



**HAL**  
open science

## Creating Integrated Evidence Graphs for Network Forensics

Changwei Liu, Anoop Singhal, Duminda Wijesekera

► **To cite this version:**

Changwei Liu, Anoop Singhal, Duminda Wijesekera. Creating Integrated Evidence Graphs for Network Forensics. 9th International Conference on Digital Forensics (DF), Jan 2013, Orlando, FL, United States. pp.227-241, 10.1007/978-3-642-41148-9\_16 . hal-01460608

**HAL Id: hal-01460608**

**<https://inria.hal.science/hal-01460608v1>**

Submitted on 7 Feb 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

## Chapter 16

# CREATING INTEGRATED EVIDENCE GRAPHS FOR NETWORK FORENSICS

Changwei Liu, Anoop Singhal and Duminda Wijesekera

**Abstract** Probabilistic evidence graphs can be used to model network intrusion evidence and the underlying dependencies to support network forensic analysis. The graphs provide a means for linking the probabilities associated with different attack paths with the available evidence. However, current work focused on evidence graphs assumes that all the available evidence can be expressed using a single, small evidence graph. This paper presents an algorithm for merging evidence graphs with or without a corresponding attack graph. The application of the algorithm to a file server and database server attack scenario yields an integrated evidence graph that shows the global scope of the attack. The global graph provides a broader context and better understandability than multiple local evidence graphs.

**Keywords:** Network forensics, probabilistic evidence graphs, attack graphs

## 1. Introduction

Evidence graphs built from network attacks can be used by network forensic practitioners to model evidence of network intrusions [13]. The nodes in an evidence graph represent computers that are of interest in a network forensic investigation and the edges represent the dependencies between evidentiary items.

Attack graphs have been used to analyze security vulnerabilities and their dependencies in enterprise networks. By incorporating quantitative metrics, probabilistic attack graphs can help estimate the probabilities of successful network attacks [11]. While a good probabilistic attack graph provides attack success probabilities to assist forensic investigations and network configuration adjustments, an attack graph with inaccurate attack paths or attack success probabilities can mislead in-

investigators and network administrators. To address this limitation, Liu, *et al.* [6] have proposed the adjustment of an inaccurate probabilistic attack graph by mapping a corresponding probabilistic evidence graph to the attack graph. Their algorithm can map an evidence graph to a small-scale attack graph, but it does not provide a solution for mapping multiple evidence graphs to a single large-scale attack graph. The reason for having multiple attack graphs is clear – evidence is collected from multiple systems and digital forensic practitioners must combine the individual graphs to obtain a complete view of the evidence trail left by a distributed attack. To the best of our knowledge, the problem of integrating evidence graphs has not been addressed by other researchers.

## 2. Background

Sheyner, *et al.* [10] have defined state-based attack graphs whose nodes correspond to global states of a system and edges correspond to state transitions. However, a state-based attack graph can yield an exponential number of states because it encodes nodes as a collection of Boolean variables to cover the entire set of possible network states. A more compact graph representation expresses exploits or conditions as nodes and attack dependencies as edges [1, 4, 9]. For example, Wang and Daniels [13] have defined the notion of an evidence graph whose nodes represent host computers involved in an attack and edges represent forensic evidence that correlate the hosts.

The NIST National Vulnerability Database (NVD) [8] standardizes vulnerability metrics that assign success probabilities to exposed individual vulnerabilities. The NVD provides the probabilities of attacks used in attack graphs [9, 12]. Evidence graphs also use quantitative metrics to estimate the admissibility of evidence and the certainty that a particular host is involved in an attack [13].

*Definition 1. Evidence Graph* [6, 13]: 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 each consisting of a particular data item that indicates activities between the source and target machines,  $N - Attr$  is a set of node attributes that include the host ID, attack states, timestamp and host importance, and  $E - Attr$  is a set of edge attributes each consisting of an event description, evidence impact weight, attack relevance and host importance. Functions  $L : N \rightarrow 2^{N - Attr}$  and  $T : E \rightarrow 2^{E - Attr}$  assign attribute-value pairs to nodes and edges, respectively.

The  $N - Attr$  attack states comprise one or more of attack source, target, stepping-stone and affiliated host computers. Affiliated hosts are

computers that have suspicious interactions with an attacker, a victim or stepping-stone host [6].

*Definition 2. Probabilistic Evidence Graph* [6]: In a probabilistic evidence graph  $G = (N, E, N - Attr, E - Attr, L, T, p)$ , the probability assignment functions  $p \in [0, 1]$  for an evidence edge  $e \in E$  and a victim host  $n \in N$  are defined as:

- $p(e) = c \times w(e) \times r(e) \times h(e)$  where  $w$ ,  $r$  and  $h$  are the weight, relevance and importance of the evidence edge  $e$ , respectively [13]. The coefficient  $c$  indicates the category of evidence, which includes primary evidence, secondary evidence and hypothesis testing from expert knowledge that are assigned values of 1, 0.8 and 0.5, respectively.
- $p(n) = p[(\cup e_{out}) \cup (\cup e_{in})]$  where  $\cup e_{out}$  contains all the edges whose source computer is a host  $n$  with a particular attack-related state, and  $\cup e_{in}$  contains all the edges whose target computer is  $n$  with the same state.

