A LOGIC-BASED NETWORK FORENSIC MODEL FOR EVIDENCE ANALYSIS

Changwei Liu, Anoop Singhal, Duminda Wijesekera

Chapter 8

A LOGIC-BASED NETWORK FORENSIC MODEL FOR EVIDENCE ANALYSIS

Changwei Liu, Anoop Singhal and Duminda Wijesekera

Abstract Many attackers tend to use sophisticated multi-stage and/or multi-host attack techniques and anti-forensic tools to cover their traces. Due to the limitations of current intrusion detection and network forensic analysis tools, reconstructing attack scenarios from evidence left behind by attackers of enterprise systems is challenging. In particular, reconstructing attack scenarios using intrusion detection system alerts and system logs that have too many false positives is a big challenge.

This chapter presents a model and an accompanying software tool that systematically addresses the reconstruction of attack scenarios in a manner that could stand up in court. The problems faced in such reconstructions include large amounts of data (including irrelevant data), missing evidence and evidence corrupted or destroyed by anti-forensic techniques. The model addresses these problems using various methods, including mapping evidence to system vulnerabilities, inductive reasoning and abductive reasoning, to reconstruct attack scenarios. The Prolog-based system employs known vulnerability databases and an anti-forensic database that will eventually be extended to a standardized database like the NIST National Vulnerability Database. The system, which is designed for network forensic analysis, reduces the time and effort required to reach definite conclusions about how network attacks occurred.

Keywords: Network forensics, network attacks, evidence graph, admissibility

1. Introduction

Network forensics is the science that deals with the capture, recording and analysis of network events and traffic in order to detect and investigate intrusions. A network forensic investigation requires the construction of an attack scenario when conducting an examination of the

attacked system. In order to present the scenario that is best supported by evidence, forensic investigators must analyze all possible attack scenarios reconstructed from the available evidence. This includes false negatives and items of evidence that are missing or destroyed (in part or in entirety) as a result of investigators using tools that are unable to capture traces of some attacks or attackers using anti-forensic techniques to destroy evidence.

Although the use of intrusion detection system alerts and system logs as evidence has been contested in courts, they provide the first level of information to forensic investigators when creating potential attack scenarios [13]. In order to reconstruct potential attack scenarios, researchers [1, 2] have proposed aggregating redundant alerts based on similarities and correlating them using predefined attack scenarios to determine multi-step, multi-stage attacks. However, this approach is manual and rather *ad hoc*. As an improvement, Wang and Daniels [15] proposed automating the process using a fuzzy-rule-based hierarchical reasoning framework that correlates alerts using local rules and grouping them using global rules. However, this approach fails when evidence is destroyed and it does not consider the potential admissibility of the evidence and the constructed attack scenario in legal proceedings.

To address these problems, this research employs a rule-based system that automates the attack scenario reconstruction process while being cognizant of standards for evidence admissibility. The rule base incorporates: (i) correlation rules that coalesce security event alerts and system logs; (ii) rules that explain missing and destroyed evidence (with the support of an anti-forensic database) by using what-if scenarios; and (iii) rules that help assess the admissibility of evidence. The viability and utility of the system is demonstrated via a prototype written in Prolog.

This research builds on a preliminary reasoning model described in [8] by extending the MulVAL attack graph generation tool [12] that is implemented in XSB Prolog [14]. Figure 1 presents the architecture of MulVAL and the extended model. The extensions, which are shaded, include: (i) an evidence module that uses MITRE's OVAL database [10] or expert knowledge (if there is no corresponding entry in the OVAL database) to convert evidence from the attacked network to the corresponding software vulnerability and computer configuration required by MulVAL; (ii) anti-forensic and expert knowledge databases that generate explanations in the face of missing or destroyed evidence; and (iii) rules of evidence and access control modules that help judge the acceptability of evidence.

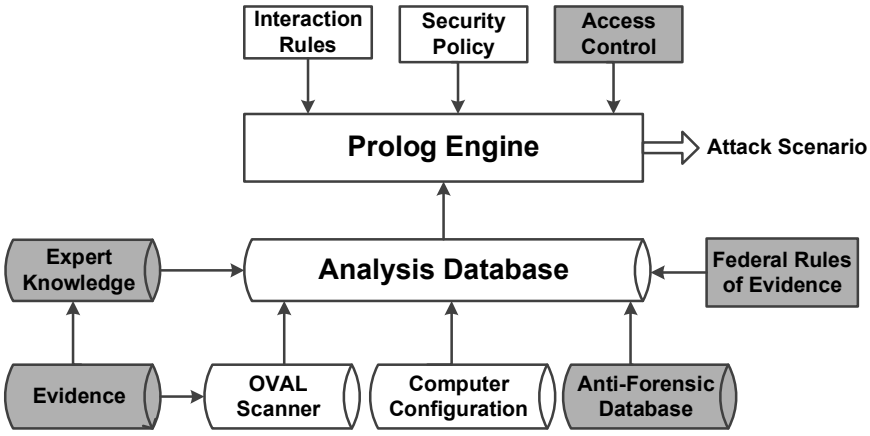


Figure 1. Architecture of MulVAL and the extended model.

