Mapping Evidence Graphs to Attack Graphs

[*]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--Attack graphs compute potential attack paths from a system configuration and known vulnerabilities of a system. Evidence graphs model intrusion evidence and dependencies among them for forensic analysis. In this paper, we show how to map evidence graphs to attack graphs. This mapping is useful for application of attack graphs and evidence graphs for forensic analysis. In addition to helping to refine attack graphs by comparing attack paths in both attack graphs and evidence graphs, important probabilistic information contained in evidence graphs can be used to compute or refine potential attack success probabilities contained in repositories like CVSS. Conversely, attack graphs can be used to add missing evidence or remove irrelevant evidence to build a complete evidence graph. In particular, when attackers use anti-forensics tools to destroy or distort evidence, attack graphs can help investigators recover the attack scenarios and explain the lack of evidence for missing steps. We illustrated the mapping using a database attack as a case study.

Keywords-attack graphs; evidence graphs; attack success probabilities; evidence probabilities; mapping algorithm

I. INTRODUCTION

Currently, attack graphs and evidence graphs are used in security analysis. Attack graphs are used to analyze security vulnerabilities in enterprise networks. An attack graph represent system states as nodes by using a collection of security-related predicates, such as vulnerability on a particular host in a network, and edges as an exploit that takes the system from one state to another [4]. In the attack graphs model, composition of exploits is considered as an attack. Evidence graphs model intrusion evidence in a network, where host computers that interest forensic investigation (i.e. potential evidence) are represented as nodes, and dependencies between such evidence are represented as edges [1].

Many papers address how to construct evidence graphs from collected evidence after an intrusion attack or attack graphs from software vulnerabilities of specific networks. Based on these papers, many researches discuss how to use or refine the two graphs. However, to the best of our knowledge, none of them provides a formal mapping between attack graphs and evidence graphs, which is our contribution in this paper. In particular, we take both kinds of graphs from the same-networked environment and enrich them with quantitative metrics to model the mapping. We show its utility by a database attack case study.

In general, the probability of the potential attack on a specific node in an attack graph decorated with quantitative measures is calculated based on scores from NVD [9] and the corresponding network configurations, which may not accurately reflect the specific network's attack probability. If attack path deviates from an actual attack scenario, it may mislead investigators. [1] shows how to create evidence graphs from evidence, which can help refine such an attack graph. In addition, when evidence is not enough to construct an evidence graph to assist forensics analysis, an attack graph that combines expert knowledge database can help to recover the attack scenario. In this way, our mapping can be used to combine both graphs in helping forensic investigators.

The rest of the paper is organized as follows. Section II describes related works. Section III provides basic definitions. Section IV provides the mapping algorithm. Section V is our case study, showing how to map the constructed evidence graph to attack graph. Lastly, we finish the paper with a conclusion in section VI.

II. RELATED WORK

Many forensics tools are used to analyze data in networked systems. Some are image tools that extract data from physical memory or disk sectors for live or dead analysis [19]. While live analysis has the risk of changing data, dead analysis requires terminating all system processes [20], where neither in itself is complete. Network forensics tools obtain data from capture files of network traffic. In addition, Intrusion detection systems (IDS) data are also used for forensics analysis, which consists of anomaly detectors [21] with false positives and pattern-based detectors that may not be able to capture attacks with unknown patterns.

While forensics tools cannot solely solve network forensics analysis, attack graphs help with this issue. Sheyner et al. defined attack graphs and used them to model multi-stage, multi-host attack paths in a network [17]. Ammann et al. proposed to use the monotonicity assumption to simply the task of modeling attacker actions [7]. Other works, such as a TVA tool [22], use an exploit dependency graph to represent the pre and post-conditions of vulnerable states. Ingols et al. proposed to create a network model using firewall rules and network vulnerability scans, and showed the effect of countermeasures on the system [18]. MulVAL [3,15] generates attack graphs from system configurations and bug-reports. This tool reduces attack graph complexity and shows which system configurations may facilitate attackers to escalate their privileges. Combing attack graphs, the attempts of measuring network security risk have been proposed. NVD by NIST standardizes vulnerability metrics that assign success probabilities to exposed individual vulnerabilities [5,15], which have been used in attack graphs to compute success probabilities of attacks that exploit a series of vulnerabilities [4, 10, 25].

Evidence graphs correlate attack evidence by using network configuration, time stamps and expert systems with fuzzy rules to reconstruct potential attack scenarios from large amount of noisy data [1,13]. Decorated by evidence probability as quantitative values, a probabilistic evidence graph attaches quantitative measures to an evidence graph.

III. BASIC CONCEPTS

This section describes the basic concepts used in the rest of the paper, which are attack graphs and evidence graphs with quantitative metrics.

1. Attack Graphs

There are many definitions on attack graphs, of which we use Ou's logical attack graph definition [3].