The weight of an evidence edge with a value in  $[0, 1]$  represents the impact of the evidence on the attack. The relevance is the measure of the evidence impact on the attack success at this specific step, which can take one of three possible values: 0 corresponding to an irrelevant true positive, 1 corresponding to a relevant true positive, and 0.5 corresponding to a situation that cannot be verified. Lastly, the importance of evidence (value in  $[0, 1]$ ) is the greater of the importance values of the two hosts connected by the evidence edge.

Two kinds of evidence can be used to construct evidence graphs: primary evidence that is explicit and direct, and secondary evidence that is implicit or circumstantial. In some cases, evidence does not exist or has been destroyed and a subject matter expert may insert an edge corresponding to his expert opinion.

*Definition 3. Logical Attack Graph* [6, 9]:  $A = (N_r, N_p, N_d, E, L, G)$  is a logical attack graph where  $N_r$ ,  $N_p$  and  $N_d$  are sets of derivation, primitive and derived fact nodes,  $E \subseteq ((N_p \cup N_d) \times N_r) \cup (N_r \times N_d)$ ,  $L$  is a mapping from a node to its label, and  $G \subseteq N_d$  is the final goal of an attacker.

Figure 1 shows a logical attack graph. A primitive fact node (box) represents a specific network configuration or vulnerability information corresponding to a host. A derivation node (ellipse) represents 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 (diamond), which is satisfied by the input facts.

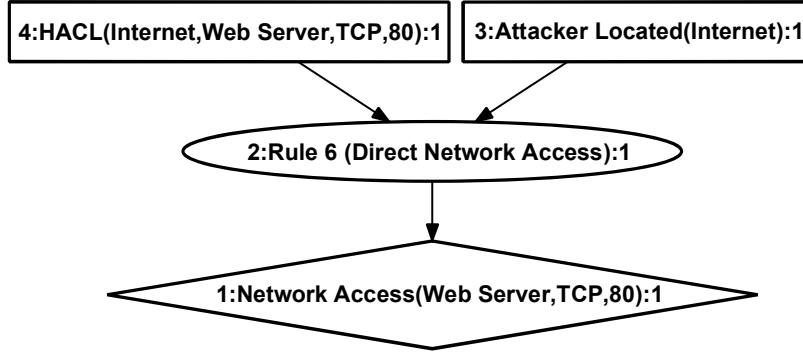


Figure 1. Example attack graph.

*Definition 4. Cumulative Probability Function* [6]: Given a probabilistic attack graph  $A = (N_r, N_p, N_d, E, L, G, p)$ , the function  $p : N_p \cup N_r \cup N_d \rightarrow [0, 1]$  assigns probabilities to nodes. Exploits ( $e$ ) are represented as derivation nodes  $N_r$  and conditions ( $c$ ) as primitive facts  $N_p$  or derived facts  $N_d$ . The cumulative probability function  $P$  for  $e$  and  $c$  of a probabilistic attack graph  $P : N_p \cup N_r \cup N_d \rightarrow [0, 1]$  is computed from  $p : N_p \cup N_r \cup N_d \rightarrow [0, 1]$  as follows:

- $P(e) = p(e) \times \prod P(c)$  where  $e \in N_r$  and  $c$  are conditions that include primitive facts and derived facts from a prior step.
- $P(c) = p(c) = 1$  if the condition is a primitive fact that is always satisfied.
- $P(c) = p(c) \times \bigoplus P(e)$  where  $c$  is the condition of the derived fact node that is derived from derivation nodes ( $e$ ).

*Definition 5. Sub 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  $A = (N_r, N_p, N_d, E, L, G)$  if the following conditions hold:

- $N'_r, N'_p, N'_d \subseteq N_r, N_p, N_d$ .
- $E' \subseteq E \cap ((N_1, N_2) \in E' \rightarrow N_1, N_2 \in N'_r \cup N'_p \cup N'_d)$ .
- $G' \subseteq G$ , and  $L' \subseteq L$ .

*Definition 6. Similar Nodes*: In a logical attack graph  $A = (N_r, N_p, N_d, E, L, G)$  or evidence graph  $G = (N, E, N - Attr, E - Attr, L, T)$ ,  $N_{1d}$  and  $N_{2d}$  are similar if both  $N_{1d}$  and  $N_{2d} \in A$  satisfy the equalities  $N_{1d} \equiv N_{2d}$ ,  $N_{1r} \equiv N_{2r}$  and  $N_{1p} \equiv N_{2p}$ , represented as  $N_{1d} \approx N_{2d}$ .