2. Background and Related Work

This section presents the background concepts and related research.

2.1 MulVAL and Logical Attack Graphs

MulVAL is a Prolog-based system that automatically identifies security vulnerabilities in enterprise networks [7]. Running on XSB Prolog, MulVAL uses tuples to represent vulnerability information, network topology and computer configurations and to determine if they give rise to potential attack traces. The graph comprising all the attack traces generated by this system is called a logical attack graph.

Definition 1: 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 [7, 11].

Figure 2 shows an example logical attack graph. A primitive fact node (rectangle) represents specific network configuration or vulnerability information corresponding to a host computer. 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.

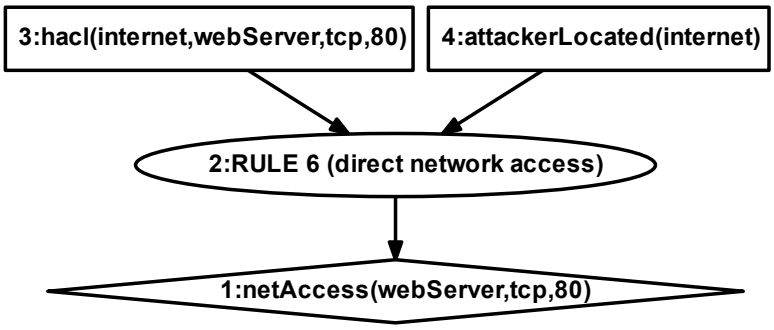


Figure 2. Example logical attack graph.

2.2 Evidence Graphs

While an attack graph predicts potential attacks, an evidence graph is constructed to present evidence of a specific enterprise network attack.

Definition 2: An evidence graph is a sextuple G = (N_h, N_e, E, L, N_h-Attr, N_e-Attr) where N_h and N_e are sets of disjoint nodes representing a host computer involved in an attack and the related evidence, E $\subseteq$ $(N_h \times N_e) \cup (N_e \times N_h)$, L is a mapping from a node to its label, and N_h-Attr and N_e-Attr are attributes of a host node and evidence node, respectively [5].

The attributes of a host node include host ID, states and timestamps. The states include the states before and after a particular attack step, which can be source, target, stepping-stone or affiliated [5, 7]. The attributes of an evidence node describe the event initiator, target and its timestamp.

2.3 Related Work

Reasoning has been used to correlate evidence when reconstructing crime scenarios. In traditional (non-digital) forensics, researchers have used inductive and abductive reasoning to model potential crime scenarios and correlate evidence [4]. In the area of digital forensics, Wang and Daniels [15] have used a fuzzy-rule base to correlate attack steps substantiated by aggregated security event alerts. Their scheme aggregates security event alerts by checking if they have the same source-destination pair, belong to the same attack class and fall within a self-extending time window. A self-extending time window is elongated to include all alerts within a predefined time difference from the original event. However, the approach does not handle situations where evidence is missing or

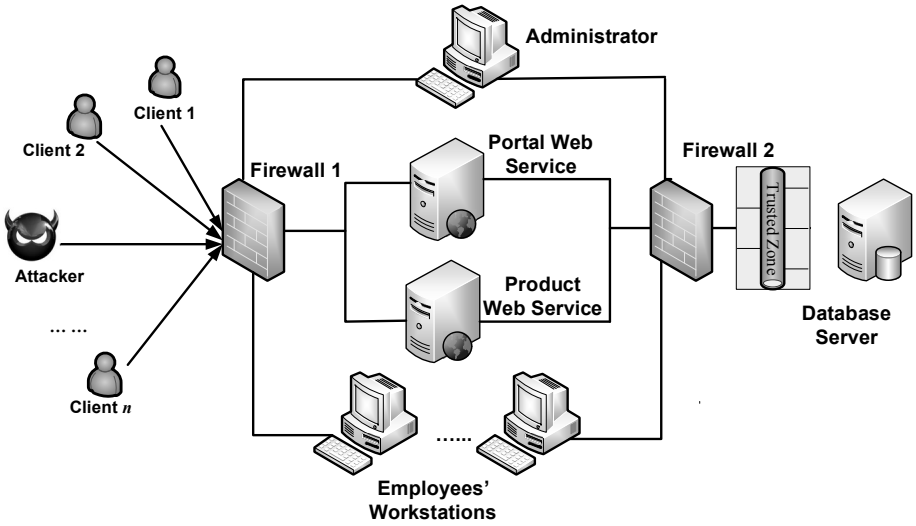


Figure 3. Example attacked network.

incomplete nor does it use any standards to determine the acceptability of evidence.

