhal@nist.gov

Abstract Evidence graphs model network intrusion evidence and their dependencies to help with network forensics analysis. With quantitative metrics, 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 available evidence forms a single evidence graph. We show how to merge different evidence graphs with or without the help of a corresponding attack graph. We show this by providing algorithms and a possible attack scenario towards a file server 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 and network administrators than multiple evidence graphs that use *local* reasoning.

Keywords: Probabilistic evidence/attack graphs, Evidence and host probabilities, Integrated probabilistic evidence graphs, Forensics

1. Introduction

Evidence graphs and attack graphs have been used in security analysis. Because evidence graphs are built from attacks occured in a specific network environment,they are generally used to help with network forensics analysis 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].

Attack graphs are used to analyze security vulnerabilities and their dependencies in an enterprise network. Combining quantitative metrics, probabilistic attack graphs can estimate probabilities of attack success in networks [2]. While a good probabilistic attack graph provides potential attack success probabilities to help with 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, [4] proposed to adjust an inaccurate probabilistic attack graph by mapping a corresponding probabilistic evidence graph to it. The mapping algorithm works well in mapping an evidence graph to a small-scaled attack graph, but it does not provide a solution to mapping multiple evidence graphs to one large-scaled attack graph. The reason for having multiple attack graphs is clear–namely evidence is collected from multiple systems, and investigators have to join them together to get a complete view of the evidence trail left over by a distributed attack. To the best of our knowledge, the issue of integrating evidence graphs has not been discussed in published literature, 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 attack scenario to show how to integrate evidence graphs, and finally section V has our conclusions.

2. 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. 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 [3,7]. 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, logical attack graphs [1,3] and both graphs associated with probabilities.

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 $L : N \rightarrow 2^{N-Attr}$ and $T : E \rightarrow 2^{E-Attr}$ assign attribute-value pairs to a node and an edge respectively.

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

Definition 2 (Probabilistic Evidence Graph) [4]: In a probabilistic evidence graph G=(N,E,N-Attr,E-Attr,L,T,p), the probability assignment functions p[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 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. 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, 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] is represented by 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. In some circumstances, the evidence does not exit or is destroyed. A subject matter expert may insert an edge as the expert opinion.

***D*efinition 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 named 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].

Figure 1 is a logical attack graph. A primitive fact node is shown as a box, which represents a specific network configuration or vulnerability information of a host. A derivation node is shaped as an ellipse, representing a successful application of an interaction rule on input facts including primitive facts and prior derived facts. The successful interaction results in a derived fact node represented by a diamond, which is satisfied by these input facts.

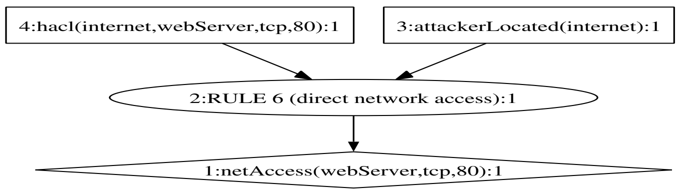

4:hacl(internet,webServer,tcp,80):1
3:attackerLocated(internet):1
2:RULE 6 (direct network access):1
1:netAccess(webServer,tcp,80):1

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 $p : N_p \cup N_r \cup N_d \to$ [0,1] assigns probabilities to nodes. If we use exploits(e) to represent derivation nodes N_r and conditions(c) to represent primitive facts N_p or derived facts N_d, the cumulative probability function P for "e"and "c"of a probabilistic attack graph $P : N_p \cup N_r \cup N_d \to [0, 1]$ is computed from $p : N_p \cup N_r \cup N_d \to [0, 1]$ as follows.

1. P(c)= p(c) =1, if the condition is a primitive fact that is always satisfied.

2. P(c)= $p(c) \times \bigoplus P(e)$, c is the condition of the derived fact node that is derived from derivation nodes(e). Probability law applies here.

