Chapter 1

# A LOGIC-BASED NETWORK FORENSICS MODEL FOR EVIDENCE ANALYSIS

Changwei Liu, Anoop Singhal and Duminda Wijesekera

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

In this paper, we present a model and an accompanying software tool that systematically addresses how to resolve the above problems to reconstruct attack scenarios that could stand up in court. These problems include a large amount of data including non-relevant data, missing evidence and incomplete evidence destroyed by using anti-forensic techniques. Our system is based on a Prolog system using known vulnerability databases and an anti-forensic database that we plan to extend to a standardized database like the existing NIST National Vulnerability Database (NVD).

In this model, we use different methods, including mapping the evidence to system vulnerabilities, inductive reasoning and abductive reasoning, to reconstruct attack scenarios. The goal of this work is to reduce the security investigators' time and effort in reaching definite conclusion about how an attack occurred. Our results indicate that such a reasoning system can be useful for network forensics analysis.

**Keywords:** Network forensics, cybercrime, digital evidence, Prolog reasoning, network attack scenario, evidence graph, admissibility

## 1. Introduction

Network forensics is the science that deals with the capture, recording and analysis of network events and traffic for detecting intrusions and investigating them, which involves post mortem investigation of the attack and is initiated after the attack has happened. Different stages of legal proceedings (obtaining a warrant or evidence to the jury) require constructing an attack scenario from an attacked system. In order to present the scenario that is best supported by evidence, digital forensics investigators analyze all possible attack scenarios reconstructed from the available evidence, which includes false negatives, or those items of evidence that are missing or destroyed (in part or as a whole) due to reasons like (1) the tools used for digital forensics are not able to capture some attacks, and (2) attackers use anti-forensic techniques to destroy evidence.

Although using IDS alerts and system logs as forensic evidence has been contested in courts, they provide the first level of information to forensics analysts on creating potential attack scenarios [15]. In order to reconstruct potential attack scenarios by using evidence such as IDS alerts, researchers have proposed aggregating redundant alerts by similarities and correlating them by using pre-defined attack scenarios to determine multi-step, multi-stage attacks [11, 16]. Currently, this method is non-automated and rather ad-hoc. As an improvement, Wang at el. [1] proposed automating the process by using a fuzzy-rule based hierarchical reasoning framework to correlate alerts using so-called local rules and group them using so-called global rules. However, this approach falls apart when evidence is destroyed, and it does not assess the potential of the evidences admissibility so that the constructed attack scenario presented to a judge or jury has legal standing. In order to resolve these problems, we propose to build a rule-based system to automate the attack scenario reconstruction process that is cognizant of the admissibility standards for evidence. Our rule base provides (1) correlation rules to coalesce security event alerts and system logs, (2) rules to explain missing or destroyed evidence (with the support of an anti-forensics database) by using what if scenarios, and (3) rules to judge some of the acceptability standards of evidence. We show the viability of our system by building a prototype using Prolog.

### 1.1 The Model of Using Evidence from Security Events for Network Attack Analysis

The paper presented here is based on our preliminary proposed Prolog reasoning based model described in [2] , which is created by extending

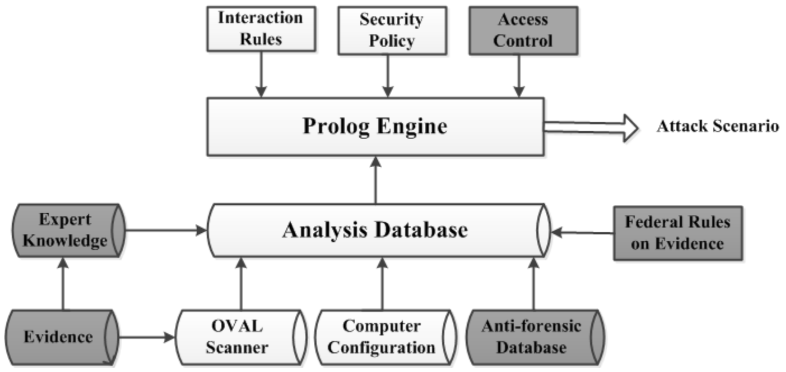


*Figure 1.* Architecture of MulVAL and Extended Models

an attack graph generation tool MulVAL [10] that is implemented on XSB Prolog [12].

