A Layered Graphical Model for Cloud Forensic and Mission Attack Impact Analysis

Changwei Liu[1], Anoop Singhal[2], and Duminda Wijesekera[1,2]

[1]Department of Computer Science, George Mason University, Fairfax VA 22030 USA

[2]National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg MD 20899 USA

[1]{*cliu6,dwijesek*}*@gmu.edu*

[2]*anoop.singhal@nist.gov*

Abstract

Cyber-attacks on the system supporting an enterprise's mission can impact achieving its objectives. We describe a layered graphical model as an extension of a forensic investigation in order to quantify mission impacts. Our model has three layers: the upper layer models operational tasks that constitute the mission and their inter-dependencies. The middle layer reconstructs attack scenarios from available evidence to reconstruct their inter-relationships. In cases where not all evidence is available, the lower level reconstructs potentially missing attack steps. Using the three levels of graphs constructed in these steps, we present a method to compute the impacts of attack activities on missions. We use NIST's National Vulnerability Database's (NVD)-Common Vulnerability Scoring System (CVSS) scores or forensic investigators' estimates in our impact computation. We present a case study to show the utility of our model.

Keywords: Mission attack impact, Enterprise infrastructure, Cloud forensic analysis, Layered-graphical model

1 Introduction

Organizational missions are used to abstract activities envisioned by organizations, usually defined at the high-level as a collection of business processes. Cyber-attacks on an enterprise infrastructure that support such missions can impact these missions. Concurrently, a growing number of organizational business processes and services are now hosted on cloud operators' data centers. Given that most networked infrastructures including cloud services are supported by hardware and software assets, any attack that impacts these assets could impact the missions they support. Therefore, analyzing and quantifying the mission impacts of cyber-attacks is of importance to infrastructure system planners in migrating security threats and improving mission resilience, which is the primary objective in this paper.

NIST's NVD-CVSS provides impact estimates of exploitable vulnerabilities on IT systems [2]. Other publications use NIST's NVD-CVSS to predict the impacts of multi-step attacks on assets by considering constructing all possible attack paths [1, 5, 6]. However, evaluating all paths is infeasible for forensic analysis to assess damages due to the large number of paths and vulnerabilities. Additionally, quoted publications only consider the vulnerabilities reported publicly, not including zero-day attacks.

Because post-attack artifacts obtained during forensic investigations provide information that can be used to analyze attacks, we create a layered graphical model that uses this information to quantify the attacks' impacts on missions. Our model has three layers. The upper layer models operational tasks that constitute the mission and their inter-dependencies, where a mission is modeled as a collection of choreographed tasks. The middle layer collects evidence from intrusion detection system (IDS) tools and event logs to reconstruct attack scenarios. In cases where not all evidence is available, the lower layer reconstructs potentially missing attack steps using system calls that were executed to fulfill the mission. Finally, the two mapping algorithms integrate the information obtained from the three layers to ascertain how the mission execution was supported midst attack activities. Using the layers of graph-like dependency information, our model provides a method to compute impacts of attacks on missions using the NIST NVD-CVSS scores or forensic investigators' estimates on attacks toward the underlying software or hardware. We present a case study to show the utility of our model, and how it can be used to migrate attack risks in a networked infrastructure. To the best of our knowledge, there isn't an integrated forensic analysis framework that quantifies the mission impacts of multi-step attacks in a complex enterprise infrastructure which uses cloud-based services.

The rest of the paper is organized as follows. Section 2 discusses background and related work. Section 3 presents the three-layered graphical model. Section 4 uses a case study to show how the model computes mission impacts of attacks in a cloud environment, and discusses how the impact analysis can be used to enhance a networked infrastructure. Section 5 gives the conclusion.

2 Background and Related Work

We present the background and related work in this section.

2.1 Cloud Forensics

NIST defined Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) as deployment models [3]. SaaS allows clients to use providers' applications running on a cloud infrastructure. PaaS allows clients to deploy on the cloud client applications using programming languages, libraries, services and tools supported by the provider. IaaS provides clients with the capability of provisioning processing, storage, networks and other fundamental computing resources, so that the clients can deploy and run software including operating systems and applications.

Digital forensic investigators seek attack evidence from computers and networks. According to Ruan et al., cloud forensics is a subset of network forensics that follows the main phases of network forensics with techniques tailored to cloud computing environments [4]. For example, data acquisition is different in SaaS and IaaS, because the investigator will have to solely depend on cloud service providers in SaaS. Investigators can acquire the virtual machine images from IaaS customers.