3. P(e)=$p(e) \times \prod P(c)$, where $e \in N_r$, and c are conditions that include primitive facts and derived facts from prior step. p(e) can be obtained from NVD from NIST [9].

We define sub attack graphs and similar nodes that are used in this paper as follows.

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) $E' \subseteq E \cap ((N_1, N_2) \in E' \to N_1, N_2 \in N'_r \cup N'_p \cup N'_d$, and

3) $G' \subseteq G$, and $L' \subseteq L$

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-Attr,E-Attr,L,T) ,

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

2) If same kinds of hosts N_1 and $N_2 \in G$, the attack status of N_1 is equal to the attack status of N_2, and $E(N_1) \equiv E(N_2)$, where $E(N_1)$ and $E(N_2)$ are the evidence edges 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 $N_1 \approx N_2$.

[1] suggests normalizing all evidence to five components (1) id, (2) source, (3) destination, (4) content and (5) time stamp, and use them as edges to connect hosts in a time order forming an evidence graph. MulVAL[5] is used to generate a logical attack graph in this paper.

3. Merging Sub Evidence Graphs

It is necessary to merge different probabilistic evidence graphs together, because an integrated evidence graph, which includes all victim hosts and attack evidence in the entire network, can provide a global attack description of the network. In order to have a valid integrated evidence graph, we use the following guidelines to ensure that the probabilities for merged host nodes or evidence edges won't violate Definition 2.

a) If a single host involved in different attacks has different attack status for each attack, we use different nodes to represent each attack on the same host.

b) If the same victim host exploited by using the same vulnerability is involved in different attacks, we merge the node in different attacks into one and give it an increased probability $p'(h) = p(h)_{G1} + p(h)_{G2} - p(h)_{G1} \times p(h)_{G2}$. Here, $p'(h)$ is the increased probability of the merged node. $p(h)_{G1}$ and $p(h)_{G2}$ are the host's probabilities in different evidence graph G_1 and G_2 respectively.

c) Similar hosts involved in similar attacks should be grouped together with an increased probability $p'(h) = p(h)_{G1} + p(h)_{G2} - p(h)_{G1} \times p(h)_{G2}$.

In either (b) or (c) where the host nodes are merged, the corresponding edges representing the same evidence should also be merged. If we use $p(e)_{G1}$ and $p(e)_{G2}$ to represent the evidence probabilities in evidence graphs G_1 and G_2 respectively, the merged evidence probability is increased to $p'(e) = p(e)_{G1} + p(e)_{G2} - p(e)_{G1} \times p(e)_{G2}$, where $p'(e)$ is the new merged evidence edge probability.

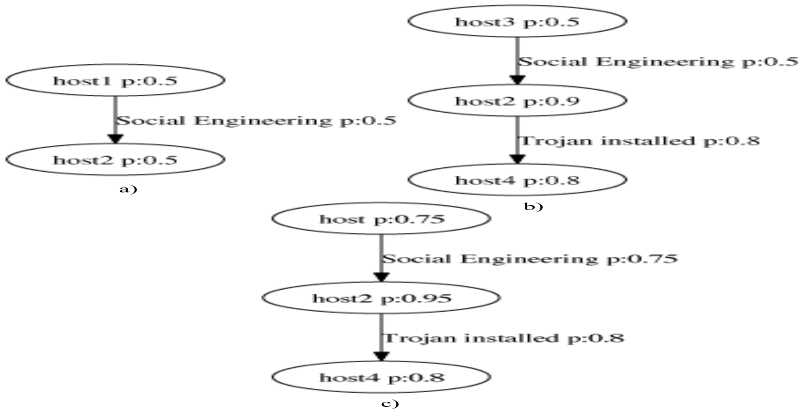


Figure 2. Evidence Graphs Merging

Figure 2 is an example of two separate evidence graphs("a)"and "b)") and their integrated evidence graph("c)"). In the two evidence graphs, suppose that "host2"is attacked by using the same vulnerability from "host1"and "host3", and host1 $\approx$ host3. Both "host2"from graphs "a)"and "b)"can be merged to a single"host2"node with an increased probability $p'(h2) = 0.5 + 0.9 - 0.5 \times 0.9 = 0.95$, and both "host1"and "host3"from two upper graphs can be merged to "host"with an increased probability $p'(h) = p(h1)+p(h3)-p(h1)\times p(h3) = 0.5+0.5-0.5\times 0.5 = 0.75$. Correspondingly, the merged evidence should be increased to $p'(e) = p(e1) + p(e3) - p(e1) \times p(e3) = 2 \times 0.5 - 0.5 \times 0.5 = 0.75$.