In order to address these limitations, Liu et al. [6] have proposed the use of an anti-forensic database containing expert knowledge to help generate hypotheses about missing or destroyed evidence and to substantiate the default assumptions of experts. MITRE's OVAL database and federal rules related to digital evidence are used to determine the potential acceptability of digital evidence [9]. Some of the solutions proposed in [9] were not implemented; these are discussed later in this chapter.

3. Network Example

Figure 3 shows an example attacked network [9]. In order to explain how an anti-forensic database can be used to create explanations in the face of destroyed evidence, anti-forensic techniques were used to remove some evidence.

Table 1 shows the machine IP address and vulnerability information. By exploiting the vulnerabilities listed in Table 1, the attacker was able to launch three attacks: (i) compromise a workstation (CVE-2009-1918) to access the database server; (ii) leverage the product web application vulnerability (SWE89) to attack the database server; and (iii) exploit a cross-site scripting (XSS) vulnerability on a chat forum hosted by the portal web service to steal the administrator's session ID so that phishing

Table 1. Machine IP addresses and vulnerabilities.

Machine	IP Address/Port	Vulnerability
Attacker	129.174.124.122	
Workstations	129.174.124.184/185/186	HTML Objects Memory Corruption Vulnerability (CVE-2009-1918)
Webserver1 – Product Web Service	129.174.124.53:8080	SQL Injection (CWE89)
Webserver2 – Product Web Service	129.174.124.53:80	SQL Injection (CWE89)
Administrator	129.174.124.137	Cross-Site Scripting Flaw (XSS)
Database Server	129.174.124.35	

emails could be sent to clients, tricking them to update their confidential information.

In the experimental network, the installed intrusion detection system, configured web server and database server were able to detect some attacks and log malicious accesses. However, there were some false positives (e.g., when attack attempts were not successful). Also, because the Snort tool used for intrusion detection did not incorporate certain rules or the actual attack activities looked benign, some attacks were not detected and, therefore, no alerts were logged as evidence in these instances. Two examples are: (i) phishing URLs sent by the attacker to the clients requesting them to update their confidential information were not detected; and (ii) database server access by the attacker from the compromised workstation was deemed to be benign. In addition, the attacker compromised the workstation and obtained root privileges, which enabled him to use anti-forensic techniques to delete evidence left on the workstation. Under these conditions, when evidence is missing or destroyed, it is necessary to find a way to show how the attack might have occurred.

4. Attack Scenario Reconstruction

This section discusses the attack scenario reconstruction process.

4.1 Rules and Facts

As stated in Section 1, the vulnerability/forensics database constructed from MITRE's OVAL [10] was used to convert intrusion detection system alerts and the associated system logs to the corresponding vulnerability entries for attack scenario reconstruction (expert knowledge

Table 2. Evidence of alerts and logs from Figure 3.

Timestamp	**Source IP Address**	**Destination IP Address**	**Content**	**Vulnerability**
08\13-12:26:10	129.174.124.122:4444	129.174.124.184:4040	SHELLCODE x86 inc ebx NOOP	CVE-2009-1918
08\13-12:27:37	129.174.124.122:4444	129.174.124.184:4040	SHELLCODE x86 inc ebx NOOP	CVE-2009-1918
08\13-14:37:27	129.174.124.122:1715	129.174.124.53:80	SQL Injection Attempt	CWE89
08\13-16:19:56	129.174.124.122:49381	129.174.124.137:8080	Cross-Site Scripting	XSS
08\13-14:37:29	129.174.124.53	129.174.124.35	name='Alice' AND password='alice' OR '1'='1'	CWE89
...	...	...	...	...

is used only when an entry is not found in OVAL) [8]. Table 2 shows the converted evidence from the experimental network in Figure 3.

Figure 4 shows the concrete predicates for the items of evidence corresponding to the computer configuration and network topology, which instantiate the corresponding predicates in reasoning rules during a MulVAL run. In the figure, predicate "attackedHost" represents a destination victim computer; predicate "hacl" denotes a host access control list; predicate "advances" represents the access rights within the firewall that were used by the attacker to reach the next computer after the attacker had compromised a computer as a stepping-stone; predicate "timeOrder" ensures that the starting time and the ending time of an attack are within a reasonable interval; and predicates "vulExists," "vulProperty" and "networkServiceInfo" represent an attack step on the target computer (the first term in the predicate "networkServiceInfo" represents the target computer).