Definition 1(Attack graph): $A=(N_r,N_p,N_d,E,L,G)$ is a logical attack graph, where N_r, N_p and N_d are three sets of disjoint nodes (namely derivation, primitive and derived fact nodes respectively) in the graph, $E\subset((N_p\cup N_d)\times N_r)\cup(N_r\times N_d)$, L is a mapping from a node to its label, and $G\subset N_d$ is an attacker's final goal [3].

An example is shown in Figure 1(Appendix has a bigger figure). Primitive fact nodes in boxes include network configuration and vulnerability information on hosts. A derivation node in an oval represents a successful application of an interaction rule, where all facts are its preconditions that are satisfied by its child, a derived node in diamond. That is, the derived nodes are the result of applying interaction rules iteratively on the input facts. The edges in a logical attack graph can only go from a fact node to a derivation node or from a derivation node to a derived fact node. The labeling function maps a fact node to the fact it represents, and a derivation node to the rule used for the derivation.

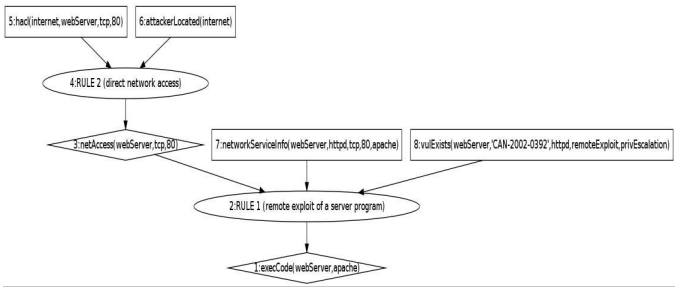


Figure 1: An Example Attack Graph

Definition 2 (Probabilistic Attack Graph)[10]: Given an acyclic attack graph $A=(N_r,N_p,N_d,E,L,G,P)$, and two functions $p:E\rightarrow[0,1]$ and $p:N_d\rightarrow[0,1]$ assigning probabilities of success of an individual exploit (e) and exploiting a condition (c) respectively, the cumulative functions for exploits and sets of conditions $P:e\rightarrow[0,1]$ and $P:c\rightarrow[0,1]$ are defined as follows.

1. $P(c)=p(c)$ if the condition (c) wasn't exploited before.
2. $P(c)=p(c).\oplus P(e)$ Otherwise, and
3. $\mathbf{P(e)\ \ p(e).\Pi\{P(c):c\in S_i\}}$ where $e\in T\subseteq S_i\times S_j$ S is the state of a host after an attack that corresponds to N_d. The operator $\oplus$ is recursively defined as $\oplus P(e)=P(e)$ for any $e\in E$ and $\oplus(S_1\cup S_2)=\oplus S_1+\oplus S_2-\oplus S_1.\oplus S_2$ for any disjoint and non-empty sets $S_1,S_2\subseteq E$.

Definition 2 captures the probability p(e) of an exploit $e\in T$ belonging to $T\subseteq S\times S$, which can be computed from an existing metrics, such as CVSS metric vector etc. [4, 5]. The likelihood of satisfying the pre-conditions (c) of an exploit is p(c), which is always 1 if it is network configuration and less than 1 only when the condition is obtained from prior attacks that have a less than 1 attack success probability. P(e) is the cumulative attack probability computed using a vulnerability that allows accessing a host.

Figure 2 shows a probabilistic attack graph satisfying Definition 2. In this graph, exploit 1 on host 1 results in a state validating post-condition c_1. Either exploit 2 or exploit 3 on host 2 results in a state satisfying post-condition c_2. c_1 and c_2 together are prerequisites to launch exploit 4. The cumulative probabilities are given in the figure (Assume p(c) =1 here). For $P(e_4)=P(c_1)\times P(c_2)\times p(e_4)$ because $P(c_1)=P(e_1)$ and $P(c_2)=P(e_2)+P(e_3)-P(e_2)\times P(e_3)$, $P(e_4)=P(e_1)\times(P(e_2)+P(e_3)-P(e_2)\times P(e_3))\times p(e_4)$.

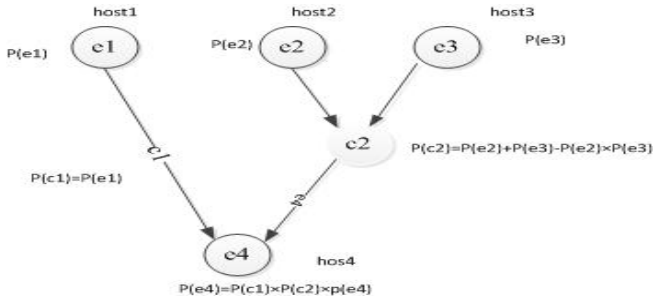


Figure2: Example Probabilistic Attack Graph

2. Evidence Graphs

Definition 3(Evidence Graph)[1]: An evidence graph is a tuple E=(N,E,N-Attr,E-Attr,L,T), where N is a set of nodes representing host computers, $E\subset(N\times N)$ is a set of directed edges consisting of a particular data item of activity between the source and target machines, N-Attr is a set of labels that indicate the attributes of nodes, and E-Attr is a set of labels that indicate the attributes of edges. $L:N\rightarrow 2^{N\text{-}Attr}$ or $T:E\rightarrow 2^{E\text{-}Attr}$ are an assignment of a set of attribute-values pairs to a node or an edge respectively. The following host labels are used in an evidence graph.