Nodes  $N_1$  and  $N_2$  are similar if they are the same type of hosts  $N_1 \wedge N_2 \in G$ , 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  $N_2$ .

Wang and Daniels [13] have suggested normalizing all evidence to five components: (i) ID; (ii) source; (iii) destination; (iv) content; and (v) timestamp. These are used as edges to connect hosts in a time sequence in order to produce an evidence graph.

### 3. Merging Sub Evidence Graphs

Merging the probabilistic evidence graphs of all the victim hosts and attack evidence in a network provides a global attack description. In order to generate a valid integrated evidence graph, the following guidelines must be followed to ensure that the probabilities of merged host nodes and evidence edges do not violate Definition 2.

- A different node is used to represent each attack on the same host.
- If a victim host is exploited by the same vulnerability in different attacks, the nodes are merged as one node, which is given an increased probability  $p'(h) = p(h)_{G_1} + p(h)_{G_2} - p(h)_{G_1} \times p(h)_{G_2}$ . Note that  $p'(h)$  is the increased probability of the merged node, and  $p(h)_{G_1}$  and  $p(h)_{G_2}$  are the host probabilities in evidence graphs  $G_1$  and  $G_2$ .
- Similar hosts involved in similar attacks are merged together with an increased probability  $p'(h) = p(h)_{G_1} + p(h)_{G_2} - p(h)_{G_1} \times p(h)_{G_2}$ .

When nodes are merged, the corresponding edges that represent the same evidence should also be merged. For evidence probabilities  $p(e)_{G_1}$  and  $p(e)_{G_2}$  in evidence graphs  $G_1$  and  $G_2$ , the merged evidence probability is increased to  $p'(e) = p(e)_{G_1} + p(e)_{G_2} - p(e)_{G_1} \times p(e)_{G_2}$  where  $p'(e)$  is the new merged evidence edge probability.

Figures 2(a) and 2(b) show two evidence graphs and Figure 2(c) shows their integrated evidence graph. In the two evidence graphs, assume that Host2 is attacked by using the same vulnerability from Host1 and Host3, and assume that  $\text{Host1} \approx \text{Host3}$ . Host2 in Figures 2(a) and 2(b) can be merged to a single Host2 node with increased probability  $p'(h_2) = 0.5 + 0.9 - 0.5 \times 0.9 = 0.95$ . Also, Host1 and Host3 from the two graphs can be merged as Host with increased probability  $p'(h) = p(h_1) + p(h_3) - p(h_1) \times p(h_3) = 0.5 + 0.5 - 0.5 \times 0.5 = 0.75$ . The merged evidence is, therefore, increased to  $p'(e) = p(e_1) + p(e_3) - p(e_1) \times p(e_3) = 2 \times 0.5 - 0.5 \times 0.5 = 0.75$ .

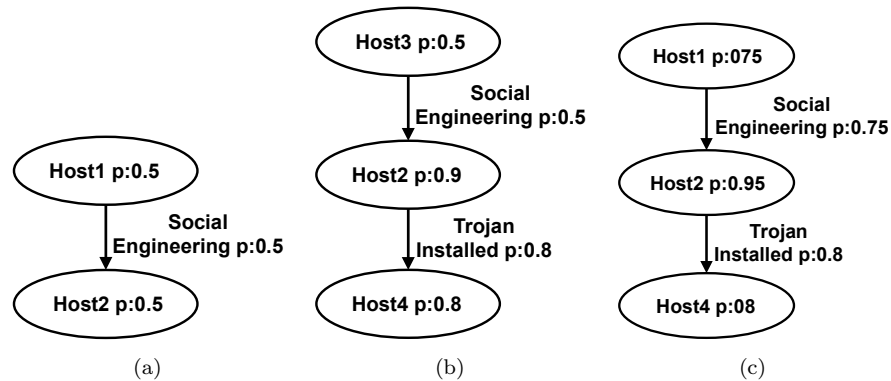


Figure 2. Merging evidence graphs.

If the sub evidence graphs are well constructed with no tainted or missing evidence, then the sub evidence graphs can be merged into an integrated evidence graph. If not, an attack graph generated from the same network is used as a reference to integrate the sub evidence graphs.

### 3.1 Merging Without an Attack Graph

Algorithm 1 merges two evidence graphs. Starting with the two victim host nodes, the depth-first search algorithm [2] traverses all the host nodes in both evidence graphs and merges them to create an integrated evidence graph. The algorithm uses a coloring scheme to keep track of nodes in the two sub evidence graphs. All the nodes are initialized as being **white**; they are colored **gray** when they are being considered and their children have not yet been fully examined. A host is colored **black** after all its children are fully examined. The merging is complete when all the nodes in the two evidence graphs are colored **black**.