Figure 5 shows a reasoning rule that describes a generic attack. Each rule has Prolog tuples that derive the postconditions from the preconditions of an attack step. For example, the rule in Figure 5 stipulates that: if the attacker has compromised the victim's computer (Line 3) and the victim has the privilege "Perm" on his computer "Host" (Line 4) and the attacker can access the victim's computer (Line 5), then the evidence representing the three preconditions as the cause is correlated with the evidence that the attacker obtained the victim's privileges on the victim's computer (Line 2). Line 1 is a string that uniquely identifies a rule and Line 6 is the description of the rule.

```
/* Final attack victims */
attackedHost(execCode(admin,_)).
attackedHost(execCode(dbServer,_,_)).
attackedHost(execCode(admin,_)).

/* Network topology and access control policy */
attackerLocated(internet).
hacl(internet,webServer,tcp,80).
hacl(webServer,dbServer,tcp,3660).
hacl(workStation1,dbServer,tcp,3660).
hacl(workStation2,dbServer,tcp,3660).
hacl(internet,workStation1,_,_).
hacl(internet,workStation2,_,_).
hacl(internet,admin,_,_).
hacl(H,H,_,_).
advances(webServer,dbServer).
advances(workStation,dbServer).

/* Timestamps used to find the evidence dependency */
timeOrder(webServer,dbServer,14.3727,14.3729).
timeOrder(workStation1,dbServer,12.2610,14.3730).

/* Configuration and attack information of workStation1 */
vulExists(workStation1,'CVE-2009-1918',httpd).
vulProperty('CVE-2009-1918',remoteExploit,
privEscalation).
networkServiceInfo(workStation1,httpd,tcp,80,apache).
...
```

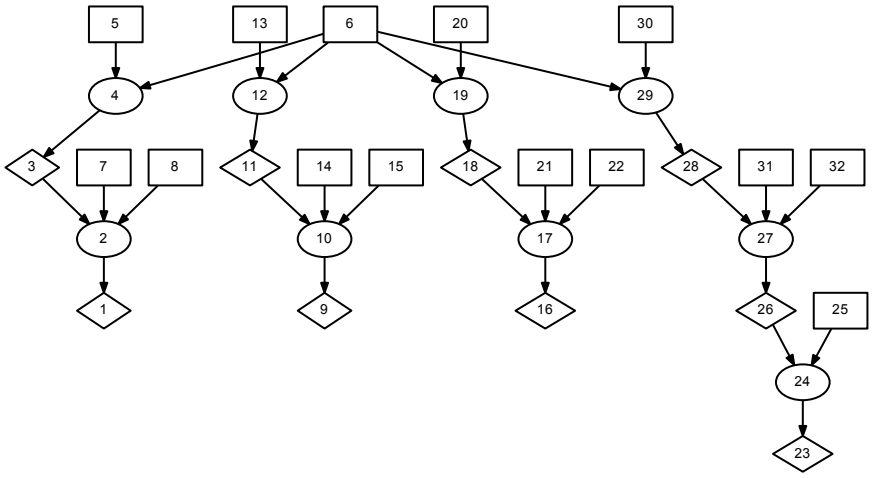
Figure 4. Input facts in the form of predicates representing evidence.

```
/* Interaction Rules */
1. interaction_rule(
2. (execCode(Host,Perm) :-
3.     principalCompromised(Victim),
4.     hasAccount(Victim,Host,Perm),
5.     canAccessHost(Host)),
6.     rule_desc('When a principal is compromised, any machine
       on which he has an account is also compromised',0.5)).
```

Figure 5. Example reasoning rule.

4.2 Evidence Graph Generation

Figure 6 shows the attack scenario that resulted from querying the logic-based system that was not integrated with an anti-forensic database and did not cover evidence acceptability standards. The table below the evidence graph provides a description of each node and the corre-



