A Layered Graphical Model for Mission Attack Impact Analysis

Changwei Liu
Department of Computer Science
George Mason University
Fairfax VA 22030 USA
cliu6@gmu.edu

Anoop Singhal
National Institute of Standards and Technology
100 Bureau Drive, Gaithersburg MD 20899 USA
anoop.singhal@nist.gov

Duminda Wijesekera
Department of Computer Science
George Mason University
Fairfax VA 22030 USA
dwijesek@gmu.edu

Abstract—**In this paper, we describe a layered graphical model to analyze the mission impacts of attacks for forensic investigation. Our model has three layers: the upper layer models operational tasks and their dependencies; the middle layer reconstructs attack scenarios by using a forensic tool to find the causality between items of evidence; the lower level reconstructs potentially missing attack steps due to missing evidence from the middle layer. Based on the graphs produced from the three layers, our model computes mission impacts by using NIST National Vulnerability Database(NVD) scores or forensic investigators' estimates. The case study shows our layered graphical model can be used for both forensic analysis and hardening an enterprise infrastructure.**

I. Introduction

In this paper, the mission for an enterprise is a set of business processes that provide a set of services. For example, the mission of a travel management system is to provide a set of business processes to support airline and hotel reservation. Organizational missions enabled by networked infrastructure can be impacted by cyber-attacks. Quantifying the impacts of cyber-attacks is of importance to mission planners. Mission impact evaluation approaches and tools provide a way to estimate the impacts of cyber-attacks on missions [1], of which NIST NVD Common Vulnerability Scoring System (CVSS) [2] provides impact estimates of exploitable vulnerabilities on IT systems. Researchers have expanded NIST NVD CVSS to multi-step attacks to predict the impacts of such attacks on missions by considering all possible vulnerabilities that construct all possible attack paths [1], [3], [4]. However, evaluating all paths is infeasible for forensic analysis to assess damages due to combinatorial explosion caused from considering all paths and vulnerabilities.

Because artifacts obtained from forensic investigations carried out after cyber-attacks provide information that can be used to analyze the attacks, we propose to create a layered graphical model that uses such information to quantify the attacks' impacts on missions. As far as we know, there is no such an integrated forensic analysis framework that quantifies the mission impacts of multi-step attacks in a complex enterprise infrastructure including cloud-based services.

The rest of the paper is organized as follows. Section II presents the three-layered graphical model. Section III uses a case study to show how the model computes mission impacts of attacks. Section IV discusses the related work, and Section V concludes this paper.

II. Our Three-layered Graphical Model

Figure 1 shows our model with three layers. The lower layers reconstruct attack paths so that the attacks can be mapped to tasks and missions in the upper layer for mission impact computation.

A. The Upper Layer

The upper layer models tasks and missions as business processes. We model business processes using a Business Process Diagram (BPD). Our BPDs are specified using the Business Process Modeling Notation (BPMN). BPMN models composite tasks by composing so-called atomic tasks using constructors hierarchically. The current version of BPMN (v 2.0) uses about 100 graphical elements, covering the description of many categories of tasks, events, errors, areas of responsibility, and general annotations [5]. A study of real-world BPMN usage found that the most used subset, labeled as the core subset of BPMN, consists of only eight elements including tasks, start/end events, exclusive decision gateways, parallel gateways, sequence flows, message flows and pools, which are the base of the definition of a BPD [6]. In this paper, we use BPDs constructed from those elements, formalized in Definition 1 (Originally defined in [6]).

Definition 1 (Business Process Diagram-BPD): In BPMN notation, a BPD is a graph BPD=(N, F, P, pool, L, lab), where:

1) $N \subseteq T \cup E \cup G$, in which tasks T are the basic actions performed as part of a business process, events $E \subseteq E^S \cup E^E$ consist of two disjoint sets E^S and E^E representing start and end events, gateways $G \subseteq G^M \cup G^F \cup G^D$ are the disjoint sets G^M, G^F and G^D representing parallel merge, parallel fork and exclusive decision gateways.
2) F $\subseteq$ S $\cup$ M is a set of flow relations, in which sequence flows $S \subseteq N \times N$ relate nodes including tasks, events, exclusive decision and parallel gateways to each other, message passing $M \subseteq T \times G^M$ is a relation between task nodes and parallel merge gateways, employed to pass messages between separate processes.
3) $P \subset P(N)$ is a set of disjoint pools.