Algorithm 1 takes two sub evidence graphs  $G_1$  and  $G_2$  as input and creates their integrated evidence graph  $G$ . The algorithm checks  $G_1$  and  $G_2$  to see if there is an identical host node in  $G$ . Note that identical host means the same machine is present in both evidence graphs and the same vulnerability is exploited. If a host node in  $G$  is found to be identical to a node in  $G_1$  or  $G_2$ , then the probability of the host in  $G$  is increased. If there is no identical host in  $G$ , then the node and its corresponding evidence from  $G_1$  or  $G_2$  are added to  $G$ .

### 3.2 Merging Using an Attack Graph

Under some circumstances, the constructed evidence graph has nodes that are not connected because of missing evidence edges. In such a case, merging evidence graphs becomes problematic. To address this

---

**Algorithm 1** : Merging Probabilistic Evidence Graphs.

**Input:** Two 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 evidence graph  $G = (N, E, N - Attr, E - Attr, L, T)$ .

---

**begin:**

```

1: for all nodes  $n_1 \in G_1$  and  $n_2 \in G_2$  do
2:   color[ $n_1$ ]  $\leftarrow$  white, color[ $n_2$ ]  $\leftarrow$  white
3:    $\pi[n_1] \leftarrow$  NIL,  $\pi[n_2] \leftarrow$  NIL
4: end for
5: for all nodes  $h \in V[G_1] \vee h \in V[G_2]$  do
6:   if color[ $h$ ] = white then
7:     DFS-VISIT( $h$ )
8:   end if
9: end for
end

```

**DFS-VISIT( $h$ )**

```

1: color[ $h$ ]  $\leftarrow$  gray
2: MERGING( $\pi[h], h, G$ )
3: for all  $v \in Adj[h]$  do
4:   if color[ $v$ ] = white then
5:      $\pi[v] \leftarrow h$ 
6:     DFS-VISIT( $v$ )
7:   end if
8:   color[ $h$ ]  $\leftarrow$  black
9: end for
10: return

```

**MERGING( $\pi[h], h, G$ )**

```

1: for all nodes  $u \in G$  do
2:   if  $u$  is identical to  $h$  then
3:      $p'(u) = p(u) + p(h) - p(u) \times p(h)$ ,  $p'(u.e) = p(u.e) + p(h.e) - p(u.e) \times p(h.e)$ 
4:   else
5:     add  $h$  to  $G$  as  $u'$ ,  $p(u') = p(h)$ 
6:     for all nodes  $n \in G$  do
7:       if  $n$  is identical to  $\pi[h]$  then
8:          $\pi[u'] \leftarrow n$ 
9:       else
10:         $\pi[u'] \leftarrow$  NIL
11:      end if
12:      if  $\pi[u'] \neq$  NIL then
13:         $E(\pi[u'], u') \leftarrow E(\pi[h], h)$ ,  $P(E(\pi[u'], u')) \leftarrow p(E(\pi[h], h))$ 
14:      end if
15:    end for
16:  end if
17: end for
18: return

```

---

problem we use an attack graph of the entire enterprise network to fill in the missing evidence by mapping sub evidence graphs to the attack graph to locate the corresponding attack path.

Runtime overhead is a problem when analyzing large-scale attack graphs [3]. It is impractical to map evidence graphs that have poly-



nomial complexity to an attack graph that has an exponential number of nodes  $V$  and edges  $E$  because the time requirement for a mapping algorithm using a depth-first search is  $O(V + E)$  [2]. The following methods are used to reduce attack graph complexity:

- Hosts that have similar vulnerabilities are grouped together. The grouped node is assigned 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 Host1 and Host2, respectively, that have similar vulnerabilities [3].
- Attack success probability also represents attack reachability. The hosts with low reachability are removed [7]. We use a value 0.02 as the minimum reachability.
- All paths for which the destination computer does not expand to the desired victim hosts and all hosts that are not involved in the desired attack paths are removed. In addition, if there are multiple exploits at the same host, then the exploits for which the Common Vulnerability Scoring System (CVSS) is smaller than a pre-selected threshold value are omitted.

Algorithm 2 uses an attack graph to merge evidence graphs that have missing evidence. It enhances Algorithm 1 by looking up the missing evidence in attack graph  $A$ . The alterations are in the MERGING algorithm, which searches for the missing evidence between a new added node  $u'$  and its parent node  $\pi[u']$ . If the evidence is missing in both  $G_1$  and  $G_2$ , then it is searched for in  $A$ . Specifically, Lines 6–8 in MERGING 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 evidence edges between  $\pi[u']$  and  $u'$  in  $G$ . If not, the evidence is missing in both the sub evidence graphs and Lines 15–19 traverse all the derived nodes in attack graph  $A$  to find the corresponding nodes of  $\pi[u']$  and  $u'$  from  $G$ , and color them **black**. Line 20 adds the attack paths that are found between the two marked nodes in the attack graph  $A$  to the integrated evidence graph  $G$  as evidence edges between  $\pi[u']$  and  $u'$ .