1:execCode(admin,apache)	2:RULE 2 (remote exploit of a server program)
3:netAccess(admin,tcp,80)	4:RULE 7 (direct network access)
5:hacl(internet,admin,tcp)	6:attackerLocated(internet)
7:networkServiceInfo(admin,httpd, tcp,80,apache)	8:vulExists(admin,'XSS',httpd, remoteExploit,privEscalation)
9:execCode(workStation1,apache)	10:RULE 2 (remote exploit of a server program)
11:netAccess(workStation1,tcp,80)	12:RULE 7 (direct network access)
13:hacl(internet,workStation1, tcp,80)	14:networkServiceInfo(workStation1, httpd,tcp,80,apache)
15:vulExists(workStation1, 'CVE-2009-1918',httpd, remoteExploit,privEscalation)	16:execCode(workStation2,apache)
17:RULE 2 (remote exploit of a server program)	18:netAccess(workStation2,tcp,80)
19:RULE 7 (direct network access)	20:hacl(internet,workStation2,tcp,80)
21:networkServiceInfo(workStation2, httpd,tcp,80,apache)	22:vulExists(workStation2, 'CVE-2009-1918',httpd, remoteExploit,privEscalation)
23:netAccess(dbServer,tcp,3660)	24: RULE 6 (multi-hop access)
25:hacl(webServer,dbServer,tcp,3660)	26:execCode(webServer,apache)
27:RULE 2 (remote exploit of a server program)	28:netAccess(webServer,tcp,80)
29:RULE 7 (direct network access)	30:hacl(internet,webServer,tcp,80)
31:networkServiceInfo(webServer, httpd,tcp,80,apache)	32:vulExists(webServer,'CWE89', httpd,remoteExploit,privEscalation)

Figure 6. Network attack scenario reconstructed from alerts and logs.

```
1. query:-
2.     tell('queryresult.P'),
3.     writeln('execCode(dbServer,user):'),
4.     listing(execCode(dbServer,user)),
5.     told.
```

Figure 7. Querying the system for explanatory hypotheses.

sponding evidence in the logical form according to Definition 1. The facts, including primary facts (rectangles) and derived facts (diamonds) before a derivation node (ellipse), represent the preconditions before an attack step. All the facts after a derivation node represent postconditions after the attack step. Figure 6 shows the four attack paths constructed by this process: (i) the attacker used a cross-site scripting attack (XSS) to steal the administrator's session ID and obtain the administrator's privileges (6→ 4 → 3 → 2 → 1); (ii) the attacker used a web application that does not sanitize user input (CWE89) to launch a SQL injection attack on the database (6 → 29 → 28 → 27 → 26 → 24 → 23); (iii) the attacker used a buffer overflow vulnerability (CVE-2009-1918) to compromise workstations (6→ 12 → 11 → 10 → 9 and 6 → 19 → 18 → 17 → 16).

Because some evidence is missing and destroyed, the phishing attack observed on the client computers and the attack on the database server launched from the compromised workstations could not be constructed or shown in Figure 6. Also, because the available evidence used for attack scenario reconstruction was not validated using standards of acceptability, Figure 6 might not reflect the real attack scenario and, hence, would have little weight in a court of law.

5. Extending MulVAL

This section describes the extension of MulVAL to support attack scenario reconstruction.

5.1 Using an Anti-Forensic Database

Abductive reasoning and an anti-forensic database are engaged by the Prolog-based framework to explain how a host computer might have been attacked when evidence is missing or destroyed [6, 8].

By using abductive reasoning, the extended Prolog logic-based framework is able to provide all potential general explanations about how an attacker might have launched a particular attack. For example, in order to explain how the database server in Figure 3 might have been attacked, the file shown in Figure 7 was created to query the extended

Table 3. Anti-forensic database.

ID	**A1**	**D1**	...
Category	attack tool	destroy data	...
Tool		BC-Wipe	...
Technique	obfuscate signature	delete content	...
Windows	all	98+	...
Linux	all	all	...
Privilege	user	user	...
Access	remote client	local client	...
Software	Snort		...
Effect	bypass detection	delete data permamently	...

logic-based system to show all the attack steps that would result in "execCode(dbServer,user)" (attack on the database server) as the explanatory hypothesis. In Figure 7, Lines 2 to 5 require the system to list all the attack steps of "execCode(dbServer,user)" and write the results to the file `queryresult.P` (Line 2 opens the output stream for writing to `queryresult.P` and Line 5 closes the output stream). The returned query results indicate that three possible hypotheses could result in "execCode(dbServer,user)." They are: (i) using a compromised computer as the stepping stone; (ii) exploiting the vulnerability of the database access software; and (iii) using a legitimate database server account to inject malicious input.

After all possible hypotheses have been generated, the extended Prolog logic-based framework uses an anti-forensic database to select the best explanation. Table 3 shows an example anti-forensic database from [6], which instantiates the predicate "anti_ Forensics(Category,Tool, Technique, Windows,Linux,Privilege,Access,Software,Consequence)" in Line 1 of Figure 8 so that it can work with the hypotheses obtained from the corresponding abductive rules to evaluate if a particular hypothesis could result in the postconditions collected from an attacked computer when evidence is missing or has been destroyed.

Figure 8 is a codification of an anti-forensic database for the purpose of providing explanations in the face of missing or destroyed evidence. Lines 2 through 11 specify two rules that use the predicate "anti_Forensics(Category,Tool,Technique,Windows,Linux,Privilege,Access,Software,Consequence)" to evaluate if the hypothesis that the attacker has used the vulnerability in the database access software to attack the database is the best explanation.