If sub evidence graphs are well constructed with no tainted or missing evidence, we propose to directly merge sub evidence graphs into an integrated evidence graph. If not, we suggest to use the attack graph generated from the same network as a reference to integrate sub evidence graphs in forming an integrated one, because the attack graph can help implement the incomplete evidence graph.

3.1 Merging Evidence Graphs Without Using An Attack Graph

Suppose we merge two evidence graphs together. Starting from both victim host nodes, 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 one. 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. 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 one. The details are in Algorithm 1.

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($\pi$[h],h,G)
9.  for each v ∈ Adj[h]
10.       do if Color[v] == WHITE
11.          then  $\pi$[v] ← h
12.                DFS-VISIT(v)
13.   Color[h] ← BLACK
14.  return

MERGING ($\pi$[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 $\pi$[h]
23.                  then   $\pi$[u '] ← n
24.                else  $\pi$[u '] ← NIL
25.                if   $\pi$[u '] ≠ NIL
26.                  then   E($\pi$[u '], u ') ← E($\pi$[h], h)
27.                         P(E($\pi$[u '], u ')) ← p(E($\pi$[h], h))
28.   return
```

We use G_1,G_2 and G to represent the two sub evidence graphs and the integrated evidence graph. Algorithm 1 checks evidence graphs G_1 and G_2 to see if there is an identical host node in G. "identical host"means the same machine that shows in different evidence graphs, but is exploited by using the same vulnerability. 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. If there is no such an identical host in G, the node and its corresponding evidence from G_1 or G_2 should be added to G.

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. If the node color is white (line 5), DFS-VISIT(h) is called(line 6) to search for one of its undiscovered child nodes by using depth first search (DFS). 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($\pi[h], h, G$) 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 $\pi[h]$ here is that we need to add the evidence edges between $\pi[h]$ 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 h as the parent of node v. Line 13 paints 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 such a node u is found to be identical to h, node 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 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 before merging. If there is no such a node in G that is identical to 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 parent node $\pi[u']$ of the new added node u' in G, which should be identical to $\pi[h]$ in G_1/G_2. Otherwise, $\pi[u\prime]$ is equal to NIL(line 24). Once the parent of u' has been decided, lines 25-27 merge the evidence edges and their corresponding probabilities between $\pi[h]$ and h in G_1 or G_2 to $\pi[u']$ and u' in G.

3.2 Merging Evidence Graphs By Referring To An Attack Graph