The architecture of MulVAL and our extended model is illustrated in Figure 1. The extended modules in this model are shown in gray color, which include (1) the module "evidence" that uses MITRE's OVAL database [9] or expert knowledge (if there is no corresponding entry in OVAL database) to convert evidence from the attacked network to corresponding software vulnerability and computer configuration that MulVAL takes; (2) the modules "anti-forensics" database and "expert knowledge" database that are integrated into the extended MulVAL to generate explanations for the missing or destroyed evidence; (3) the modules of "Federal Rules on Evidence" and "Access Control" that are codified to the extended architecture for evidence acceptability judgment.

The rest of the paper is constructed as follows. Section 2 is background and related work. Section 3 is a motivating network example. Section 4 explains how we use rules to reconstruct an attack scenario from the obtained evidence. Section 5 discusses our approaches of adding predicates and rules to implement the proposed model. Section 6 presents our experiment result and section 7 concludes the paper.

## 2. Background and Related Work

### 2.1 MulVAL, XSB and Logical Attack Graph

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 to determine if they give rise to

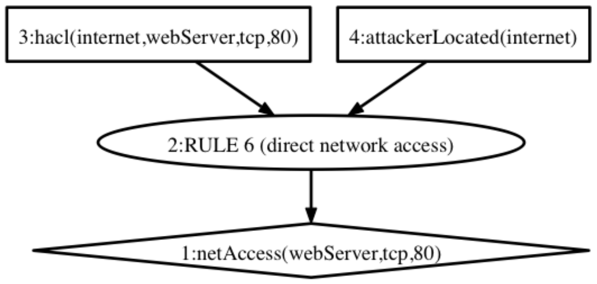

3:hacl(internet,webServer,tcp,80)
4:attackerLocated(internet)
2:RULE 6 (direct network access)
1:netAccess(webServer,tcp,80)

*Figure 2.* an Example Logical Attack Graph

potential attack traces. The graph composed of all attack traces generated by this system is defined as 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 [4, 7].

Figure 2 shows an example logical attack graph. A primitive fact node (box) represents specific network configuration or vulnerability information that is 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.

## 2.2 Evidence Graph

Different from an attack graph that predicts potential attacks, an evidence graph is constructed by using evidence, aiming to hold attackers to their crimes towards an enterprise network.

**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 two 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 mapping from a node to its label, $N_h$-Attr and $N_e$-Attr are attributes of a host node and an evidence node respectively [3].

The attributes of a host node include “host ID”, “states” and “timestamps”. The “states” consist of the states before and after a particular attack step, which can be “source”, “target”, “stepping-stone” or “affiliated” [3, 4]. The attributes of an evidence node describe the event initiator, target and its timestamp.

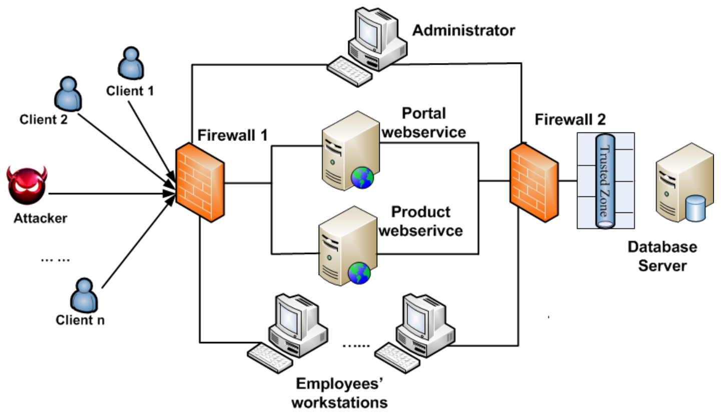


*Figure 3.* an Experimental Attacked Network

## 2.3 Related Work

Reasoning has been used to correlate evidence to reconstruct crime scenarios. In the area of non-digital forensics, researchers use inductive and abductive reasoning to model potential crime scenarios and correlate evidence [13]. In the area of digital forensics, paper [1] described a fuzzy-rule base to correlate attack steps substantiated by aggregated security event alerts. This schema 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 to the original event. However, this work did not provide a good way to resolve the problem when the evidence is missing or incomplete, nor did it use any standards to determine the acceptability of evidence. In order to solve these limitations, we proposed to use an anti-forensics database to implement the expert knowledge, so that it can help generate hypotheses about the missing or destroyed evidence to substantiate an expert's default assumptions [6], and use MITRE's OVAL database and corresponding federal rules on digital evidence to determine the potential acceptability of digital evidence [5]. Some of the proposed solutions were not implemented, which will be discussed in this paper.