In the first rule (Lines 2–7), the head "vulHyp(H,_vulID,Software, Range,Consequence)" (the hypothetical vulnerability the attacker might

```
1. anti_Forensics(Category,Tool,Technique,Windows,Linux,Privilege,
Access,Program,Consequence).

2. vulHyp(H,_vulID,Software,Range,Consequence) :-
/* The following three predicates are from the abductive reasoning result */
3. vulExists(Host,_vulID,Software,Access,Consequence),
4. networkServiceInfo(Host,Software,Protocol,Port,Perm),
5. netAccess(Host,Protocol,Port).
/* Introduce a hypothetical vulnerability */
6. anti_Forensics(Category,Tool,Technique,OS,Privilege,Access,Software,Effect).
7. hostConfigure(Host,OS,Software).

8. with_hypothesis(vulHyp, Post-Condidtion) :-
9. cleanState,
10. assert(vulHyp(H,_vulID,Software,Range,Consequence)),
11. post-Condition.
```

Figure 8. Codifying the anti-forensic database.

have used) is derived from three predicates: (i) hypothesis obtained from one of the results of Figure 7 (Lines 3–5); (ii) "anti-Forensics" predicate in Line 6 where the variable terms ("Category," "Tool," "Technique," "Windows," "Linux," "Privilege," "Access," "Program" and "Consequence") are instantiated by the corresponding concrete data from Table 3 during a system run; and (iii) configuration of the host (Line 7) where the evidence has been destroyed by an anti-forensic technique used by the attacker.

In the second rule (Lines 8–11), the derived fact "vulHyp(H,_vulID, Software,Range,Consequence)" obtained from the first rule is asserted to the logic runtime database (Line 10) and the asserted hypothetical condition is checked to see if it results in the postconditions (Line 11). The predicate "cleanState" (Line 9) is used to retract all previously asserted dynamic clauses that might affect the asserted predicate "vulHyp(H,_vulID, Software,Range,Consequence)." After the asserted "vulHyp" is proved to cause the postconditions, the hypothesis is evaluated as the potential cause of the attack. Note that an investigator should perform an additional examination or even simulate the attack for the purpose of validation, especially when multiple hypotheses can explain the same attack.

5.2 Integrating Evidence Standards

Federal evidence admissibility criteria place additional constraints on evidentiary data and data handling procedures, which include chain of custody. Whenever the admissibility of digital evidence is called into

question, five federal rules are applied: (i) authenticity (Rules 901 and 902); (ii) hearsay or not (Rules 801–807); (iii) relevance (Rule 401); (iv) prejudice (Rule 403); and (v) original writing (Rules 1001–1008) [8], the most important being the relevance criterion. If these constraints are not considered, the evidence runs the risk of being ruled as insufficient.

The federal rules were codified in the Prolog logic-based framework [9] to determine the admissibility of evidence. The original MulVAL rules only use positive predicates in order to control complexity. The extended version incorporates negation to disqualify unacceptable evidence during an admissibility judgment.

Extended logic programs have two kinds of negations: (i) default negation that represents a procedural failure to find facts; and (ii) explicit negation (classic negation) that represents known negative facts [14]. Because a default negated predicate cannot be used as the head of a Prolog rule, default negated predicates (expressed using "\+" in XSB Prolog) are used in the body of a rule to exclude impossible facts, and an explicit negated predicate (expressed using "-" in XSB Prolog) is used in the head of a rule to judge if a derived fact that represents the corresponding evidence holds. If the logic program that includes negated predicates, including explicit negated predicates and the corresponding rules, generates execution cycles due to negated predicates, then it is necessary to ensure that the program is stratified [3]. Figure 9 shows a stratified Prolog program that uses positive and explicit negated predicates to determine if an attacker can gain access to a host computer (i.e., web server or workstation in the example) using the network protocol and ports (Lines 9–12). The conclusion (Lines 13–20) shows that the attacker can access the web server via TCP on Port 80, but not Port 8080. As such, only the evidence based on accessing the web server via TCP on Port 80 is acceptable when constructing an attack scenario.