Under some circumstances, the constructed evidence graph has unconnected 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 network 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.

3.2.1 Processing A Large-scaled Attack Graph. Run-time overhead has been a problem to a large scaled attack graph. It is impractical to map evidence graphs that usually have a polynomial complexity to a large-scaled attack graph that has an exponential number of nodes(V) and edges(E), because a mapping algorithm [4] using depth-first search algorithm has a run time O(V + E)[10]. In order to solve this complexity problem, we use the following methods to reduce the attack graph complexity.

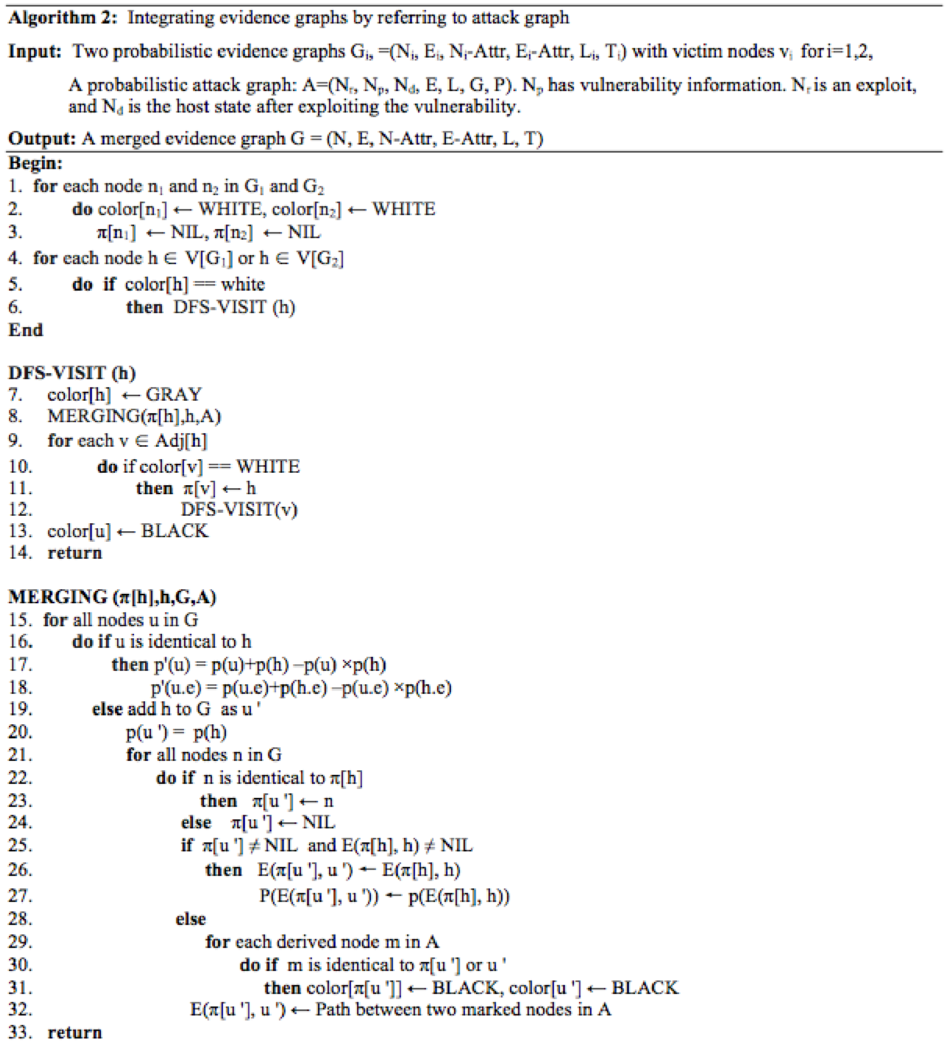
1. Use the method suggested in [6] to group hosts that have similar vulnerabilities together and assign the grouped node a higher probability p(h)= $p(h_1 \cup h_2) = p(h_1) + p(h_2) - p(h_1) \times p(h_2)$, where h is the grouped machine node, h_1 and h_2 represent host 1 and host 2 that have similar vulnerabilities.

2. Attack success probility also represents the attack reachability. Drop those hosts that have low reachability[8]. We use a value 0.02 as the minimum reachability in this paper.

3. Drop off all paths where the destination computer does not 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.

3.2.2 Merging Algorithm. Algorithm 2 is designed to use an attack graph to merge evidence graphs that have missing evidence. It 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 $\pi[u']$ in the attack graph A (if the evidence is missing in both 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 $\pi[h]$ and h in G_1 or G_2, which correspond to evidence edges between $\pi[u']$ and u' in G. If these evidence edges are found, they are added as the evidence edges between $\pi[u']$ and u' in G(ine 25 to line 27). If not, the evidence is missing in both the sub evidence graphs G_1 and G_2. In this case, lines 29 to 31 traverse all derived nodes in attack graph A to find the corresponding nodes of $\pi[u']$ 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 evidence graph G as the evidence edges between $\pi[u']$ and u'.

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



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 $n_1$ 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 $\in$ V[$G_1$] or h $\in$ V[$G_2$]
5.      do if color[h] == white
6.          then DFS-VISIT (h)
End

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

MERGING ($\pi$[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 $\pi$[h]
23.                  then $\pi$[u '] ← n
24.                  else $\pi$[u '] ← NIL
25.              if $\pi$[u '] $\neq$ NIL and E($\pi$[h], h) $\neq$ NIL
26.                  then E($\pi$[u '], u ') ← E($\pi$[h], h)
27.                       P(E($\pi$[u '], u ')) ← p(E($\pi$[h], h))
28.              else
29.                  for each derived node m in A
30.                      do if m is identical to $\pi$[u '] or u '
31.                          then color[$\pi$[u ']] ← BLACK, color[u '] ← BLACK
32.              E($\pi$[u '], u ') ← Path between two marked nodes in A
33. return
```