## 3. A Network Example

Figure 3 is an example network from [5]. Based on this network, in order to explain how to use an anti-forensic database to help find explanations on destroyed evidence, we used anti-forensics techniques to remove

*Table 1.* Machine IP Address and Vulnerability

| 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–Portal 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 | |

*Table 2.* Formalized Evidence of the Alerts and Log from Figure 3

| Timestamp | Source IP | Destination IP | 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 |
| ... | | | | |

some evidence. Table 1 shows the machine IP address and vulnerability information. By exploiting vulnerabilities listed in Table 1, the attacker was able to successfully launch the following attacks: (1) compromising a workstation (CVE-2009-1918) to access the database server, (2) using the vulnerability on the product web application (SWE89) to attack the database server, and (3) exploiting a cross-site scripting (XSS) vulnerability on a chatting forum hosted by the portal web service to steal the administrator's session ID so that the attacker can send out phishing emails to the clients, tricking them to update their confidential information.

In this experimental network, the installed intrusion detection system, configured webserver and database server were able to detect some attacks and log malicious accesses, which, however, had false positives (e.g., the attack attempts that were not successful). Besides, because SNORT used for IDS did not have corresponding rules or the actual attack activities looked benign, some attacks might not be caught by the IDS, and therefore did not log any IDS alerts as evidence. Two such examples are: (1) the phishing URLs the attacker sent to the clients for confidential information updating could not be caught; (2) the attacker's accessing the database server from the compromised workstation was thought benign. In addition, the attacker compromised the workstation and obtained root privilege, which enabled him to use anti-forensic techniques to delete the evidence left on the workstation. Under these conditions where evidence is missing or destroyed, we need to find a way to show how the attack might have happened.

## 4. Attack Scenarion Reconstruction

### 4.1 Rules and Facts Used for Reasoning

As stated in Section 1, we used the formalized vulnerability/forensics database constructed from MITRE's OVAL [9] to convert IDS alerts and corresponding system logs to the corresponding vulnerability entries for an attack scenario reconstruction (expert knowledge is used only when the corresponding entry cannot be found in OVAL) [2]. Table 2 shows the converted evidence from our experimental network in Figure 3 , and Figure 4 are the concrete predicates written from these items of evidence, corresponding computer configuration and network topology, which will instantiate the corresponding predicates in reasoning rules during a MulVAL run. In these predicates, (1) Predicate "attackedHost" represents a destination victim computer; (2) Predicate "hacl" means a host access control list; (3) Predicate "advances" represents the access rights within the firewall, which were used by the attacker to reach the next computer after the attacker had comprised a computer as stepping-stone; (4) Predicate "timeOrder" ensures that an attack step's start time and end time fell within a reasonable interval; and (5) predicates "vulExists", "vulProperty" and "networkServiceInfo" as follows represent an attack step on the target computer(the first term in the predicate of "networkServiceInfo").

*"vulExists(workStation1, 'CVE-2009-1918', httpd).*
*vulProperty('CVE-2009-1918', remoteExploit,privEscalation).*
*networkServiceInfo(workStation1 , httpd, tcp , 80 ,apache)."*