(1) **Host ID**: Identification of a suspicious host.
(2) **States**: States of the host nodes are one or many of the attribute values "source", "target", "stepping stone" and "Affiliated". While other attribute value nomenclature is standard, "Affiliated" hosts are those that have suspicious interactions with an attacker, victim or stepping-stone. For example, if a victim host that was compromised in an attack is used as a relay to transfer stolen files is an affiliated host.
(3) **Time stamps**: $T_{activate}$ and T_{latest} time stamps record the initial and latest state of a machine.
(4) **Value**: The value between 0 and 1 indicates the importance of a specific host in a network.

Following edge labels are used in an evidence graph.

(1) **General attributes**: Commonly used attributes of evidence, including the initiator host of event, the target host of the event, the event description and time stamp(s) of the event.
(2) **Weight (w)**: A value between [0, 1] is used to represent the impact of evidence on the attack. For example, port-scan evidence gets less weight than buffer overflow evidence.
(3) **Relevancy(r)**: Measure of impact on attack success.
False/ irrelevant true positive = 0, Unable to verify = 0.5 and Relevant true positive =1

For example, IE6-aurora attack for a Linux machine has relevancy 0, because Linux does not support IE explorer that is the prerequisite of a successful IE6-aurora attack.
(4) **Host Importance (h)**: This is a decimal value that categorizes the importance of a host for an attack plan.

In order to evaluate how much investigators are confident about a host's attack-related states, we use the following definition for a probabilistic evidence graph.

Definition 4 (Probabilistic Evidence Graph): In an acyclic evidence graph E=(N,E,N-Attr,E-Attr,L,T), the probability assignment function p∈[0,1] is defined as follows.

1. $p(e) = c \times w(e) \times r(e) \times h(e)$, where "e" is a particular edge and "w", "r" and "h" are the weight, relevancy and host importance respectively [1]. "c" is a coefficient that indicates the categories of evidence, which are primary evidence, secondary evidence and hypothesis testing from expert knowledge. They are assigned as 1, 0.8 and 0.5 respectively in this paper as examples.
2. $p(h) = p[(\cup e_h.out) \cup (\cup e_h.in)]$, where e_{out} are all edges that initiate from host h with a particular attack-related state, and e_{in} are all edges whose target computer is h with the same state. We use $p(e_1 \cup e_2)=p(e_1)+p(e_2)-p(e_1 \cap e_2)$ and $p(e_1 \cap e_2)=p(e_1) \times p(e_2)$.

The probability of a single edge p(e) represents the overall importance of a piece of intrusion evidence, that is calculated by multiplying the edge attributes weight, relevancy and host importance. These three values can be obtained from NVD [14] as well as investigators' judgment on the particular attack.

Evidence graphs have two kinds of data, primary and secondary. Primary evidence is explicit and direct, and secondary evidence is implicit or circumstantial. We assign a primary evidence edge a higher value, and a secondary evidence edge a lower value. Additionally, sometimes, e.g. when attackers used anti-forensics tools or techniques to destroy evidence, expert knowledge is used to add an evidence edge, to which we attach a smaller probability than secondary evidence, resulting in our choice of 0.8, 0.6 and 0.5 that are for the three weights. p(h) in Definition 3 is the probability of all evidence whose source or target host is h, since investigators may find several pieces of evidence between two host computers.

Figure 3 is a probabilistic evidence graph, where there are two pieces of evidence from "129.174.128.148" to workstation with probabilities $p(e_1)$ and $p(e_2)$ respectively. The evidence of an attack going from the workstation to the database server has a probability $p(e_4)$. e_4 is second evidence that is represented as a dotted line, which has a coefficient 0.8. With these values, for example, the probability assigned to the probability of the workstation's being a stepping stone is $p(h_2) = p(e_1) \cup p(e_2)+0.8 \times p(e_4)-(p(e_1) \cup p(e_2)) \times 0.8 \times p(e_4)$. $p(e_1) \cup p(e_2)$ is the conjunctive probability of two pieces of attack evidence from "129.174.128.148" to workstation.