3.3 Complexity Analysis

We use “n”and “e”to represent the numbers of 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 add all these run time up, 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 to find attack steps corresponding to missing evidence in the evidence graphs. 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. [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, an $O(N^2)$ size can still be a problem. By using the methods in 3.2.1, we could greatly reduce the number of N. Besides, because an evidence graph is usually smaller than an attack graph, if we only keep the hosts involved in evidence graphs in the corresponding attack graph, the size of the attack graph can be even smaller. Therefore, the complexity of algorithm 2 is polynomial instead of exponential, because "m"(equivalent to N) is polynomial.

4. An Example Of An Attack Scenario

We simulated an attack scenario by using a small example network in Figure 3. In this network, the external firewall controls network access from the Internet to the enterprise network, where the Apache Tomcat webserver using Firefox 3.6.17 hosts a webpage that allows Internet users' visit via port 8080. The internal firewall controls the access to MySQL database server and fileserver. The webserver can access the database server via the default port 3306 and the fileserver through the NFS protocol. There are also workstations with IE6 installed in this network, which have direct access to internal database server.

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 use two methods:(1) a SQL injection attack by exploiting a java servlet code on the webserver; (2) direct access by compromising a workstation. The java servlet code on the webserver does not sanitize input values: *theStatement.executeQuery("select * from profiles where name='Alice' AND password=' "+passWord+" ' ");*, which corresponds to CWE89 in NVD

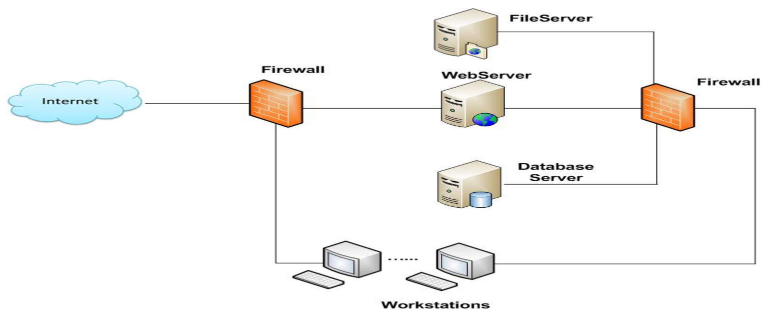


Figure 3. Experiment Network

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-2011-236
Victim	Database server/File server	CVE-2003-0252

[12]. The workstations run Windows XP SP3 operating system with IE6, which has a vulnerability CVE-2009-1918 [9]. This vulnerability allows an attacker to use social engineering, easily enabling the attacker's machine to control the workstations. For the attack plan on the fileserver, we assume to use vulnerability in the NFS service daemons (CVE-2003-0252) to compromise the file server. The webserver, as a stepstone to attack towards fileserver, can be compromised by using its vulnerability CVE-2011-2365 on Firefox 3.6.17. All of the attack roles and vulnerabilities on all computers are stated in Table 1.