In addition to using (explicit) negated predicates to exclude unacceptable evidence, in order to accommodate the federal rules of evidence, rules related to "timestamp," "relevance" and "not hearsay" were added to enhance the determination of evidence acceptability. Predicate "timeOrder" is used to verify if the attack steps are constructed in chronological order and the corresponding evidence falls in a reasonable timeframe. Predicate "vulRelevance" models expert knowledge, bug reports and vulnerability databases to determine if the given evidence is relevant to the observed attack. Predicate "notHearsay" is used to ensure that the evidence resource is not declared "hearsay" (a verbal report is generally not admissible). Interested readers are referred to [9] for a detailed discussion related to the federal rules of evidence in the context of the extended MulVAL framework.

```
1. nnetAccess(H,Protocol,Port):-
2. nattackerLocated(Zone),
3. nhacl(Zone,H,Protocol,Port).

4. -nnetAccess(H,Protocol,Port) :-
5. nattackerLocated(Zone),
6. -nhacl(Zone,H,Protocol,Port).

7. nattackerLocated(internet).
8. -nattackerLocated(webServer).
9. nhacl(internet,webServer,tcp,80).
10. nhacl(internet,workstation,tcp,4040).
11. nhacl(internet,workstation,udp,6060).
12. -nhacl(internet,webServer,tcp,8080).

13. | ?- -nnetAccess(webServer,tcp,8080).
14. yes
15. | ?- nnetAccess(webServer,tcp,8080).
16. no
17. | ?- nnetAccess(webServer,tcp,80).
18. yes
19. | ?- -nnetAccess(webServer,tcp,80).
20. no
```

Figure 9. Rule using explicit negation.

6. Experimental Results

The framework was supplied with the experimental evidentiary data and the new evidence graph shown in Figure 10 was obtained. The new evidence graph has several differences compared with the previous evidence graph in Figure 6. First, the attack path ($Node6 \rightarrow Node19 \rightarrow Node18 \rightarrow Node17 \rightarrow Node16$) on "Workstation 2" in Figure 6 is removed. This is because the evidence is not acceptable as false negatives are used (According to MITRE's OVAL database, "Workstation 2" is a Linux machine that uses Firefox as the web browser, which does not support a successful attack using "CVE-2009-1918" that only succeeds on Windows Internet Explorer). Second, a new attack path ($Node1 \rightarrow Node42 \rightarrow Node43$) corresponding to the phishing attack on the clients launched using the compromised administrator's session ID is added. This is obtained by using abductive reasoning on predicate "exec(client,_)" and a further investigation of the declared "hearsay" (the clients' phishing reports). Third, an attack path between the compromised workstation and the database server ($Node27 \rightarrow Node38 \rightarrow Node11$) is added; using the anti-forensic database, the reasoning system discovered that the attacker used the compromised workstation to gain

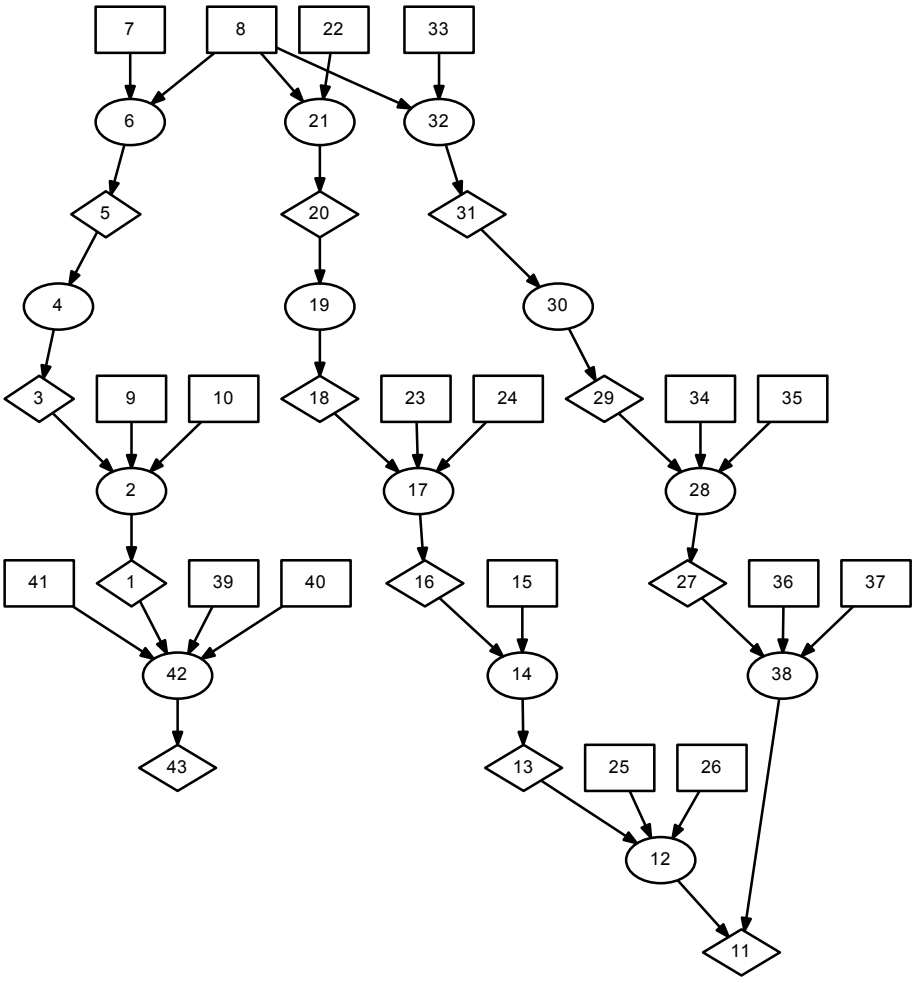


Figure 10. New reconstructed attack scenario.