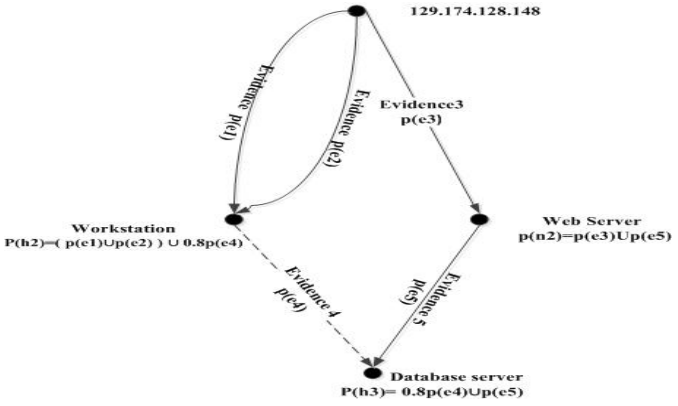


Figure 3: Example Probabilistic Evidence Graph

3. Building Graphs

As stated, we use MulVAL [15] to build our attack graph. We normalize all evidence using the five components (1) id, (2) source, (3) destination, (4) content and (5) time stamp in order to build our evidence graphs. Time order is taken as evidence (primary or secondary) dependencies to connect all hosts as nodes and evidence as edges to create an evidence graph. IDS alerts, suspicious activity log information closer to attack time and any information that directly reveals attack activities are considered as primary evidence. Secondary evidence may include various general-purpose sensor data [1], which include false positives triggered by benign activities and irrelevant attacks that are not part of the foreground attack scene of interest, or the tainted data from memory or hard disk images. Because secondary evidence is not as explicit and tangible as primary evidence, it is used only when primary evidence is insufficient to prove an attack activity.

Hypothesis testing is used in evidence graph construction. Sometimes, IDS may not be triggered by attacks, perhaps due to either not setting up the IDS properly or the attack activities do not seem malicious. In addition, the evidence might be so tainted that the investigators may make a wrong diagnosis. Also, attackers may actively destroy all evidence for the anti-forensic purpose. As a solution, [11] proposes to use a valid attack graph with a combination of expert knowledge database and anti-forensic database to implement the evidence graph. However, the hypothesis testing based on this must not contradict existing evidence that represents the attack truth, unless investigators are certain the existing evidence left by attackers is for obfuscation purposes [2].

IV. Mapping Evidence Graphs to Attack Graphs

We propose to map attack paths in an evidence graph to their corresponding paths in the attack graph. Both attack graphs and evidence graphs are directed acyclic graphs with the final victim hosts as sink nodes [16]. We reverse the edge direction in both graphs and take one of the final victim hosts as a source node to do the mapping (for simplicity, in the following mapping algorithm, the evidence graph and attack graph refer to direction-reversed graph and attack graph respectively). Because both graphs reflect the same network configuration and the evidence graph generally has fewer hosts including victim hosts, the evidence graph has a simpler structure than the attack graph (ideally, the evidence graph is a subset of the attack graph). As such, we take this final victim host in the evidence graph as its source host, from which we map the evidence graph to the attack graph. In the mapping, Breath First Search (BFS) [16] is iteratively used to find adjacent nodes, and the unique host ID combining evidence or vulnerability information is used for the node mapping between two graphs. During the process, exploitation information from the attack graph is compared against evidence from the evidence graph, and the probability values on the corresponding edge and nodes in both graphs are compared.

In the mapping algorithm we color nodes to mark if a node has been discovered on either side. All nodes are initially colored white. A node is colored gray when it is discovered but its children have not been fully examined, then black after all children are examined. After mapping, the edges between all black nodes construct attack paths in the attack graph.

Algorithm 1: Mapping evidence graph to attack graph
Input: Direction-reversed evidence graph E= (N, E, N-Attr, E-Attr, L, T) with N_{Victim} as the node with victim state;
Direction-reversed attack graph: A=(N_r,N_p, N_d, E,L,G,P), where N_p includes vulnerability information, N_d indicates the particular state on a host after a vulnerability is exploited
Output: Marked attack path in both E and A