2.2 Related Work

Attackers tend to use multi-step, multi-stage attacks to impact important services protected using complex mechanisms. Researchers have proposed and designed models to estimate the mission impacts of such attacks by considering all known vulnerabilities. Sun et al. proposed using a multi-layered impact evaluation model to estimate the mission impacts [10]. In this model, a lower vulnerability layer is proposed to map to an asset layer, and then to a service layer, which finally maps to the mission layer, so that the mission impacts can be calculated by using vulnerabilities' CVSS scores and the relationships between missions to the lower level vulnerabilities. However, this model does not provide a method to construct the attack paths. 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 [6]. Noel at el. designed a cyber-mission impact assessment framework by leveraging Business Process Modeling Notation (BPMN) and their attack graph generation tool 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 [5, 6]. However, these approaches use vulnerabilities collected from the bug-report community such as NIST's NVD to assess the impacts of attacks. These do not scale to large infrastructures or zero-day attacks.

Forensics researchers have reasoned on post-attack evidence using correlation rules to reconstruct

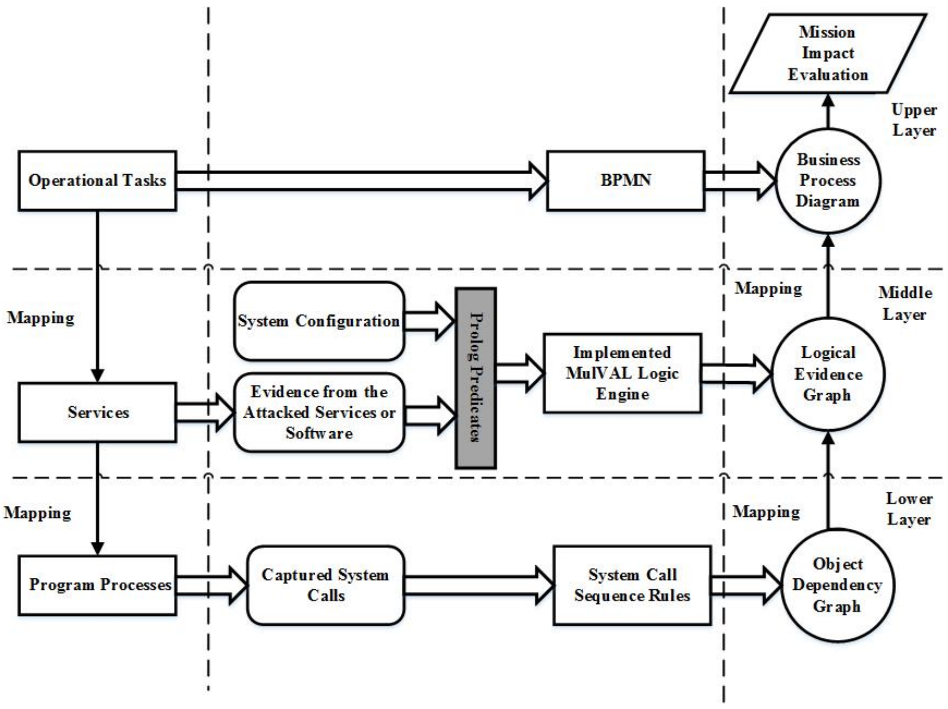


Figure 1: The three-layered graph model for mission impact evaluation

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 [9]. We also expanded this work by using system calls to reconstruct the missing attack steps due to missing evidence in the higher application level, and using Bayesian Network to estimate the experts' belief on the reconstructed attack scenarios [14]. However, no work exists to estimate the mission impacts of attacks launched toward an enterprise's infrastructure.

3 Our Three-layered Graphical Model

Figure 1 shows our model. 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.

3.1 The Upper Layer

The upper layer models tasks and missions as business processes. We model business processes using a Business Process Diagram (BPD) using the Business Process Modeling Notation (BPMN). We use tasks, events, sequencing, exclusive choice, parallel gateways, message flows and pools [16] to constuct BPDs formalized in Definition 1 (Originally defined in [16]).

Definition 1 *Business Process Diagram-BPD:*

Any quintuple $(Pool, T, E, C, OP)$ satisfying the following conditions are said to be a BPD.