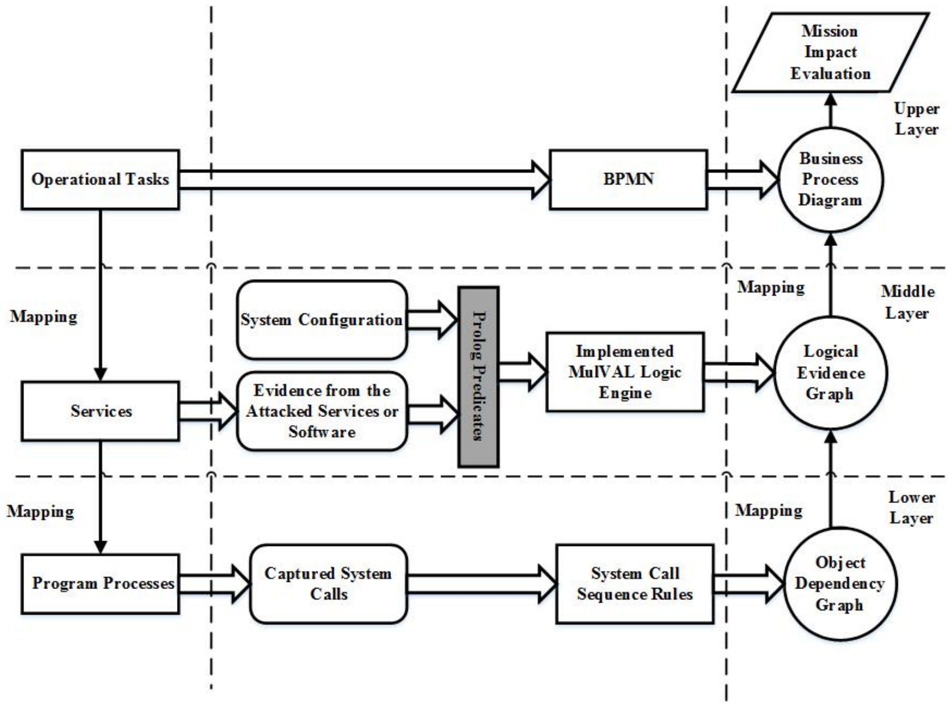

Fig. 1. The three-layered graph model for mission impact evaluation

4) Pool: $N \rightarrow P$ maps nodes to a pool $p \in P$. A pool is a basic BPMN element that sets the boundaries of a business process, which contains at most one business process.
5) L is a set of labels.
6) Lab: $F \rightarrow L$ is a labelling function that assigns labels to flows.

B. The Middle Layer

The middle layer creates attack scenarios using evidence available from system logs, intrusion detection system (IDS) alerts. Our objective is to map these attack scenarios to missions modeled as BPDs. Because we only consider attack scenarios substantiated using available evidence, the created attack paths do not include all possible attack paths and vulnerabilities that are infeasible for forensic analysis. However, sometimes, we may not be able to find all evidence to reconstruct all attack scenarios in this layer, which will be addressed in the lower layer.

We reconstruct attack scenarios by using a forensic analysis tool created in our previous work [7]. This tool uses rules to create directed graphs of available evidence and correlate them together as witnesses of attacks. Because rules are used to create these graphs, they are called logical evidence graphs (LEGs) formalized in Definition 2 (originally defined in [7]).

Definition 2 (Logical Evidence Graph-LEG): A LEG=(N_r, N_f, N_c, E, L, G) is said to be a logical evidence graph (LEG), where N_f, N_r and N_c are three sets of disjoint nodes in the graph (they are called fact, rule, and consequence fact nodes respectively), E $\subseteq ((N_f \cup N_c) \times N_r) \cup (N_r \times N_c)$, L is the mapping from a node to its labels, and G $\subseteq N_c$ are the observed attack events. Every rule node has a consequence fact node as its single child and one or more fact or consequence fact nodes from prior attack steps as its parents. The labels of nodes consist of instantiations of rules or sets of predicates specified as follows:

1) A node in N_f is an instantiation of predicates that codify system states including access privileges, network topology consisting of interconnectivity information, or known vulnerabilities associated with host computers in the system. We use the following predicates:
   a) "hasAccount(_principal,_host, _account)", "canAccessFile(_host, _user, _access, _path)" and etc. to model access privileges.
   b) "attackerLocated(_host)" and "hacl(_src, _dst, _prot, _port)" to model network topology, namely, the attacker's location and network reachability information.
   c) "vulExists(_host, _vulID, _program)" and "vulProperty(_vulID, _range, _consequence)" to model vulnerabilities exhibited by nodes.
2) A node in N_c represents a predicate that codifies the post attack state as the consequence of an attack step. We use predicates "execCode(_host,_user)" and "netAccess(_machine,_protocol, _port)" to model the attacker's capability after an attack step. Valid instantiations of these predicates after an attack will update valid instantiations of the predicates listed in (1).
3) A node in N_r consists of a single rule in the form $p \leftarrow p_1 \wedge p_2, \cdot\cdot\cdot, \wedge p_n$. p as the child node of N_r is an instantiation of predicates from N_c. All p_i for $i \in$

$\{1 \ldots n\}$ as the parent nodes of N_r are the collection of all predicate instantiations of N_f from the current step and N_c from the prior attack step.