Algorithm:

```
1.  Qe←Ø ; Qa ←Ø
2.  For each node u∈V[E] − {NVictim} and u ∈V[A]
3.     Do color[u] ← WHITE
4.          π[u] ← NIL
5.  color[NVictim] ← GRAY
6.  ENQUEUE (Qe, NVictim)
7.  While(Qe ≠Ø)
8.     Do u ← DEQUEUE(Qe)
9.        For each v ∈ Adj[u] in E= (N, E, N-Attr, E-Attr, L, T)
10.           Do Evidence[]←Ø
11.              If color[v] == WHITE
12.                 Then color[v] ← GRAY
13.                      ENQUEUE(Qe, v); π[v] ← u
14.                      Evidence[]. add(E-Attr)
15.                      Pe ← P(e);  Pp ← P(u); Pc ← P(v)
16.                      Mapping(u, v, evidence[],Pe, Pp,Pc)
17.       color[u] ← BLACK

Mapping  (u, v, evidence[],Pe,Pp, Pc )
18.  While(Qa==Ø)
19.     Qt←Ø ; Found ← false
20.     Do For each derived node d∈ V[A]
21.          If d.Id == u.id AND color[d] == white
22.             Then For each primitive fact v ∈ Adj[Adj[d]]
23.                     Do If (evidence[]. Contains(v))
24.                          Then color[d] ← BLACK
25.                               Found← true
26.                               Compare P(d) with Pp
27.                               ENQUEUE(Qp , d)
28.                               Break
29.                          Else  Color[d] ← GRAY
30.                               ENQUEUE(Qt ,d)
31.           Else  Add a node n with u.Id to Graph A
32.                 Color[n] ← black; New[n]  ← True
33.                  n.Vul ← Evidence[]
34.                  P[n] ←Pp; P[n.Vul] ← Pe; Qa← Ø
35.                  ENQUEUE(Qa,n)
36.                  Found← true
37.    If (Not Found)
38.         Then ENQUEUE(Qa, DEQUEUE(Qt))
39.  While (Qa≠Ø)
40.    Do  m← DEQUEUE(Qa)
41.      For each n ∈ Adj[Adj[m]]
42.         Do If n is a derived node
43.               Then Id =n.ID
44.            If n is a primary fact node
45.              Then Vul=n.Content
46.            If Id==v.Id AND evidence[].contains(Vul)
47.              Then  color[n] ← BLACK
48.                    Found ← true
49.                    π[n] ← p; Qa← Ø ; ENQUEUE(Qa,n)
50.                    Compare p(Vul) with Pe
51.                    Compare P(m) with Pp
52.                    Compare P(n) with Pc
53.                    break
54.             Else If Id==v.Id
55.                     Then color[n] ← GRAY; ENQUEUE(Qt , n)
56.     If (Qa==Ø) )
57.        Then  Add a node a with u.Id to Graph A
58.              Color[a] ← black; New[a]  ← True
59.               a.Vul ← Evidence[]; P[a.Vul] ← Pe
60.               π[a] ← p; Qa← Ø; ENQUEUE(Qa,a)
61.  Return
```

Line 1 initializes two empty queues. Q_e holds gray nodes for evidence graph, and Q_a holds the last mapped nodes in the attack graph. Lines 2-4 paint every host node in evidence graph and derived node in attack graph white, and set the parent of each of those nodes to be NIL. The reason why we use derived nodes in attack graph is that they correspond to host nodes in evidence graph. Line 5 paints the victim node in the attack graph gray, since we consider it to be discovered first when the procedure begins. Line 6 pushes the victim node to the queue because it has been discovered. The while loop in lines 7–16 iterates as long as there remain gray nodes in the queue Q_e. Those gray nodes are discovered nodes that have not yet had their adjacency lists fully examined. In this while loop, line 8 takes out the first gray node in the queue, and lines 9-12 examine all its white color neighbor nodes, and paint them as gray color. Because there may be many, the array evidence[] is initialized to hold evidence for a mapping later. Once an adjacent node v is found, line 13 pushes it to the queue and assigns its parent as u. The evidence (E-Attr) between u, v and probabilities of u, v and e (e as collection of all evidence E-Attr between u and v) are saved in lines 14-15, where P_e is the probability of the evidence e, P_p is the target host probability, and P_c is source host probability (p=parent, c=child). The mapping function is called in line 16 with the above saved parameters, which are discovered parent node u, child node v, evidence[] between u and v and their corresponding probabilities for conjunctive evidence and two nodes u, v. Lastly, when node u's all neighbor nodes have been examined, it is marked as black in Line 17, which will not be examined any more. If all connected nodes are marked black, the algorithm terminates, and an attack path could be constructed by linking those black nodes.

The mapping function traverses all nodes in the attack graph, trying to find the derived nodes that correspond to nodes u and v from the evidence graph and compare the corresponding evidence, vulnerability and probabilities information.

Lines 18--38 find the corresponding host in the attack graph that corresponds to victim node in the evidence graph. We do so by searching all derived nodes that has host ID information in the attack graph. The reason why we did not only focus on victim nodes in attack graph is that the attach path in attack graph might be longer than the corresponding path in the evidence graph. Line 18 checks if Q_a, the queue that holds last mapped derived node, is empty or not. If it is empty, we have not started any mapping, and we search for the first node that corresponds to source node in evidence graph. Lines 21--28 checks every derived node that has same ID as u --victim node from evidence graph and same vulnerability information as evidence attached to u (The vulnerability is primitive fact v∈ Adj[Adj[d]]). Once such a derived node is found, it is colored black and pushed into Q_a, which will be used as the parent node to search child derived nodes in next mapping steps. Once the searching on the child derived nodes have been successful, the black node will be de-queued (in line 40) so that new parent nodes will be en-queued to Q_a for an iterative search. Lines 29—30 use gray color to mark all derived nodes that have same ID as node u from evidence graph but have no matching vulnerability information as the evidence attached to u. These derived nodes are pushed into queue Q_t temporarily, which is

initialized in line 19. Once all searching finishes, if there are only gray color nodes (i.e. no matching black node is found), they will be de-queued from Q_t and en-queued into Q_a as parents for next step's search. This is on lines 37—38 (Because of multi-vulnerability in a particular host, there might be several derived nodes with same host ID in attack graph). We call these gray nodes half matched because the host ID is matched to node in evidence graph but there is no corresponding matched vulnerability to the evidence. In this case, we use all gray nodes with the same host ID for the next step in the search. Consequently, the attack paths in the attack graph may include gray nodes and black nodes. Lines 31—36 add a new node to the evidence graph if there is no matched or half match node. This happens when the attack graph does not include a host computer with vulnerability exploitable for this attack. For later attack graph refining purpose, corresponding evidence and probabilities from the evidence graph are added to the newly added node in the attack graph.