### 3.3 Complexity Analysis

The complexity analysis uses  $n$  and  $e$  to represent the numbers of all the nodes and edges in the two evidence graphs. In Algorithm 1, Lines 1–3 have  $O(n)$  time because the three lines only go through each node once. Lines 4–6 traverse every node to do a depth-first search (DFS-VISIT),

---

**Algorithm 2** : Integrating Evidence Graphs Using an Attack Graph.

**Input:** Two evidence graphs  $G_i = (N_i, E_i, N_i - Attr, E_i - Attr, L_i, T_i)$  with victim nodes  $v_i$  for  $i = 1, 2$ . One attack graph  $A = (N_p, E_p, N_p - Attr, E_p - Attr, L_p, T_p)$ .

**Output:** One evidence graph  $G = (N, E, N - Attr, E - Attr, L, T)$ .

---

**begin:**

```

1: for all nodes  $n_1 \in G_1$  and  $n_2 \in G_2$  do
2:   color[ $n_1$ ]  $\leftarrow$  white, color[ $n_2$ ]  $\leftarrow$  white,  $\pi[n_1] \leftarrow$  NIL,  $\pi[n_2] \leftarrow$  NIL
3: end for
4: for each node  $h \in V[G_1]$  or  $h \in V[G_2]$  do
5:   if color[ $h$ ] = white then
6:     DFS-VISIT( $h$ )
7:   end if
8: end for
end

```

**DFS-VISIT( $h$ )**

```

1: color[ $h$ ]  $\leftarrow$  gray, MERGING( $\pi[h], h, G, A$ )
2: for all  $v \in Adj[h]$  do
3:   if color[ $v$ ]  $\equiv$  white then
4:      $\pi[v] \leftarrow h$ , DFS-VISIT( $v$ )
5:   end if
6: end for
7: color[ $h$ ]  $\leftarrow$  black
8: return

```

**MERGING( $\pi[h], h, G, A$ )**

```

1: for all nodes  $u \in G$  do
2:   if  $u$  is identical to  $h$  then
3:      $p'(u) = p(u) + p(h) - p(u) \times p(h)$ ,  $p'(u.e) = p(u.e) + p(h.e) - p(u.e) \times p(h.e)$ 
4:   else
5:     add  $h$  to  $G$  as  $u'$ ,  $p(u') = p(h)$ 
6:     for all nodes  $n \in G$  do
7:       if  $n$  is identical to  $\pi[h]$  then
8:          $\pi[u'] \leftarrow n$ 
9:       else
10:         $\pi[u'] \leftarrow$  NIL
11:      end if
12:      if  $\pi[u'] \neq$  NIL and  $E(\pi[h], h) \neq$  NIL then
13:         $E(\pi[u'], u') \leftarrow E(\pi[h], h)$ ,  $P(E(\pi[u'], u')) \leftarrow p(E(\pi[h], h))$ 
14:      else
15:        for all derived nodes  $m \in A$  do
16:          if  $m$  is identical to  $\pi[u'] \vee u'$  then
17:            color[ $\pi[u']$ ]  $\leftarrow$  black, color[ $u'$ ]  $\leftarrow$  black
18:          end if
19:        end for
20:         $E(\pi[u'], u') \leftarrow$  Path between two marked nodes in  $A$ 
21:      end if
22:    end for
23:  end if
24: end for
25: return

```

---

which has  $O(n)$  time. DFS-VISIT calls the MERGING function, which includes two loops that have  $O(n^2)$  time. Depth-first search traverses

every edge and node in both evidence graphs, which takes  $O(n+e)$  time as proved in [2]. Upon totaling these times, Algorithm 1 has 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  $n$  because there are usually at most three edges between two nodes, so the execution time is polynomial.

Algorithm 2 is similar to Algorithm 1, except that Algorithm 2 has to traverse all the derived nodes in the corresponding attack graph to find the attack steps corresponding to missing evidence in the evidence graphs. This is done by a loop in MERGING (Lines 15–19). Using  $m$  to represent the 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  $m$  because  $e$  is polynomial to  $n$ .

Ou, *et al.* [9] have proved that a logical attack graph for a network with  $N$  machines has a size at most  $O(N^2)$ . However, in a large-scale network with large  $N$ , an  $O(N^2)$  size can still be a problem. By reducing the number  $N$  to only the hosts involved in evidence graphs in the corresponding attack graph, the size of the attack graph can be made smaller. Therefore, the complexity of Algorithm 2 is polynomial instead of exponential because  $m$  (equivalent to  $N$ ) is polynomial.