- *Let T be a set of tasks and E be a set of events.*
- *Let E^{send}, E^{rec} be two disjoint subsets of E and $e_{start}, e_{end} \in E$ be two events we refer to as start and end events. Hence, $E = e_{start} \cup e_{end} \cup E^{send} \cup E^{rec}$ are respectively called start, stop, message sending, message receiving events.*
- *$T^{com} = \{(t, e_{send}, c), (t, e_{rec}, c) : t \in T, e_{send} \in E^{send}, e_{rec} \in E^{rec}, c \in C\}$ are said to be the set of communicating tasks.*
- *$OP = \{F, M, XOR, ;\}$ called parallel fork, parallel merge, exclusive choice and sequencing.*
- *C is a set of channels.*

Business processlets and a Business Process are defined as follows.

- *Any $t \in T \cup T^{com}$ is said to be a processlet.*
- *If P and Q are processlets, then so are $P; Q, F(P, Q)M$ and $P(XOR)Q$.*
- *If $P \in T \setminus T^{com}$ and e_{start}, e_{end} are start and end events, then $e_{start}; P; e_{end}$ is said to be a business process.*
- *Pools are business processes with the constrains: for two pools P_1 and P_2, there is a task (t_1, e_{send}, c) in P_1 if and only if there will be another task (t_2, e_{rec}, c) in P_2, so that the message can be passed through channel c.*

3.2 The Middle Layer

The middle layer constructs potential attacks from evidence available from system logs and 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. However, sometimes, not all evidence may be available in IDS logs, 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 items of 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 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 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) 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* $p \leftarrow p_1 \wedge p_2, \ldots, \wedge p_n$ *with* 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.*

Table 1: 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

3.3 The Lower Layer

The lower layer uses instances of interactions between services and the execution environment to obtain evidence unavailable from systems logs and IDS alerts. We obtain 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 chosen 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.

3.4 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 map attacked services in corresponding vertices in BPDs, LEGs and ODGs so that the source graphs can be mapped to the destination graphs. A LEG is easily mapped to a BPD by matching the attacked services to the corresponding tasks supported by the services. We use depth first search (DFS) method to map an ODG to a LEG, which is shown in Algorithm 1.

In Algorithm 1, all object nodes in an ODG are initially marked as *having not been checked* by using color white as shown in the *for* loop between Line 2 and 4. Then, for each given unchecked object node V_O (Line 6 and 7), the algorithm repeatedly calls $Find(V_O, LEG)$ function (calls from Line 13, and the function itself is between Line 36 and Line 50), attempting to find the matching post-attack status node in the LEG by checking if the attacked service in the LEG is equal to the attacked service in the ODG. If such a post-attack status node, say N_{c1}, is found (Line 15), then, Algorithm 1 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 (Line 19). If there is no such a matching attack step, one is added to the LEG (Line 22 to 25). If there

Algorithm 1: Mapping an ODG to a LEG

```
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.
1  //Color all nodes in ODG WHITE
2  for each node $V_O$ in ODG do
3      color[$V_O$] $\leftarrow$ WHITE
4  end
5  //Go through each object in ODG
6  for each node $V_O$ in ODG do
7      if $V_O$ == WHITE then
8          //Initialize all nodes in LEG white
9          for each node $N_c$ in LEG do
10             color[$N_c$] $\leftarrow$ WHITE
11         end
12         //Search for the corresponding $N_{c1}$ in LEG
13         $N_{c1}$ = Find($V_O$, LEG)
14         //If there is such a matching $N_{c1}$
15         if $N_{c1} \neq \emptyset$ then
16             color($V_O$) $\leftarrow$ BLACK
17             //See if the object's parent matches
18             //corresponding $N_{c1}$' s parent
19             N$_{c2}$=Find(parent($V_O$), LEG)
20             //If not matching parents,
21             //add the missing attack step from ODG to LEG
22             if $N_{c2} \neq parent(N_{C1})$ then
23                 LEG $\leftarrow$ Flow($N_{c1}, V_E$)
24                 LEG $\leftarrow$ Flow($V_E, N_{c2}$)
25             end
26         end
27         else
28             //If there is no such a matching $N_{c1}$ in LEG
29             //Add the new object to LEG
30             LEG $\leftarrow$ $V_O$
31             color [$V_O$]=GRAY
32         end
33         V$_O$ =child($V_O$)
34     end
35 end
36 Function Find($V_O$, LEG)
37     //Go through each $N_c$ in LEG
38     for each post attack status $N_c$ from LEG do
39         //Check if there is any matching $N_c$ for $V_O$
40         if ($N_c$.service == $V_O$.service AND
41         color[$N_c$] == white then
42             color[$N_c$] $\leftarrow$ BLACK
43             return $N_c$
44         end
45         else
46             color[$N_c$] $\leftarrow$ GRAY
47             $N_c$ $\leftarrow$ the child post attack status node of $N_c$
48         end
49     end
50     return $\emptyset$
```

isn't a mapping post-attack status Node N_{c1} in the LEG for Node V_O (Line 27) in the ODG, one is added to the LEG (Line 27 to 32), and the search continues (Line 33) until all nodes in the ODG are checked (i.e. colored).