Lines 39—60 are similar to line 18—38, which seek a derived node n in the attack graph that corresponds to u's child node v in the evidence graph. The only difference between 39—60 and 18—38 is that, before searching for the child derived node n in 39—60, the algorithm have found n's mapped parent/ancestor node in the attack graph, which has been saved in Q_a(Line 39).

V. CASE STUDY

A. Experiment Network

We implemented a small-scale multi-state attack in an experimental network as shown in Figure 4 to construct two graphs and experiment with the mapping. The external firewall controls network access from the Internet to the enterprise network, where the Apache Tomcat webserver hosts a webpage, which allows Internet users' visit through port 8080. The internal firewall controls the access to the MySQL database server, where the databases can be accessed by the webserver and workstations through a default port 3306.

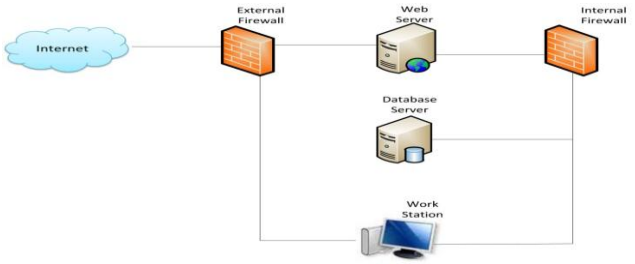


Figure 4: Experiment Network

In this network, the webserver is configured to record all incoming IP addresses with timestamps. The database server is configured to record all query information, and the IDS is used to catch network traffic from the Internet.

Our attack objective is to gain access to database tables as an Internet user. Our attack plan is to launch a SQL injection attack that exploits a java servlet code that does not sanitize input values: *theStatement.executeQuery("select * from profiles where name='Alice' AND password='"+passWord+"'");*. This exploit corresponds to CWE89 in NVD [12]. The workstation that runs Windows XP SP3 operating system with IE6 with the vulnerability (*CVE-2009-1918* [9]) enables executing any code on this machine. Our external attacker uses social engineering to trick the workstation user to click on a malicious web link, which enables the attack machine to control the workstation that has direct access to the database.

B. The Attack Graph

We use MulVAL [15] to generate the attack graph in Figure 5. The 25 steps of the attack scenario are explained in Appendix 2.

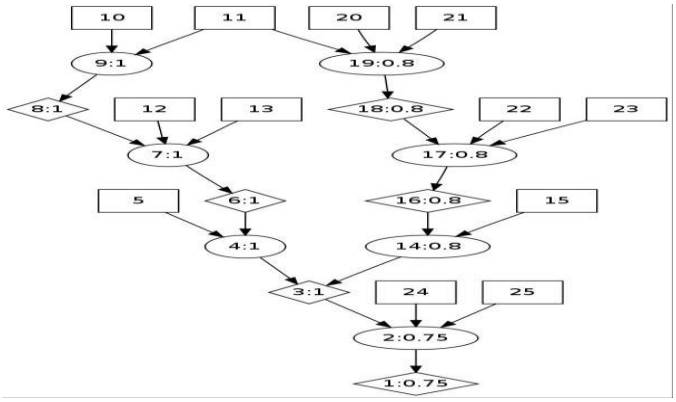


Figure 5: Experiment Probabilistic Attack graph

Table 1 holds all probabilities of single exploits in the network obtained from CVSS [5] and human factors in a network. Number 2 is a human factor, which varies from user to user. We assign it 0.8, because it is easy to trick an employee to click on a well-disguised link [4]. Using Table 1 and Definition 2, we calculated the probabilities of exploits that are in derivation nodes and the cumulative probabilities of attack success in derived nodes. These values are shown in Figure 5.

TABLE 1: EXAMPLE NETWORK VULNERABILITY/ACCESS PROBABILITY

Number	Configuration/Vulnerability	Probability
1	Direct Network Access	1
2	Social Engineering	0.8
3	CVE-2009-1918	1
4	CWE89	0.75
5	Access to database from workstation	1

C. Evidence Graph

TABLE 2:ROLE CONFIGURATION IN THE ATTACKS

Attacker	129.174.128.148
Step Stone	Workstation
Step Stone/Affiliated	Web server
Victim	Database server

Our forensic analysis uses the attacker roles stated in Table 2, corresponding to the following scenario.

(1) An un-sanitized string was entered into the password text input field of the web server Webpage, which is " anything' or '1'='1 ". This caused a SQL injection attack to a database named "profiles" in database server.