access to the database server. Note that the reason why evidence could not be found is because the attacker was able to remove all the evidence using escalated root privileges that were obtained in a malicious manner.

Clearly, the reconstructed attack scenario obtained using the extended MulVAL framework (Figure 10) is considerably different from the reconstructed attack scenario obtained without the extension (Figure 6). This demonstrates that the extended framework can account for missing and/or destroyed evidence and can enhance the acceptability of a reconstructed attack scenario.

7. Conclusions

The network forensic model described in this chapter extends the MulVAL Prolog logic-based reasoning framework to automate the causality correlation of evidentiary data collected after a security event in an enterprise network. The extended model uses inductive and abductive reasoning, an anti-forensic database and legal acceptability standards for evidence to construct evidence graphs for network forensic analysis. The extension also excludes evidence such as false positives that are inadmissible and provides explanations for missing and destroyed evidence. In addition, it automates the process of using evidence that meets acceptability standards for attack scenario reconstruction.

Future research will attempt to develop a method for finding the best explanation of a network attack from among alternative explanations, validate the framework using realistic attack scenarios and work with attorneys to refine the evidence acceptability determination procedure. Also, attempts will be made to standardize the anti-forensic database.

Note that this chapter is not subject to copyright in the United States. Commercial products are only identified in order to adequately specify certain procedures. In no case does such identification imply a 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] O. Dain and R. Cunningham, Building scenarios from a heterogeneous alert stream, *Proceedings of the IEEE SMC Workshop on Information Assurance and Security*, pp. 231–235, 2001.

[2] H. Debar and A. Wespi, Aggregation and correlation of intrusion-detection alerts, *Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection,* pp. 85–103, 2001.

[3] M. Fitting and M. Ben-Jacob, Stratified and three-valued logic programming semantics, *Proceedings of the Fifth International Conference and Symposium on Logic Programming,* pp. 1054–1069, 1988.

[4] J. Keppens and J. Zeleznikow, A model based reasoning approach for generating plausible crime scenarios from evidence, *Proceedings of the Ninth International Conference on Artificial Intelligence and Law*, pp. 51–59, 2003.

[5] 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.

[6] C. Liu, A. Singhal and D. Wijesekera, Using attack graphs in forensic examinations, *Proceedings of the Seventh International Conference on Availability, Reliability and Security,* pp. 596–603, 2012.

[7] C. Liu, A. Singhal and D. Wijesekera, Creating integrated evidence graphs for network forensics, in *Advances in Digital Forensics IX,* G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 227–241, 2013.

[8] C. Liu, A. Singhal and D. Wijesekera, A model towards using evidence from security events for network attack analysis, *Proceedings of the Eleventh International Workshop on Security in Information Systems*, pp. 83–95, 2014.

[9] C. Liu, A. Singhal and D. Wijesekera, Relating admissibility standards for digital evidence to attack scenario reconstruction, *Journal of Digital Forensics, Security and Law*, vol. 9(2), pp. 181–196, 2014.

[10] MITRE, Open Vulnerability and Assessment Language: A Community-Developed Language for Determining Vulnerability and Configuration Issues in Computer Systems, Bedford, Massachusetts (`oval.mitre.org`), 2015.

[11] 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.

[12] X. Ou, S. Govindavajhala and A. Appel, MulVAL: A logic-based network security analyzer, *Proceedings of the Fourteenth USENIX Security Symposium*, 2005.

[13] P. Sommer, Intrusion detection systems as evidence, *Computer Networks*, vol. 31(23-24), pp. 2477–2478, 1999.

[14] T. Swift, D. Warren, K. Sagonas, J. Friere, P. Rao, B. Cui, E. Johnson, L. de Castro, R. Marques, D. Saha, S. Dawson and M. Kifer, The XSB System Version 3.6.x, Volume 1: Programmer's Manual (`xsb.sourceforge.net/manual1/manual1.pdf`), 2015.

[15] W. Wang and T. Daniels, A graph based approach towards network forensics analysis, *ACM Transactions on Information and System Security*, vol. 12(1), article no. 4, 2008.