#### 4. Example Attack Scenario

Figure 3 shows the network used to simulate an attack scenario. In this network, an external firewall controls access from the Internet to the enterprise network, where an Apache Tomcat web server using Firefox 3.6.17 hosts a website that allows access to Internet users via port 8080. The internal firewall controls access to a MySQL database server that also functions as a file server. The web server can access the database server via the default port 3306 and the file server through the NFS protocol. Workstations with Microsoft Internet Explorer 6 (IE6) installed have direct access to the internal database server.

The attacker's objective is to gain access to database tables stored on the database server and to compromise the file server. The attack plan involves: (i) an SQL injection attack that exploits a Java servlet on the web server; and (ii) direct access by compromising a workstation.

The Java servlet on the web server does not sanitize inputs; it is vulnerable to: `theStatement.executeQuery("SELECT * FROM profiles WHERE name='Alice' AND password=' "+passWord+" ' ");`, corresponding to CWE89 in NVD [5]. The workstations run Windows XP SP3 with IE6, which has the vulnerability CVE-2009-1918 [8]. This vulnerability allows an attacker to use social engineering to enable his machine to control the targeted workstation. The file server attack plan uses a

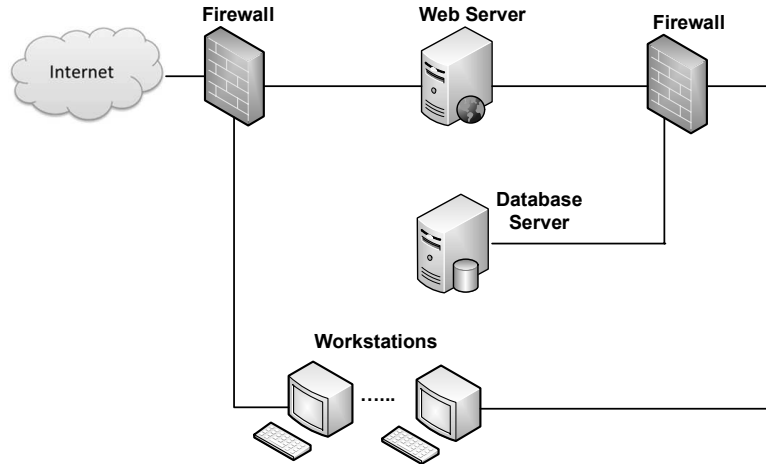


Figure 3. Experimental network.

Table 1. Attack role configuration.

System	Role	Vulnerability
129.174.128.148	Attacker	
Workstation	Stepping Stone	CVE-2009-1918
Web Server	Stepping Stone/Affiliated	CWE89/CVE-2011-2365
Database Server/File Server	Victim	CVE-2003-0252

vulnerability in NFS service daemons (CVE-2003-0252) to compromise the file server. The web server, which is a stepping stone to attack the file server, can be compromised by exploiting vulnerability CVE-2011-2365 in Firefox 3.6.17. Table 1 lists the attack roles and vulnerabilities.

#### 4.1 Merging Graphs Using Algorithm 1

The evidence from the attack scenario is modeled as a vector (ID, source, destination, content, timestamp) and appears in the evidence graphs in Figures 4(a) and 4(b). In both graphs, solid edges represent primary evidence and secondary evidence, which have coefficients of 1 and 0.8, respectively. Dotted edges represent expert knowledge with a coefficient of 0.5. The probabilities for the evidence edges and hosts are calculated using Definition 2.

Executing Algorithm 1 yields the integrated evidence graph in Figure 4(c). Because the workstations in the two sub evidence graphs are involved in the same attack, they are merged to a single node with an increased probability  $p'(h) = p(h.G_1) + p(h.G_2) - p(h.G_1) \times p(h.G_2) =$

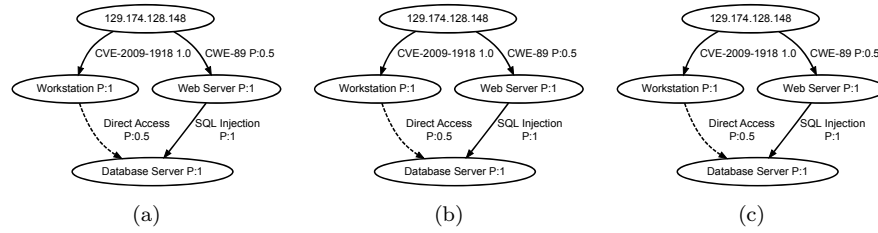


Figure 4. Evidence graphs for the experimental network attack scenario.

$1 + 1 - 1 \times 1 = 1$ . Note that  $p(h.G_1)$  and  $p(h.G_2)$  are the probabilities of  $p(h)$  in evidence graphs  $G_1$  and  $G_2$ , respectively.