3.5 Computing Mission Impact

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

3.5.1 Compute the impact scores of attacks in LEGs

In a LEG, we use $P(a)$ to represent the impact of attacks on services deployed on host computers. NIST's 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's NVD. If the attack a cannot be found in NIST's 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 attacks on the same service 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. $P(a_1) \cup P(a_2) = P(a_1) + P(a_2) - P(a_1) \times P(a_2)$.

3.5.2 Assign weight to tasks/missions

A value between [0,1] is proposed as the weight of mission impact of attacks on a task, indicating the importance of the corresponding task to the mission of a business process.

3.5.3 Compute mission attack impacts in BPDs

We map LEGs to BPDs so that the mission impact of attacks I(T) on a task T is computed using Equation 2.

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

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

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

In Equation 2, P(T) is the impact of attacks on a task T in a BPD. Depending on the mapping relationship from the attacked service(s) (represented by a, a_1, a_2) in a LEG to a task (represented by T) in a BPD, P(T) is computed by using Equation 3(one-to-one mapping relationship) and Equation 4(many-to-one mapping relationship) respectively.

3.5.4 Compute the cumulative mission impact

Mission impact of attacks on each task can be computed using Equation 2, Equation 3 and Equation 4. However, in some cases, the cumulative mission attack impact for the final mission is required to estimate the overall damage, which we compute as the maximum. We use M to represent the cumulative mission impact. Correspondingly, for the four kinds of relationships between tasks composing of a business process, M is computed using the following equations, explained below.

$$M(B) = Max\{I(T_1), I(T_2), \ldots, I(T_n)\} \quad (5)$$

$$M(B) = Max\{I(T_1), I(T_2), I(T_4) \ldots, I(T_n)\} \quad (6)$$

$$M(B) = Max\{I(T_1), I(T_3), I(T_4) \ldots, I(T_n)\} \quad (7)$$

$$M(B) = Max\{M(B_{before}), I(T_2\prime)\} \quad (8)$$

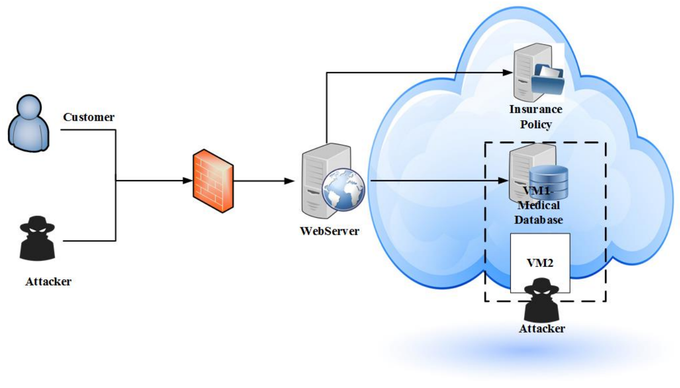
1. If the tasks $T_1, T_2, \ldots, T_n$ composing of the final mission B have a sequential relationship with each other, or among them, there are tasks, say T_2, T_3 that have a parallel fork relationships with the predecessor task T_1 and a parallel merge relationships with the successor tasks T_4, M(B) is computed as shown in Equation 5.

2. Among tasks $T_1, T_2, \ldots, T_n$ that compose of the final mission B, if there are tasks, say T_2, T_3, which have an exclusive decision relationship with the predecessor task T_1 and the successor tasks T_4, and all other tasks including $T_4, \ldots, T_n$ have a sequential relationship with each other, depending on which task (either T_2 or T_3) the business process chooses, either Equation 6 or Equation 7 is used to compute M(B).

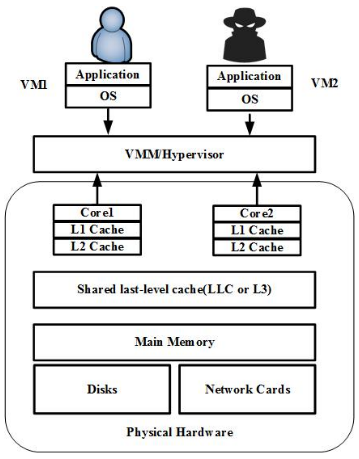
3. Suppose tasks $T_1, T_2, \ldots, T_n$ compose of the final mission B in Pool 1. We use $M(B_{before})$ to represent the cumulative mission attack impact of B without any message passing from other pools. If there is message passing relationship between a task $T_2\prime$ from Pool 2 to T_2 in Pool 1, Equation 8 is used to compute M(B).