(2) Internet Explorer "Aurora" Memory Corruption was launched from attacker machine to the Windows workstation with IE6 browser. This was done by tricking a windows

workstation user to click on a link sent from the attacker machine.

(3) After the workstation has been hijacked, a tool like TightVNC [8] is uploaded to the workstation, which could remotely control the workstation to access the database.

Evidence was divided into two categories. Because the suspicious SQL injection query "*select * from profiles where name='Alice' AND password='alice' or '1' = '1'* "has obvious injection feature that is ***'1'='1'***[23], we categorized it to primary evidence. By using its time stamp, "*120416 15:32:51*", we investigated other machines connected to this database, which are webserver and workstations. According to the above time stamp, suspicious log information was found on the webserver, which was "*129.174.92.32 - - [16/Apr/2012:15:32:51 -0400] "POST /lab/Test HTTP/1.1" 200 980*"(129.174.92.32 is workstation's IP). Obviously, the IP address "129.174.92.32" that does not belong to this network is the attacker. Now, an attack path can be constructed.

Because of the small size of our experiment, it was easy to find evidence in the workstation by investigating browser history and "local setting/temp" folder to search for suspicious links or executable file (The executable file is TightVNC [8] in our case) sent or uploaded by the attacker. However, in a real scenario, there must be many workstations connected to a database server. Under this situation, an expert knowledge database such as the one in [11] or investigators' empirical experiment should be used to reduce the investigation scope.

Because the compromised workstation has direct access to the database server, it is difficult to ascertain if the access record is that of an attacker. Consequently, this attack step from the workstation to the database server was added by using expert knowledge, which completed another attack path "*attacker → workstation →**database server***" in the evidence graph.

We normalized the above evidence into a form "(1) id, (2) source, (3) destination, (4) content (5) time stamp" and used a Java program to reason the evidence dependency in a time order, which was converted into an evidence graph in Figure 6 by using Graphviz[6]. The solid edge represents primary evidence. The dotted line from "workstation" to "dbServer" was added by using expert knowledge.

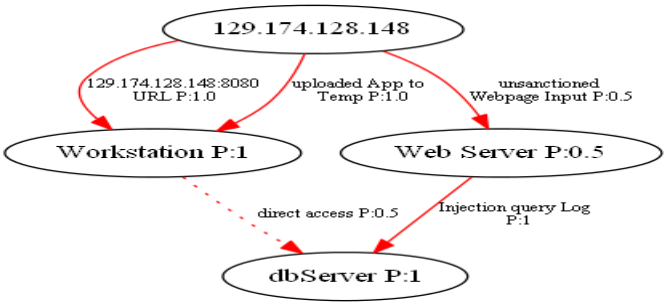


Figure 6: Experiment Network Evidence Graph

D. Mapping Evidence Graph to Attack Graph

We mapped our evidence graph to attack graph using Algorithm 1. Our result shows that the attack paths in the attack graph are almost the same as those in evidence graph, except that the attack graph shows that the attack from the compromised workstation to the database server uses SQL injection (*Node 25*), while evidence graph shows a direct access. According to this information, we re-checked the attack graph and found that node 25 should be attached to node 4 instead of node 2. Having fixed this by changing Prolog rules, the attack graph was changed to have the same attack paths as evidence graph in Figure 6, but with changed probabilities. Now, the probability of attack from the webserver to database server is 0.75 using the SQL injection vulnerability, and the probability of an attack from workstation to the database server is 0.8, because the prior social Engineering attack success is 0.8. The cumulative probability on the database server is 0.8+0.75-0.8×0.75 = 0.95, which is closer to the real world scenario than the one before we made the corresponding change.

By reading mapped graphs, the corresponding forensics or network defense tools should be taken in the network so that they can serve a better forensics analysis. In our network, two places should be enhanced. They are webserver and the access from workstations to database server, because the analysis showed that they provided the attack paths to the database, which was not configured for forensics analysis.

We also did an anti-forensics experiment on this network, where the workstation was fully compromised using root privileges that allowed us to totally remove the evidence that indicates workstation's being compromised by our attacker. With anti-forensics tools, the attack path "*attacker → workstation →database*" would be missing in the evidence graph. In this case, the attack graph implemented with anti-forensics activity nodes, which we proposed in [11], has to be used to recover the attack scenario, helping in reconstructing the complete evidence graph.

VI. CONCLUSION

Based on the formal definition of probabilistic attack graphs and evidence graphs, we showed how to map the evidence graph to the attack graph. This mapping helps in adjusting the attack graph and find what is missing in the evidence graph, which would assist in forensic investigations and reconfigure the network to defeat attacks. Furthermore, by using this mapping algorithm, investigators could find out if attackers have used anti-forensics. Therefore, corresponding measures, such as the method proposed in [11], would be used to recover an attack scenario in order to have a complete forensic analysis.

REFERENCES

[1]. W. Wang and T. Daniels, "Building evidence graphs for network forensics analysis", Proceedings of the Twenty-First Annual Computer Security Applications Conference, pp. 254–266, 2005.