4.1 Use Algorithm 1 to Merge Evidence Graphs

We modeled evidence from the above attack scenario as a vector (id, source, destination, content, time stamp) and used Graphviz [11] to generate two evidence graphs ("a)"and "b)"in Figure 4). In both graphs, solid edges represent primary evidence and second evidence, which has coefficient 1 and 0.8 respectively. Dotted edges represent expert knowledge with a coefficient 0.5. The probabilities for the evidence edges and hosts are calculated by using Definition 2.

We applied algorithm 1 to both evidence graphs to generate the integrated evidence graph("c)"in Figure 4). Because the workstations in the two sub evidence graphs are involved in the same attack, they

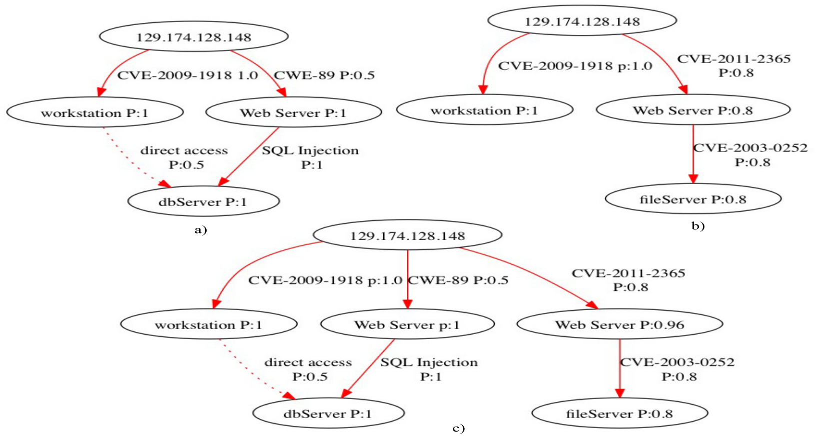


Figure 4. Evidence graphs for the experiment network attack scenario

are merged into a single node with an increased probability $p'(h) = p(h.G_1)+p(h.G_2)-p(h.G_1)\times p(h.G_2) = 1+1-1\times 1 = 1$. 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.

4.2 Use Algorithm 2 to Merge Evidence Graphs

4.2.1 Reducing Attack Graph Complexity. In order to have a succinct visual effect, we only took two workstations out of many in Figure 3 for an attack graph generation. With all vulnerability information mentioned above, we generated an attack graph by using MulVAL[5](Figure 5). In this graph, 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 6 that obviously has a smaller complexity than Figure 5 because of less host nodes. Correspondingly, the grouped host nodes have updated probabilities as follows.

$$p'(21) = p(21) + p(31) - p(21) \times p(31) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96;$$
$$p'(20) = p(20) + p(30) - p(20) \times p(30) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96;$$
$$p'(19) = p(19) + p(29) - p(19) \times p(29) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96;$$
$$p'(18) = p(18) + p(28) - p(18) \times p(28) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96.$$

4.2.2 Merging Evidence Graphs . By referring to the compact attack graph in Figure 6, we used Algorithm 2 to integrate the two evidence graphs. In order to simulate having missing evidence in evidence graphs, we assume the evidence from web server to fileserver in graph "b)"in Figure 4 has not been found. The corresponding integrated