C. The Lower Layer

This layer uses instances of interactions between services and the execution environment to obtain evidence unavailable from systems logs and IDS alerts. We obtain such interaction instances from systems call logs. This usage is based on our postulate of missing evidence due to the attackers' using anti-forensic techniques, limitation of forensic tools, or zero-day attacks. Because there are many system calls, we use those used in [8], listed in the right-hand column of Table 1. Our abstraction of them appear in the left-hand column of Table 1. A process making system calls creates dependencies between itself and other processes, files, or sockets for network connection. We model these dependencies as graphs that we call object dependency graphs (ODGs), formalized in Definition 3.

Definition 3 (Object Dependency Graph-ODG): The reflexive transitive closure of " $\rightarrow$ " defined in Table 1 is an object dependency graph. We use the notation ODG=(V_O, V_E, E) to represent an object dependency graph, where V_O is the set of vertexes that are composed of objects including Processes P, Files F or Sockets S; V_E is the set of textual event descriptions listed in the middle column; and E is the set of dependency edges listed in the left-hand column of Table 1.

D. The Mapping between the Three Layers

The left-hand and right-hand columns in Figure 1 show the system resource mapping and graph mapping of our model. We use the resource mapping obtained from the infrastructure configuration and software deployment to map graphs. To do so, we add the attacked services as attributes to the corresponding nodes(vertexes) in LEGs and ODGs, so that the mapping can match between node attributes of source and destination graphs. These nodes are either processes, infrastructure services or tasks that are the entities shown in the left-column of Figure 1. A LEG is easily mapped to a BPD by matching the attacked services to the corresponding tasks. We use depth first search (DFS) method to map an ODG to a LEG, which is explained in Algorithm 1.

In Algorithm 1, the first *for* loop colors all object nodes in an ODG white, marking them as *having not been checked*. The second *for* loop repeatedly calls $Find(V_O, LEG)$ function, where, for a given white node V_O, the algorithm attempts to find the matching Node N_{c1} by checking if the attacked service in the LEG is equal to the attacked service in the ODG(the pseudo code in $Find(V_O, LEG)$ function is $N_c.service == V_O.service$). If such a post attack status node N_{c1} is found, the algorithm checks if the attack step between Node V_O and its parent node *parent(V_O)* in the ODG has a mapping attack step between Node N_{c1} and its parent node(s) *parent(N_{c1})* in the LEG. If there is no such a mapping attack step, the attack step is added to the LEG. If there is no any matching post attack status Node N_{c1} for node V_O, one is added to the

```
Input: An ODG=$(V, V_E, E)$ and a
       LEG=$(N_r, N_f, N_c, E, L, G)$.
Output: A LEG integrated with attack paths from the
        ODG.
//Color all nodes in ODG WHITE
for each node $V_O$ in ODG do
    color[$V_O$] ← WHITE
end
//Go through each object in ODG
for each node $V_O$ in ODG do
    if $V_O$ == WHITE then
        //Initialize all nodes in LEG white
        for each node $N_c$ in LEG do
            color[$N_c$] ← WHITE
        end
        //Search for the corresponding $N_{c1}$ in LEG
        $N_{c1}$ = Find($V_O$, LEG)
        //If there is such a matching $N_{c1}$
        if $N_{c1} \neq \emptyset$ then
            color($V_O$) ← BLACK
            //See if the object's parent matches
            //corresponding $N_{c1}$' s parent
            $N_{c2}$=Find(parent($V_O$), LEG)
            //If not matching parents,
            //add the missing attack step from ODG to
             LEG
            if $N_{c2} \neq parent(N_{C1})$ then
                LEG ← Flow($N_{c1}, V_E$)
                LEG ← Flow($V_E, N_{c2}$)
            end
        end
        else
            //If there is no such a matching $N_{c1}$ in LEG
            //Add the new object to LEG
            LEG ← $V_O$
            color [$V_O$]=GRAY
        end
        $V_O$ =child($V_O$)
    end
end
Function Find($V_O$, LEG)
    //Go through each $N_c$ in LEG
    for each post attack status $N_c$ from LEG do
        //Check if there is any matching $N_c$ for $V_O$
        if $N_c.service == V_O.service$ AND
        color[$N_c$] == white then
            color[$N_c$] ← BLACK
            return $N_c$
        end
        else
            color[$N_c$] ← GRAY
            $N_c$ ← the child post attack status node of $N_c$
        end
    end
    return $\emptyset$
```
Algorithm 1: Mapping an ODG to a LEG