Figure 5 shows one of the reasoning rules that describe generic attack techniques in this logic-based network forensic system. Each rule has Prolog tuples deriving the post-conditions from the pre-conditions of an attack step. For example, the rule in Figure 5 represents, if (1) the attacker has compromised the victim's computer (Line 3), (2) the victim has the privilege "Perm" on the host computer "Host"(Line 4), and (3) the attacker can access the victim's computer (Line 5), then the evidence representing the three preconditions as the cause is correlated to the evidence representing the attacker has obtained the victim's privilege on

```
/*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

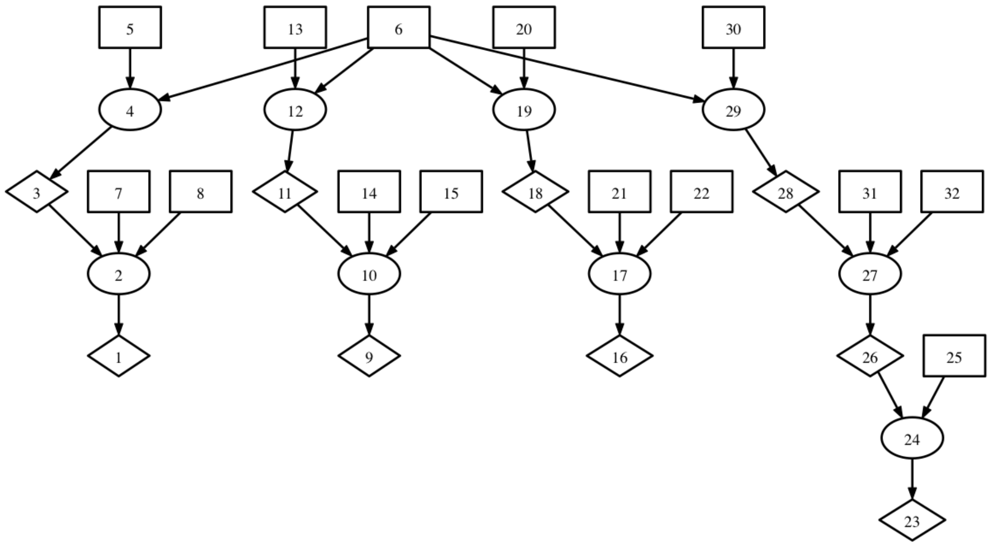
```
/**** Interaction Rules *****/
1. interaction_rule(
2. (execCode(Host, Perm) :-
3. principalCompromised(Victim),
4. hasAccount(Victim, Host, Perm),
5. canAccessHost(Host)),
6. rule_desc('When a principal is compromised any machine he has an account on
will also be compromised',0.5)).
```

*Figure 5.* An Example Reasoning Rule

the host computer (Line 2). Line 1 is a string that uniquely identifies a rule, and Line 6 is the description of the rule.

## 4.2 Evidence Graph Generation

Querying the logic-based system that has not been integrated with any anti-forensic database and evidence acceptability standards gen-



| | |
|---|---|
| 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,80) | 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.* Experimental Network Attack Scenario Reconstructed from the Alert/Log

erates the attack scenario as Figure 6 (The table below describes the notations of all nodes) with the corresponding evidence (hence an evidence graph) in the logical form defined in Definition 1, where all facts including primary facts(boxes) and derived facts(diamonds) before a derivation node(an ellipse) represent the pre-conditions before an attack step, and all facts after a derivation node represent post-conditions after the attack step. Figure 6 shows the four attack paths constructed by this process: (1) the attacker used a cross-site scripting attack (XSS) to steal the administrator's session ID and therefore obtain the administrator's privilege ($6 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$); (2) the attacker used a web application that does not sanitize users' input (CWE89) to launch a SQL injection attack towards the database ($6 \rightarrow 29 \rightarrow 28 \rightarrow 27 \rightarrow 26 \rightarrow 24 \rightarrow 23$); (3) the attacker used a buffer overflow vulnerability (CVE-2009-1918) to compromise workstations ($6 \rightarrow 12 \rightarrow 11 \rightarrow 10 \rightarrow 9$ and $6 \rightarrow 19 \rightarrow 18 \rightarrow 17 \rightarrow 16$).

Because corresponding evidence is missing and destroyed, the phishing attack that has been observed on the clients' computers and the attack towards the database server done by using the compromised workstations were not constructed and shown in Figure 6. Also, because the available evidence used for attack scenario reconstruction has not been validated by any standards of acceptability, Figure 6 might not reflect the real attack scenario, and hence has little impact in a court of law.

## 5. Extending MulVAL for Attack Scenario Reconstruction

### 5.1 Using Anti-forensic Database to Explain Missing/Destroyed Evidence

We use abductive reasoning and an anti-forensic database in our Prolog-based framework to explain how a host computer might have been attacked when the evidence is missing or destroyed [2, 6].

By using abductive reasoning, our extended Prolog logic-based framework can provide all potential general explanations about how an attacker might have successfully launched a particular attack. For example, in order to find the explanations on how the database server in Figure 3 might have been attacked, we constructed a query file as illustrated in Figure 7 to query the extended logic-based system to show all attack steps that would cause "execCode(dbServer,user)" (the attack on the database server) as the explanatory hypotheses. In Figure 7, lines 2 to 5 require the system to list all attack steps of "execCode(dbServer,user)" and write the results to "queryresult.P" file( line 2 opens output stream for writing to "queryresult.P" and line 5 closes

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

*Figure 7.* Example of Querying the System for Explanatory Hypotheses

*Table 3.* The Anti-forensic Database

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

the output stream). The returned query results indicate that three possible hypotheses could cause "execCode(dbServer,user)". They are (1) using a compromised computer as the stepping stone, (2) using the vulnerability of the database access software, and (3) using the legitimate account in the database server to inject malicious input.

Once all possible hypotheses have been generated, in a subsequent evaluation process, our extended Prolog logic-based framework uses an anti-forensic database to decide which explanation could be the best. Table 3 is the example anti-forensic database constructed in paper [6], which instantiates our constructed predicate "anti_Forensics(Category, Tool, Technique, Windows, Linux, Privilege, Access, Program, Consequence)" as illustrated 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 post conditions collected from an attacked computer when the attack evidence is missing or has been destroyed.

Line 2 to 11 in Figure 8 are two rules that illustrate using "anti_Forensics(Category, Tool, Technique, Windows, Linux, Privilege, Access, Program, Consequence)" to evaluate whether the hypothesis that the attacker has used the vulnerability on the database access software to attack the database is the best explanation in the experimental network. In the first rule (Line 2 to 7), the head "vulHyp(H, _vulID, Software, Range, Consequence)" (the hypothetical vulnerability the attacker might have used) is derived from three sets of predicates: (1) the hypothesis obtained from one of the results of Figure 7(Line 3 to 5), (2) the "anti-Forensics" predicate in Line 6, where the variable terms