[2]. B. D. Carrier and E. H. Spaord, "An Event-Based Digital Forensic Investigation Framework", In Proceedings of the 4th Digital Forensic Research Workshop, 2004.

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

[5] CVSS--A Complete Guide to the Common Vulnerability Scoring System Version 2.0, http://www.first.org/cvss/cvss-guide

[6] Graphviz-Graph Visualization Software, http://www.graphviz.org/

[7] P. Ammann, D. Wijesekera, S. Kaushik. "Scalable, graph-based network vulnerability analysis", In Proceedings of 9th ACM Conference on Computer and Communications Security, Washington, DC, November 2002

[8] TightVNC Software,

http://www.tightvnc.com/.

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

[10] 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.

[11] C. Liu, A. Singhal, D. Wijesekera, "Using Attack Graphs in Forensic Examinations", 2012 Seventh International Conference on Availability, Reliability and Security, August 2012

[12] http://nvd.nist.gov/cwe.cfm

[13] N. Liao, S. Tian, T. Wang, "Network forensics based on fuzzy logic and expert system", Computer Communication, vol. 32, no. 17, pp. 1881–1892, 2009.

[14] National Vulnerability Database, http://nvd.nist.gov/.

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

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

[17] 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, pages 254–265, 2002.

[18] K. Ingols, M. Chu, R. Lippmann, S. Webster and S. Boyer. "Modeling Modern Network Attacks and Countermeasures Using Attack Graphs", Proceedings of ACSAC Conference 2009.

[19] SANS Institute InfoSec Reading Room, "an Overview of Disk Imaging Tool in Computer Forensics", 2001.

[20] B. Carrier, "File System Forensic Analysis", Addison-Wesley Professional, March 2005.

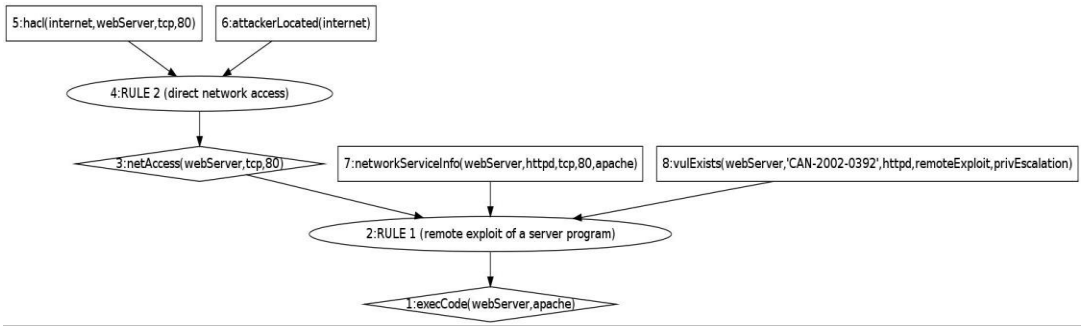
[21] H. Debar, M. Becker, D. Siboni, A neural network component for an intrusion detection system, Proceedings of IEEE Symposium on Research in Computer Security and Privacy, 1992.

[22] S. Jajodia, S. Noel, B.O.'Berry. "Topological Analysis of Network Attack Vulnerability", In Managing Cyber Threats: Issues, Approaches and Challenges, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Springer, 2005.

[23] W. G. Halfond, J. Viegas, and A. Orso. A Classification of SQL-Injection Attacks and Countermeasures. In Proc. Of the Intern. Symposium on Secure Software Engineering (ISSSE 2006), Mar. 2006.

APPENDIX

1. Attack graph in Figure 1



2. Predicates used in Figure 3

1	execCode(dbServer,user)
2	RULE 2 (remote exploit of a server program)
3	netAccess(dbServer,tcp,3306)
4	RULE 5 (multi-hop access)
5	hacl(webServer,dbServer,tcp,3306)
6	execCode(webServer,apache)
7	RULE 2 (remote exploit of a server program)
8	netAccess(webServer,tcp,8080)
9	RULE 6 (direct network access)
10	hacl(internet,webServer,tcp,8080)
11	attackerLocated(internet)
12	networkServiceInfo(webServer,httpd,tcp,8080,apache)
13	vulExists(webServer,'CWE89',httpd,remoteExploit,privEscalation)
14	RULE 5 (multi-hop access)
15	hacl(workStation,dbServer,tcp,3306)
16	execCode(workStation,user)
17	RULE 3 (remote exploit for a client program)
18	accessMaliciousInput(workStation,secretary,'IE')
19	RULE 22 (Browsing a malicious website)
20	hacl(workStation,internet,httpProtocol,httpPort)
21	inCompetent(secretary)
22	hasAccount(secretary,workStation,user)
23	vulExists(workStation,'CVE-2009-1918','IE',remoteClient,privEscalation)
24	networkServiceInfo(dbServer,mySQL,tcp,3306,user)
25	vulExists(dbServer,'SQLinjection',mySQL,remoteExploit,privEscalation)