TABLE I
DEPENDENCIES ARISING OUT OF SYSTEMS CALLS

Dependency	Event Description	Unix System Calls
$process \rightarrow file$	Process modifies file	write, pwrite64, rename, mkdir, linkat, link, symlinkat, etc
$file \rightarrow process$	Process reads file	stat64, lstat6e, fsat64, open, read, pread64, execve, etc.
$process \leftrightarrow file$	Process uses/modifies file	open, rename, mount, mmap2, mprotect etc.
$process1 \rightarrow process2$	Process1 creates/terminates Process2	vfork, fork, kill, etc.
$process \rightarrow socket$	process writes socket	write, pwrite64, etc.
$socket \rightarrow process$	process checks/reads socket	fstat64, read, pread64, etc.
$process \leftrightarrow socket$	Process reads/writes/checks socket	mount, connect, accept, bind, sendto, send, sendmsg, etc.
socket $\leftrightarrow$ socket	process reads/writes socket	connect, accept, sendto, sendmsg, recvfrom, recvmsg

LEG and the search continues until all nodes in the ODG are checked(colored).

E. Mission Impact Computation

We propose to use the interval [0,1] to quantify a mission impact of an attack, computed by using the following steps.

- **Compute the impact scores of attacks in a LEG**
  In a LEG, we use P(a) to represent the impact of attacks on a service deployed on a host computer. NIST NVD CVSS published reported vulnerabilities with assigned impact scores, which we propose to use for each P(a) if an attack a can be found in NIST NVD. If the attack a cannot be found in NIST NVD, we suggest using expert knowledge to assign an impact score to P(a). We use our previous work [9] to compute a cumulative impact score of an attack as follows.

$$P(a) = P(a_1) \cup P(a_2) \tag{1}$$

  In Equation 1, a_1 and a_2 are two attacks on the same service. $P(a_1) \cup P(a_2) = P(a_1) + P(a_2) - P(a_1) \times P(a_2)$.
- **Assign weight to tasks/missions**
  A value between [0,1] is proposed as the weight of mission impacts, indicating the importance of the corresponding tasks/missions.
- **Map LEGs to BPDs**
  We map LEGs integrated with missing attack steps to BPDs so that the mission impact of attacks I(B) on a business process B is computed using Equation 2.

$$I(B) = weight \times P(B) \tag{2}$$

  In Equation 2, P(B) is the impact of attacks on a business process *B* in a BPD. It is computed by using Equation 3 and Equation 4 respectively, since the mapping from attacks(represented by a, a_1, a_2) in a LEG to a business process(represented by *B*) in a BPD has two relationships including one-to-one(Equation 3) or many-to-one(Equation 4).

$$P(B) = P(a) \tag{3}$$

$$P(B) = P(a_1) \cup P(a_2) \tag{4}$$

- **Compute the cumulative mission impact**
  Mission impact of attacks on each business process(task) can be computed using Equation 2, Equation 3 and Equation 4. However, in some cases, the cumulative mission impact for the final mission is required to estimate the overall damage, which is computed using the *Max* function. We use *M* to represent the cumulative mission impact. Correspondingly, in the flow relationships including *sequence* and *message passing* as defined in Definition 1, M is computed as follows.
  - If the tasks $B_1, B_2, \ldots, B_n$ composing of the final mission B have a sequence relationship.

$$M(B) = Max(I(B_1), I(B_2), \ldots, I(B_n)) \tag{5}$$

  - If, in the tasks $B_1, B_2, \ldots, B_n$ composing of the final mission B, there are tasks, say B_2, B_3, which have exclusive decision relationship with the predecessor task B_1 and the successor tasks $B_4, \ldots, B_n$.

$$M(B) = Max(I(B_1), I(B_2), I(B_4), \ldots, I(B_n)) \quad or \quad M(B) = Max(I(B_1), I(B_3), I(B_4), \ldots, I(B_n)) \tag{6}$$

  - If, in the tasks $B_1, B_2, \ldots, B_n$ composing of the final mission B, there is message passing relationship between a task $B\prime$ from another pool to tasks in this pool.

$$M(B) = Max(I(B_1), I(B_2), \ldots, I(B_n), I(B\prime)) \tag{7}$$

III. THE CASE STUDY

This section describes our case used to determine the utility of our model. Figure 2 shows our experimental network configured to manage the customers' medical records and their health insurance policy files. Customers' medical records are stored in a MySQL database server deployed in a private cloud.

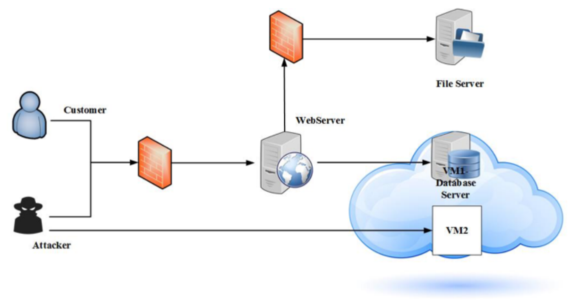

Fig. 2. The network example and corresponding attacks

Customers can query the medical records and the policy files using a web application on a webserver, using valid (username, password) pairs as an access control mechanism. We built the cloud on OpenStack, a free and open source cloud system.