```
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 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 Anti-forensics Database to Explain Missing/Destroyed Evidence

("Category", "Tool", "Technique", "Windows", "Linux", "Privilege", "Access", "Program", "Consequence") will be instantiated by the corresponding concrete data from Table 3 during the system run, (3) the configuration of the host (in Line 7) where the evidence has been destoryed by attacker's using anti-forensic technique. In the second rule (from Line 8 to Line 11), we assert the derived fact "vulHyp(H, _vulID, Software, Range, Consequence)" obtained from the first rule to the logic runtime database (Line 10), checking whether the asserted hypothetical condition results in the post conditions(Line 11). The predicate "cleanstate" (Line 9) is used to retract all previous asserted dynamic clauses that might affect the asserted "vulHyp(H, _vulID, Software, Range, Consequence)". Once the asserted "vulHyp" is proved to cause the post conditions, the hypothesis is evaluated as the potential cause of the attack. Investigators should perform a further investigation or even simulate the attack for the purpose of validation, especially when there are different hypotheses that can explain the same attack.

## 5.2 Integrating Evidence Acceptability Standards

Federal admissibility criteria of evidence place additional constraints on the data and their handling procedures, which include the chain of custody issues. Whenever the admissibility of digital evidence is called into question, the following federal rules are applied: (1) Authenticity (Rules 901 and 902), (2) Hearsay or not (Rule 801-807), (3) Relevance (Rule 401), (4) Prejudice (Rule 403), (5) Original writing (Rule 1001-

1008)[8], where the most important rule is relevance criterion. Without considering these constraints, the prosecution runs the risk of evidence being ruled as insufficient.

In order to ascertain the legal admissibility standards of evidence, we codified the federal rules into our Prolog logic-based framework [5] to determine the admissibility of evidence. The original MulVAL rules only use positive predicates in order not to increase the complexity of the system. We extended the system with negation to disqualify unacceptable evidence for admissibility judgement.

Extended logic programs have two kinds of negations—default negation representing procedural failure to find facts and explicit negation (classic negation) representing known negative facts [12]. Because a default negated predicate cannot be used as the head of a Prolog rule, we use default negated predicates (expressed by using "\+" in XSB Prolog) in the body of a rule to exclude impossible facts, and use an explicit negated predicate (expressed by using "-" in XSB Prolog) as the head of a rule to judge if a derived fact representing corresponding evidence holds. In case the logic program that includes negated predicates including explicit negated predicates and corresponding rules generates execution cycles due to negated predicates, we ensure that the program is stratified [14]. Figure 9 shows an example stratified prolog logic program, which uses both positive and explicit negated predicates to determine if the attacker can get access to a host computer (i.e., webserver or workstation in our example) by using the network protocol and ports shown between Line 9 to Line 12. The conclusion (between Line 13 to Line 20) shows that the attacker can access the webserver by TCP at Port 80, but Port 8080. As such, only the evidence based on accessing webserver by using TCP through Port 80 can be acceptable to construct attack scenarios.