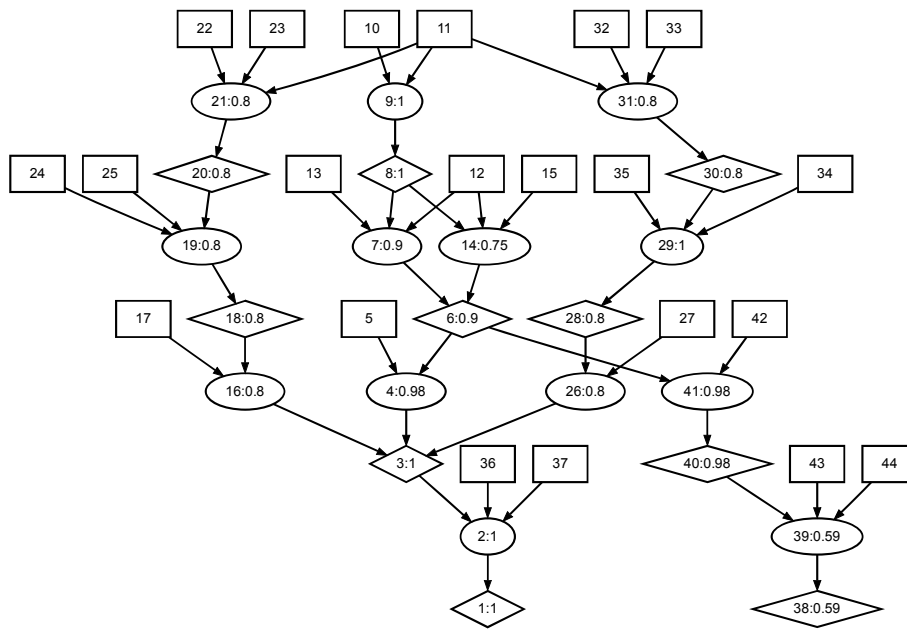


Figure 5. Attack graph with two workstations.

### 4.2 Merging Graphs Using Algorithm 2

Figure 5 shows the attack graph with the vulnerability information in Table 1. The graph shows two attack paths on the left and right sides. Merging the two workstations yields Figure 6, which has less computational complexity than Figure 5. The grouped host nodes have updated probabilities of:

$$\begin{aligned}
 p'(21) &= p(21) + p(31) - p(21) \times p(31) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96 \\
 p'(20) &= p(20) + p(30) - p(20) \times p(30) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96
 \end{aligned}$$

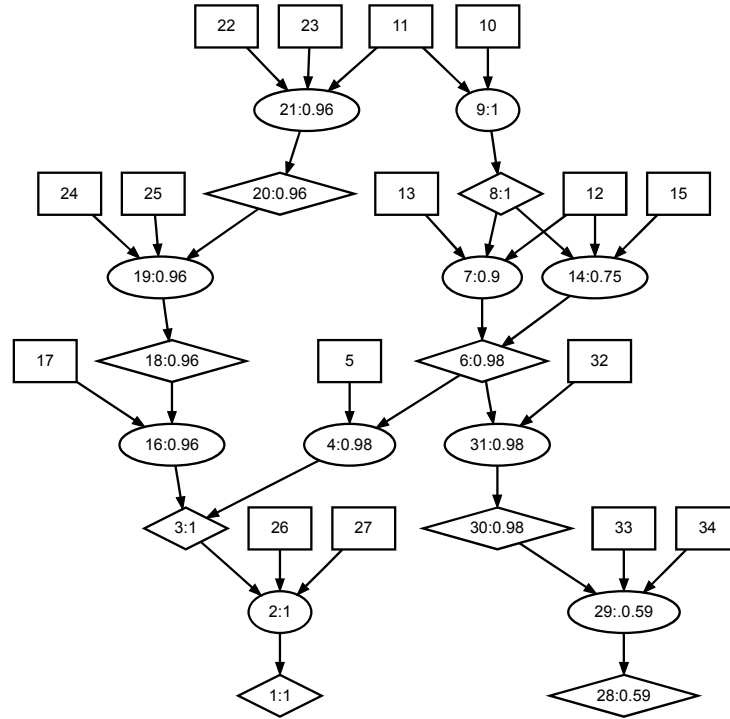


Figure 6. Attach graph with grouped workstations.

$$p'(19) = p(19) + p(29) - p(19) \times p(29) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96$$

$$p'(18) = p(18) + p(28) - p(18) \times p(28) = 0.8 + 0.8 - 0.8 \times 0.8 = 0.96$$

Algorithm 2 uses the compact attack graph in Figure 6 to integrate the two evidence graphs. In order to simulate missing evidence in the evidence graphs, we assume that the evidence from the web server to the file server in the graph in Figure 4(b) has not been found. The corresponding integrated evidence graph is shown in Figure 7. In this graph, the dashed edge is merged from the two evidence graphs and the attack path between the web server and the file server 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 file server indicates that the web server has been used as a stepping stone to compromise the file server using vulnerability CVE-2003-0252.