We assume that an attacker's objective is to steal customers' medical records or prevent service availability. The attacker can probe deployed web and cloud services looking to find vulnerabilities that can be exploited to satisfy his/her objective. Our case study used two such vulnerabilities. The first vulnerability was the web application not sanitizing user input, named CVE-89 that created a SQL injection attack to access customers' medical records. We played the attacker role that created the SQL injection query *select * from profile where name='Alice' and (password='alice' or '1' = '1')*, where *profile* was the database name, and *'1'='1'* was the payload that made the query bypass the password check. The second vulnerability is named CVE-2015-3241 that allows authenticated users to prevent service availability by first resizing and then deleting virtual machine instances of OpenStack Compute (Nova) versions 2015.1 through 2015.1.1, 2014.2.3 and earlier. We, playing the attacker role, exploited this vulnerability as a privileged IaaS user by repeatedly resizing and deleting VM2 that co-resided in the same physical machine as the database server residing on VM1, which bypassed user quota enforcement to deplete available disk space. The three kind of graphs, including a BPD, a LEG and an ODG, generated by using our experimental example are as follows.

A. The Business Process Diagram

Figure 3 is the constructed BPD. In this BPD, there is only one pool that has start, end events and two missions *return customer records* and *return files*. The two missions are fulfilled by tasks *visit web application, request customer records, request files, verify username and password, query SQL database, query a file, data available* and *file available* that are represented by boxes. Figure 3 shows parallel fork and exclusive or gateways represented by diamonds. The exclusion or gateways have *yes* and *no* choices.

B. The Logical Evidence Graph

Table II shows evidence of the SQL injection attack including Snort(the IDS we deployed in the experimental network) alerts and database server access logs. Using timestamps, corresponding alert content and MySQL general query logs, we asserted that the attacker used a typical SQL injection with payload *'1'='1'*. Our IDS failed in capturing the DoS attack launched by exploiting the vulnerability CVE-2015-3241 in OpenStack Nova services. Because OpenStack API logs provide users' operations of running instances, we used them to conclude that the user admin (the attacker in our experiment) was trying to resize and delete the instance VM2 that co-resided in the same physical machine as the database server (VM1).

We converted the system configuration and the evidence to Prolog predicates as shown in Figure 4 and Figure 5 as input files to our LEG reconstruction tool. During the system runtime, the input files instantiated the rules representing the generic attack techniques in this tool to correlate constant predicates representing different items of evidence or system configuration, forming the LEGs as shown in Figure 6 and Figure 7 respectively (the notation can be found in Table III and Table IV). The two LEGs could not be grouped together, because the attacker's locations were different. Consider an attack step (Nodes $3, 7, 8 \rightarrow 2 \rightarrow 1$ in Figure 7) as an example. Facts of LEGs are shown in boxes (Nodes 7, 8), representing network configurations and vulnerabilities as the evidence prior to the attack step. The consequence fact node is shown in a diamond (Node 1), representing the evidence of the post attack status that is derived by applying a rule to the parent facts (Nodes 7, 8) and the parent consequence facts (Node 3 obtained from a prior attack step as the attacker's stepping-stone to the current attack step). The rule node is shown in an ellipse representing the attack and connects pre-attack system status (Nodes 3, 7, 8) and the post attack status (Node 1).

C. The Object Dependency Graph

To show how to use system call sequences to construct an ODG, we simulated an attack without triggering IDS alerts in our experimental network. We assume the attacker used a social engineering attack to obtain a legitimate user's (username, password) pair to log into the file server using ssh. In our experiment, the legitimate user's name is gmu. The corresponding server log showed that the attacker stole the user gmu's credentials. We used the right column in Table 1 to filter system calls and used dependency rules listed in the left column of Table 1 to construct an ODG as shown in Figure 8, showing the attacker modified the policy file in the file server. In this figure, the two objects *attacker* and *file in fileserver* are represented by boxes, and the rule *modify* is represented by an ellipse. Figure 8 was mapped the LEG in Figure 6, showing the attacker from the Internet could attack the file server by using stolen credentials and attack the database server by using a SQL injection attack (figure omitted due to space limitations).