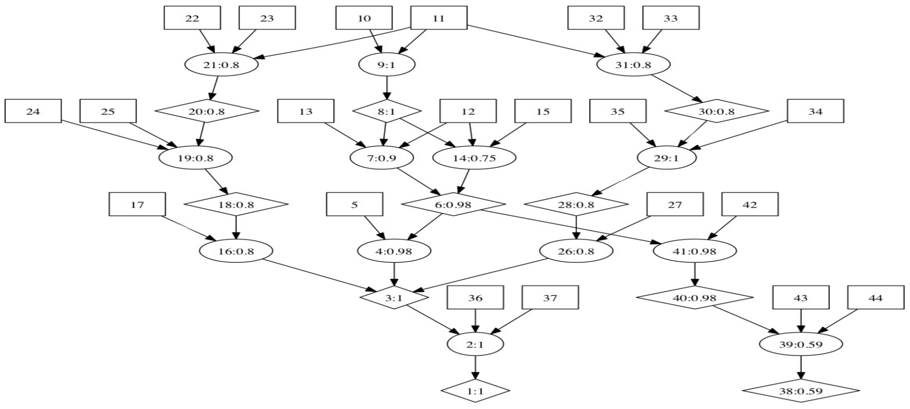


Figure 5. Attack graph with two workstations

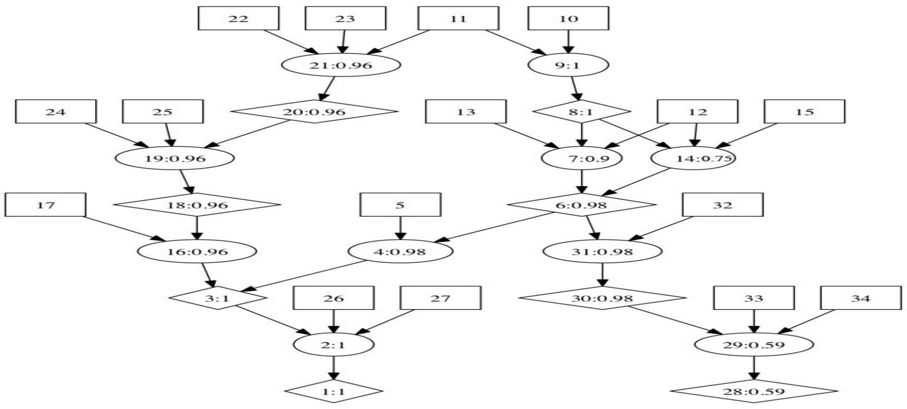


Figure 6. Attack graph with grouped workstations

evidence graph is in Figure 7. In this graph, the edges colored in red are merged from both evidence graphs, and the black attack path between "Web Server "and "fileserver "is the corresponding attack path (between node 6 and node 28) from the attack graph. This new added attack path between the "Web Server"and the "fileserver"means the webserver has been used as a step stone to comprise file server by using the vulnerability of "CVE-2003-0252"in NVD[9] (RPC requests to mountd that does not contain newlines).

5. Conclusion

We developed two algorithms to integrate sub probabilistic evidence graphs to an integrated probabilistic evidence graph. Our first algo-

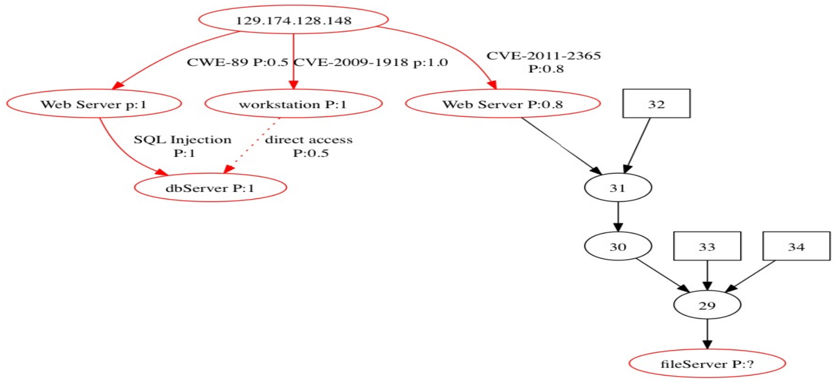


Figure 7. Integrated evidence graph by referring to Figure 6

rithm directly merges probabilistic evidence graphs without paying any attention to infrastructural vulnerabilities. Our second algorithm merges multiple evidence graphs in the presence of an attack graph that captures the infrastructural vulnerabilities and their dependencies in the form 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 in getting 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. 336345 (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] 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 (DBSEC08), 2008.

[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 (CSFW02), 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&P02), pages 273284, 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 (CCS02), pages 217224, 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.