In addition to using (explicit) negated predicates to exclude unacceptable evidence, according to the federal rules on digital evidence, we added rules related to "timestamp", "relevancy" and "not hearsay" to enhance the determination of evidence acceptability. Predicate"timeOrder" is used to verify whether the attack steps are constructed in a chronological order and the corresponding evidence falls in a reasonable timeframe; Predicate "vulRelevance" models expert knowledge, bug-report and vulnerability database to determine if the given evidence is relevant to the observed attack, and Predicate "notHearsay" is used to ensure that the evidence resource is not declared "hearsay"(e.x., verbal report is generally not admissible in law). Our paper [5] has a detailed discussion on relating the federal rules on digital forensics to the extended MulVAL framework, so we skip the details here.

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

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

nattackerLocated(internet).
-nattackerLocated(webServer).
nhacl(internet,webServer,tcp,80).
nhacl(internet,workstation,tcp,4040).
nhacl(internet,workstation,udp,6060).
-nhacl(internet,webServer,tcp,8080).

| ?- -nnetAccess(webServer,tcp,8080).
yes
| ?- nnetAccess(webServer,tcp,8080).
no
| ?- nnetAccess(webServer,tcp,80).
yes
| ?- -nnetAccess(webServer,tcp,80).
no
```

*Figure 9.* Example Rule that Uses Explicit Negation

## 6. Experimental Results

We tested our framework with our experimental evidence data, and obtained a new evidence graph as shown in Figure 10 with the following changes. First, the attack path ($Node6 \rightarrow Node19 \rightarrow Node18 \rightarrow Node17 \rightarrow Node16$) on "Workstation 2" from Figure 6 has been removed, because the evidence is not acceptable as false negatives have been used (According to MITRE OVAL database, the "Workstation 2" is a Linux machine that uses Firefox as the web browser, which does not support a successful attack by using "CVE-2009-1918" that only succeeds on Windows Internet Explorer). Second, a new attack path "$node1 \rightarrow node42 \rightarrow node43$" representing that the attacker launched a phishing attack towards the clients by using the compromised administrators session ID has been added. This is obtained by using abductive reasoning on predicate "exec(client,_)" and a further investigation on the declared "hearsay" (the clients' phishing reports). Third, an attack path between the compromised workstation and the database server ($node27 \rightarrow node38 \rightarrow node11$) has been added—with the use of an anti-forensics database, our reasoning system found out that the attacker used the compromised workstation to get access to the database server.

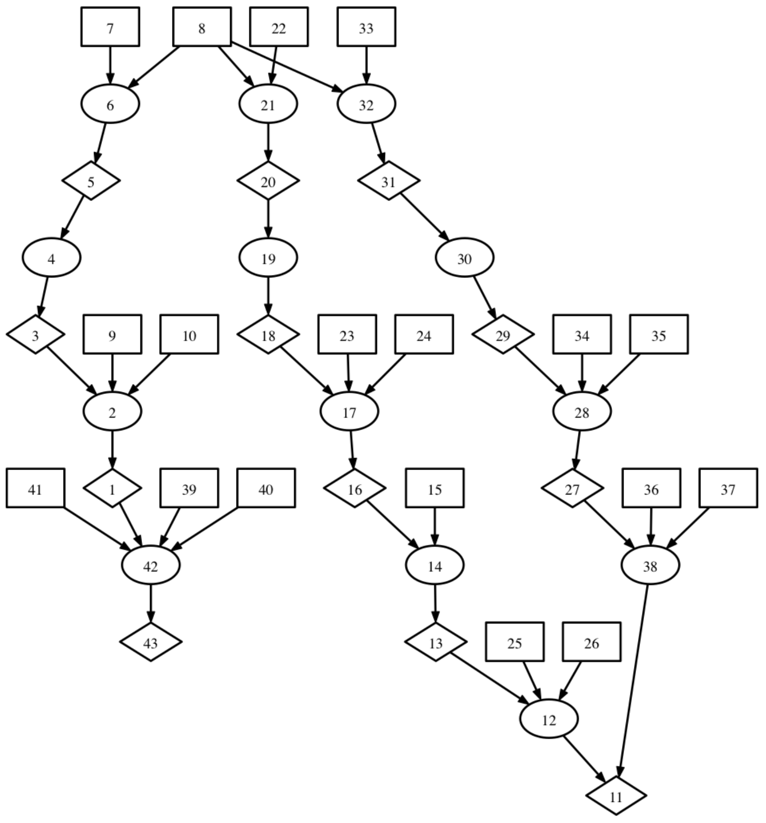


*Figure 10.* New Reconstructed Attack Scenarios by Using the Extended MulVAL

The reason why we could not find evidence is because the attacker was able to remove all evidence by using the escalated root privilege obtained maliciously.

We realized that the reconstructed attack scenario (given in Figure 10) by using our extended framework is different from the one (given in Figure 6) that we constructed by using the framework without the extension discussed in Section 5, showing that the extended framework can find the missing/destroyed evidence and enhance the acceptability of the reconstructed attack scenarios.