D. Mission Impact Computation in Our Case Study

The impact score of each attack step in Figure 6, 7 and 8 is shown in Table V, where two impact scores of CWE-89 and CVE-2015-3241 are obtained from NIST NVD CVSS. The

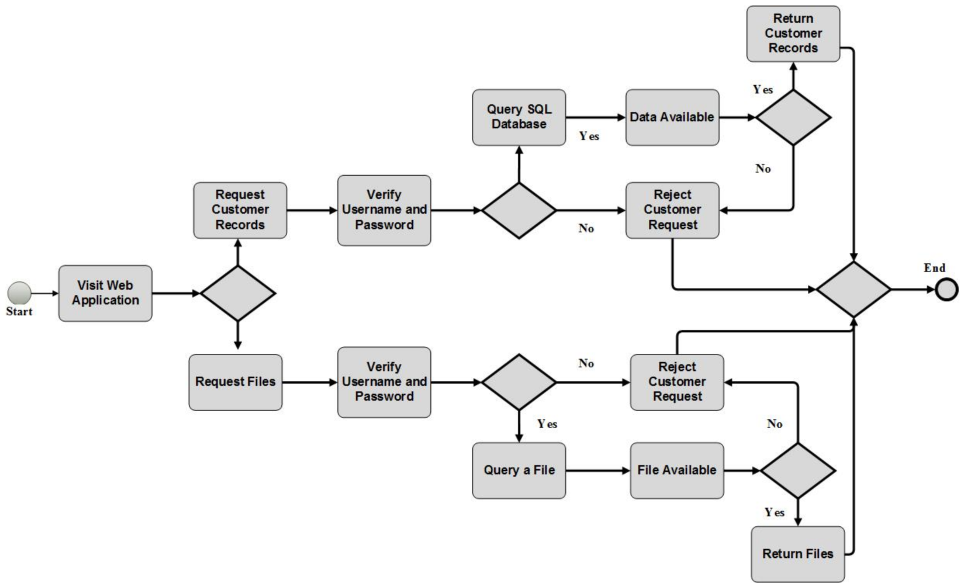

Fig. 3. The BPD of customers' retrieving their medical records and policy files

TABLE II
THE SNORT ALERT AND DATABASE SERVER LOG OF SQL INJECTION ATTACK

Time Stamp	Machine	IP Address/Port	Snort Alert and Database Server Access Log
	Attacker	129.174.124.122	
06/13-14:37:27	web server	129.174.124.184	SQL injection attack(CWE-89)
13/Jun/2017:14:37:34	Database server	129.174.124.35	Access from 129.174.124.184

TABLE III
THE NOTATION OF ALL NODES IN FIGURE 6

No.	Notation of all nodes
1	execCode(database,_)
2	RULE 2 (remote exploit of a server program)
3	netAccess(database,tcp,3306)
4	RULE 5 (multi-hop access)
5	hacl(webServer,database,tcp,3306)
6	execCode(webServer,apache)
7	RULE 2 (remote exploit of a server program)
8	netAccess(webServer,tcp,80)
9	RULE 6 (direct network access)
10	hacl(internet,webServer,tcp,80)
11	attackerLocated(internet)
12	networkServiceInfo(webServer,httpd,tcp,80,apache)
13	vulExists(webServer,'directAccess',httpd, remoteExploit,privEscalation)
14	networkServiceInfo(database,httpd,tcp,3306,_)
15	vulExists(database,'CWE-89',httpd, remoteExploit, privEscalation)

```
/* the initial attack location and final attack status*/
attackerLocated(internet).
attackGoal(execCode(database,user)).

/* the network access configuration*/
hacl(internet, webServer, tcp, 80).
hacl(webServer, database, tcp, 3306).

/* configuration information of webServer */
vulExists(webServer, 'directAccess', httpd).
vulProperty('directAccess', remoteExploit, privEscalation).
networkServiceInfo(webServer , httpd, tcp , 80 , apache).

/* the vulnerability of the web application */
vulExists(database, 'CWE-89', httpd).
vulProperty('CWE-89', remoteExploit, privEscalation).
networkServiceInfo(database , httpd, tcp , 3306, user).
```

Fig. 4. Prolog predicates for SQL injection

impact scores in NIST NVD CVSS are based on a [0, 10] scale, which we converted to a [0,1] interval scale. Because our example does not have services attackable using multiple methods, equation (1) is not used in our LEGs.