## 5. Conclusions

Two algorithms are presented for merging probabilistic sub evidence graphs to create an integrated probabilistic evidence graph. The first algorithm merges two probabilistic evidence graphs without using an

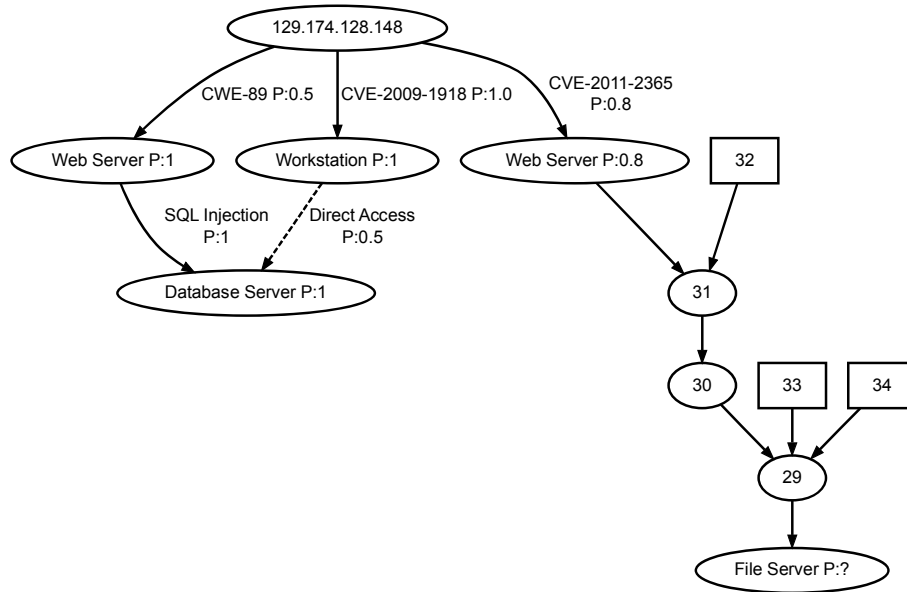


Figure 7. Integrated evidence graph.

attack graph of infrastructure vulnerabilities. The second algorithm, which merges multiple evidence graphs using an attack graph, is used in situations where evidence has been tampered with or is missing. Because the second algorithm can have a lengthy execution time for an attack graph with high complexity, certain approximations can be applied to shrink attack graphs to reduce the execution time.

Note that this paper is not subject to copyright in the United States. Commercial products are identified in the paper in order to present and evaluate certain procedures. The identification of the products does not imply a recommendation or endorsement by the National Institute of Standards and Technology or the U.S. Government, nor does it imply that the identified products are necessarily the best available products for the task.

## References

- [1] P. Ammann, D. Wijesekera and S. Kaushik, Scalable, graph-based network vulnerability analysis, *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, pp. 217–224, 2002.
- [2] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2009.

- [3] J. Homer, A. Varikuti, X. Ou and M. McQueen, Improving attack graph visualization through data reduction and attack grouping, *Proceedings of the Fifth International Workshop on Visualization for Cyber Security*, pp. 68–79, 2008.
- [4] K. Ingols, R. Lippmann and K. Piwowarski, Practical attack graph generation for network defense, *Proceedings of the Twenty-Second Annual Computer Security Applications Conference*, pp. 121–130, 2006.
- [5] S. Jha, O. Sheyner and J. Wing, Two formal analyses of attack graphs, *Proceedings of the Fifteenth Computer Security Foundations Workshop*, p. 49, 2002.
- [6] C. Liu, A. Singhal and D. Wijesekera, Mapping evidence graphs to attack graphs, *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 121–126, 2012.
- [7] V. Mehta, C. Bartzis, H. Zhu, E. Clarke and J. Wing, Ranking attack graphs, *Proceedings of the Ninth International Conference on Recent Advances in Intrusion Detection*, pp. 127–144, 2006.
- [8] National Institute of Standards and Technology, National Vulnerability Database, Version 2.2, Gaithersburg, Maryland ([nvd.nist.gov](http://nvd.nist.gov)).
- [9] X. Ou, W. Boyer and M. McQueen, A scalable approach to attack graph generation, *Proceedings of the Thirteenth ACM Conference on Computer and Communications Security*, pp. 336–345, 2006.
- [10] O. Sheyner, J. Haines, S. Jha, R. Lippmann and J. Wing, Automated generation and analysis of attack graphs, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 273–284, 2002.
- [11] A. Singhal and X. Ou, Security Risk Analysis of Enterprise Networks using Probabilistic Attack Graphs, NIST Interagency Report 7788, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.
- [12] L. Wang, T. Islam, T. Long, A. Singhal and S. Jajodia, An attack graph based probabilistic security metric, *Proceedings of the Twenty-Second Annual IFIP WG 11.3 Conference on Data and Applications Security*, pp. 283–296, 2008.
- [13] W. Wang and T. Daniels, A graph based approach toward network forensic analysis, *ACM Transactions on Information and Systems Security*, vol. 12(1), article no. 4, 2008.