## 7. Conclusions and Future Work

We have proposed a network forensics model, which extends a Prolog logic-based reasoning framework, MulVAL, to automate the causality correlation between evidence collected from security events in an enterprise network. In this model, we use different methods, including inductive reasoning, abductive reasoning, working with an anti-forensics database and legal acceptability standards for evidence to construct an evidence graph for network forensics analysis. Our extension also excludes evidence such as false positives that are not admissible and provides explanations for the missing or destroyed evidence. In addition, our framework can automate the process of using evidence that supports a given level of acceptability standard for attack scenario reconstruction. Our ongoing work is trying to (1) find the best explanation when there are different explanations towards the same attacked network, (2) validate our system in realistic attack scenarios, and (3) work with cybercrime attorneys to ensure that acceptability determinations can be useful to them. Also, as the support to this extended network forensic Prolog-based framework, we plan to work with NIST to standardize the anti-forensics database.

## References

[1] W. Wang, T.E. Daniels, A graph based approach towards network forensics analysis, ACM Transactions on Information and System Security (TISSEC), 12 (1), Oct 2008.

[2] C. Liu, A. Singhal and D. Wijesekera, A Model Towards Using Evidence From Security Events For Network Attack Analysis, 11th International Workshop on Security in Information Systems, April 2014.

[3] C. Liu, A. Singhal, D. Wijesekera, Mapping Evidence Graphs to Attack Graphs, IEEE International Workshop on Information Forensics and Security, Dec 2012.

[4] C. Liu, A. Singhal, and D. Wijesekera, Creating Integrated Evidence Graphs for Network Forensics, IFIP Int. Conf. Digital Forensics 2013: 227-241.

[5] C. Liu, A. Singhal, and D. Wijesekera, Relating Admissibility Standards for Digital Evidence to Attack Scenario Reconstruction, JDFSL 9(2): 181-196 (2014).

[6] C. Liu, A. Singhal, and D. Wijesekera, Using attack graphs in forensic examinations, in Proceedings of the 7th International Conference on Availability, Reliability and Security (ARES '12), pp. 596603, IEEE, Prague, Czech Republic, August 2012.

[7] 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. 336345, 2006.

[8] Federal Rules of Evidence, Dec 1, 2010. Retrieved from http://www.uscourts.gov/uscourts/rulesandpolicies/rules/2010

[9] MITRE Open Vulnerability and Assessment Language-A Community-Developed Language for Determining Vulnerability and Configuration Issues on Computer Systems, retrieved from https://oval.mitre.org/.

[10] MulVAL: A logic-based enterprise network security analyzer, retrieved from http://www.arguslab.org/mulval.html.

[11] O. Dain,R. Cunningham, Building scenarios from a heterogeneous alert stream, In Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, pages 231-235, June 2001.

[12] David S. Warren et al. The XSB system version 3.1 volume 1: Programmer's manual. Technical Report Version released on August, 30, Stony Brook University, USA, 2007. (Available from URL: http://xsb.sourceforge.net/).

[13] J. Keppens and J. Zeleznikow, A model based reasoning approach for generating plausible crime scenarios from evidence, In Proceedings of the 9th International Conference on Artificial Intelligence and Law (2003).

[14] M. Fitting, M. Ben-Jacob, Stratified and Three-Valued Logic Programming Semantics, Proceedings of the 5th International Conference and Symposium on Logic Programming (1988), pp. 10541069.

[15] Sommer, P. (2003), Intrusion Detection Systems as Evidence, In Recent Advances in Intrusion detection 1998(RAID98), 1998.

[16] H. Debar, A. Wespi, Aggregation and correlation of intrusion-detection alerts, In Recent Advances in Intrusion Detection 2001, LNCS 2212, pages 85103, 2001.