We mapped all attacks shown in Figures 6, 7 and 8 to the

```
/* the initial attack status of being an iaas user and the final attack status*/
attackerLocated(iaas).
attackGoal(execCode(nova,admin)).

/*the cloud configuration, the "_" represents any protocol and port*/
hacl(iaas,nova,_, _).

/* the vulnerability in nova */
vulExists(nova, 'CVE-2015-3241', 'REST').
vulProperty('CVE-2015-3241',remoteExploit, privEscalation).
networkServiceInfo(nova , 'REST', http, _, admin).
```

Fig. 5. Prolog predicates for DoS attack

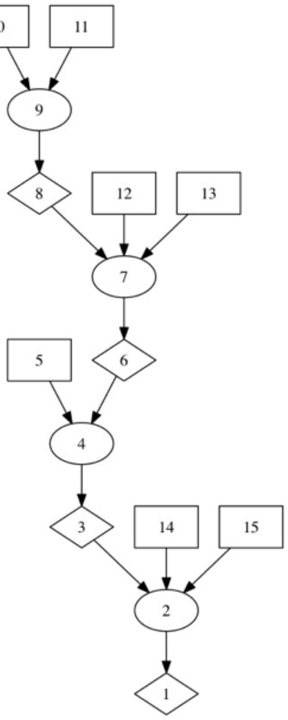

Fig. 6. The LEG of SQL injection attack toward the database

TABLE IV
THE NOTATION OF ALL NODES IN FIGURE 7

No.	Notation of all nodes
1	execCode(nova,admin)
2	RULE 2 (remote exploit of a server program)
3	netAccess(nova,http,_)
4	RULE 6 (direct network access)
5	hacl(cloud,nova,http,_)
6	attackerLocated(cloud)
7	networkServiceInfo(nova,'REST',http,_,admin)
8	vulExists(nova,'CVE-2015-3241', 'REST', remoteExploit,privEscalation)

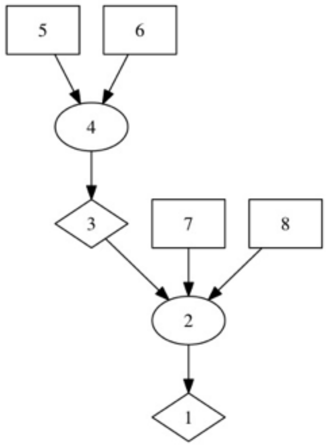

Fig. 7. The LEG of DoS attack toward the database server

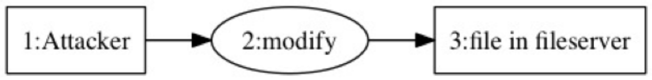

Fig. 8. The attackers using social engineering attack to modify a file in the fileserver

BPD in Figure 3, and calculated the mission impact of the three attacks as shown in Table VI, where different weights were given respectively. Based on Table VI and the BPD in Figure 3, the cumulative mission impacts were also calculated and listed in Table VII. Table VI shows that SQL attack on the task of *verifying username and password* in the database server is considered to have a higher mission impact than the DoS attack toward the database server(on the task of *data available*) and using social engineering to modify policy files in the file server(on the task of *verify username and password in the file server*). Table VII shows that the mission impact of attacks on the customers' medical records is higher than the attack on the policy files.

IV. RELATED WORK

Modern-day attackers tend to use multi-step, multi-stage attacks to impact important services protected using complex mechanism. Researchers have proposed and designed models to estimate the mission impacts of such attacks. Sun et al. proposed using a multi-layered impact evaluation model to estimate the mission impacts [10]. In this multi-layered model consisting of four layers, a lower vulnerability layer is mapped to an asset layer, and then to a service layer, which finally maps to the mission layer, where the mission impacts are calculated by using vulnerabilities' CVSS scores and the relationships between missions to the lower level assets, services and vulnerabilities. Another group of researchers, Sun et al., combined mission dependency graphs with attack graphs generated by an attack graph generation tool MulVAL [11] to estimate the attack mission impacts in the clouds [4]. Noel at el. designed a cyber-mission impact assessment framework by leveraging BPMN and their attack graph generation tool

TABLE V
THE CVSS IMPACT SCORES

Symbol Representation	Attack Step	Attack	Attack Impact
N_1	Figure 6: (3,14,15)$\rightarrow$ 2 $\rightarrow$ 1	CWE-89	0.9 (from NIST NVD)
$N_1\prime$	Figure 7: (3,7,8)$\rightarrow$ 2$\rightarrow$1	CVE-2015-3241	0.69 (from NIST NVD)
N_s	Figure 8: 1 $\rightarrow$ 2 $\rightarrow$ 3	Social Engineering	0.5 (expert knowledge)

TABLE VI
THE MISSION IMPACT SCORES

Symbol Representation	Server	Task	Weight	Mission Impact
A	Database Server	Verify Username and Password	1	$I(A) = 1 \times P(N_1) = 1 \times 0.9 = 0.9$
B	Database Server	Data Available	0.9	$I(B) = 0.9 \times P(N_1\prime) = 0.9 \times 0.69 = 0.621$
C	File Server	Verify Username and Password	1	$I(C) = 1 \times P(N_s) = 1 \times 0.5 = 0.5$

TABLE VII
THE CUMULATIVE MISSION IMPACT

Symbol Representation	Server	Mission	Cumulative Mission Impact
D	Database Server	Return Customer Records	M(D)=Max(I(A),I(B))=Max(0.9,0.621)=0.9
E	File Server	Return Files	M(E)=Max(I(C)) =Max(0.5)= 0.5

named Topological Vulnerability Analysis (TVA) [12] that combines an exploit knowledge base and a remote network scanner, analyzing all potential attack paths leading to attack goals to evaluate potential mission impacts [3], [4]. However, these approaches use vulnerabilities collected from the bug-report community such as NIST NVD to assess the impacts of attacks. These do not scale to large infrastructures or zero-day attacks.

Forensics researchers have proposed using reasoning on collected evidence from attacked infrastructures using evidence correlation rules to reconstruct the attack scenarios. The objective of this work has been to reconstruct criminal or unauthorized actions shown to be disruptive to missions [7], [13]. To reconstruct attack scenarios that have legal standing, we integrated a Prolog logic tool, MulVAL, with two databases, including a vulnerability database and an anti-forensic database, to ascertain the admissibility of evidence and explain missing evidence due to attackers' using anti-forensics [8]. We also expanded their work by using system calls to reconstruct the missing attack steps due to missing evidence in the higher application levels, and using Bayesian Network to estimate the experts' belief on the reconstructed attack scenarios [14]. However, no researchers have proposed any method to estimate the mission impacts of attacks launched toward an enterprise's infrastructure, which leaves a gap between the mission impact analysis and forensic analysis.

V. CONCLUSION

In this preliminary work we proposed a three-layered graphical model to quantify mission impacts of cyber-attacks computable using forensic techniques. We did so by reconstructing attacks based on available evidence from attack logs and system call sequences when logs did not have requisite evidence for attack steps. We used attack impact scores published in the NIST NVD CVSS and expert opinions when such numbers are unavailable. We then mapped the attacks to higher-level business processes and considered their importance weight for business processes to compute the impacts of cyber-attacks on missions.

DISCLAIMER

REFERENCES

[1] S. Musman and A. Temin, A cyber mission impact assessment tool, In Technologies for Homeland Security (HST), 2015 IEEE International Symposium on 2015 Apr 14 (pp. 1-7).
[2] NIST National Vulnerability Database Common Vulnerability Scoring System, available at https://nvd.nist.gov/vuln-metrics/cvss.
[3] S. Noel, J. Ludwig, P. Jain, D. Johnson, R. K. Thomas, J. McFarland, B. King, S. Webster and B. Tello, Analyzing mission impacts of cyber actions (AMICA), In NATO IST-128 Workshop on Cyber Attack Detection, Forensics and Attribution for Assessment of Mission Impact, Istanbul, Turkey, 2015.
[4] X. Sun, A. Singhal, P. Liu, Towards Actionable Mission Impact Assessment in the Context of Cloud Computing, In Livraga G., Zhu S. (eds) Data and Applications Security and Privacy XXXI. DBSec 2017. Lecture Notes in Computer Science, vol 10359.
[5] Online Resource for Markup Language Technologies, retrieved from http://xml.coverpages.org/bpm.html#bpmi.
[6] L. Herbert, Specification, Verification and Optimization of Business Processes, A Unified Framework, Technical University of Denmark (2014).

[7] C. Liu, A. Singhal and D. Wijesekara, A logic-based network forensic model for evidence analysis, in Advances in Digital Forensics XI, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 129-145, 2015.

[8] X. Sun, J. Dai, A. Singhal, P. Liu and J. Yen, Towards Probabilistic Identification of Zero-day Attack Paths, Accepted for IEEE Conference on Communication and Network Security, Philadelphia, October 17th 19th, 2016.

[9] C. Liu, A. Singhal and D. Wijesekera, Mapping evidence graphs to attack graphs, In Information Forensics and Security (WIFS), 2012 IEEE International Workshop on (pp. 121-126). IEEE.

[10] Y. Sun, T. Y. Wu, X. Liu, X. and M.S. Obaidat, Multilayered Impact Evaluation Model for Attacking Missions, IEEE Systems Journal, 10(4), pp.1304-1315, 2016.

[11] X. Ou, S. Govindavajhala, S. and A. W. Appel, MulVAL: A Logic-based Network Security Analyzer, In USENIX Security Symposium (pp. 8-8), July 2005.

[12] S. Jajodia and S. Noel, Topological vulnerability analysis, In Cyber situational awareness, pp. 139-154. Springer US, 2010.

[13] W. Wang, E.D. Thomas, A graph based approach toward network forensics analysis, ACM Transactions on Information and Systems Security 12 (1) 2008.

[14] C. Liu, A. Singhal and D. Wijesekera, A Probabilistic Network Forensic Model for Evidence Analysis, IFIP International Conference on Digital Forensics. Springer International Publishing, 2016.