4 The Case Study

This section describes our case study used to show the utility of our model.



(a) The Experimental Network



(b) VM1 and VM2

Figure 2: The experimental network and attacks using cloud services

4.1 The experimental network and attacks

Figure 2a shows our experimental network configured to manage the customers' medical records and their health insurance policy files. These records and files are stored on two separate VMs in a private cloud set up by using OpenStack (we used the version *Juno 2014.2.3*). OpenStack is a collection of python-based software projects that manage access to pooled storage, computing and network resources that reside in one or many machines of a cloud system [17]. These projects include *Neutron (Networking)*, *Nova (Compute)*, *Glance (Image Management)*, *Swift (Object Storage)*, *Cinder (Block Storage)* and *Keystone (Authorization and Authentication)*. OpenStack can be used to deploy *SaaS*, *PaaS* and *IaaS* cloud models, but is mostly deployed as an *IaaS* cloud. Authenticated users can access the file server to retrieve policy files using *ssh* and query the medical database stored in the database server using *MySQL* queries through a web application.

We assume that the attacker's objective is to steal customers' medical records, prevent the medical records' availability and modify the health insurance policies. By probing the deployed web and cloud services, as the attacker, we launched the following attacks.

The SQL injection attack: Because our web application did not sanitize user input, the attacker could use it to create a SQL injection attack (CWE-89) to access customers' medical records. By using the 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, we retrieved all customer medical records.

The DoS attack: According to NIST's NVD, the vulnerability *CVE-2015–3241* that is in OpenStack Compute (Nova) versions *2015.1 through 2015.1.1, 2014.2.3 and earlier* allows authenticated users to cause Denial of Services (DoS) by re-sizing and then deleting an instance (VM). The process of resizing and deleting an instance is also called an instance migration. The migration process does not terminate when an instance is deleted with *CVE-2015-3241*, so an authenticated user could bypass user quota enforcement to deplete all available disk space by repeatedly performing instance migration. By using this vulnerability, playing a privileged IaaS malicious user, we launched the DoS attack toward the database server by repeatedly re-sizing and deleting VM2 that co-resided in the same physical machine as the medical database server(VM1).

The cross-VM side-channel attack: Side-channel attacks can be used to extract fine grain information across VMs that reside on the same hypervisor [18].

In our experimental network, we simulated Yarom's cache side-channel attack [19] shown in Figure 2b on our two VMs that co-reside in the same multi-core processor (an Intel quad-core i7). In this attack, the cache shared between the victim and attacking VMs can be used to fill with data from the attacking machine. Each time when an encryption occurs, the processor evicts one or more lines of the attacker's memory from the shared cache, causing timing variation. By measuring the timing, we can obtain the information to hijack the encryption key being used with the GNU Privacy Guard (GnuPG) application in the victim VM. Because Yarom's attack uses the implementation weakness existing in GnuPG 1.x before 1.4.16 versions to obtain the information used to extract the private encryption key, we installed GnuPG 1.4.12 in our VM1 (the medical database server), and executed the attack from the VM2.

The social engineering attack: We simulated a social engineering attack toward the file server. Assuming that the attacker obtained a legitimate user's (username, password) credentials, the attacker could easily log into the file server, using the user's privilege to modify corresponding insurance policy files in the file server.

To capture attacks, in our experimental network, we deployed snort as the IDS, installed Wireshark to monitor network traffic, and configured all servers to log users' access. Also, in order to obtain evidence for those attacks that are missed by the IDS alerts and service logs, we intercepted system calls from the users' processes in the cloud.

4.2 The three levels of graphs

By using the network configuration, service deployment and captured evidence, we constructed the three levels of graphs, including a BPD, two LEGs and two ODGs, described as follows.

1. The Business Process Diagram

   Figure 3 shows our BPD with 3 pools. They are Pool 1 (Web Interface), Pool 2 (Public Cloud Service) and Pool 3 (IaaS User Service). The business process in Pool 1 is the web interface for the clients(medical customers), which is composed of start/end events, two consecutive tasks *enter username/password*, *send out request* and an exclusive gateway for tasks *review policy file* and *review medical records* that depends on the client's request. The business process in Pool 2 is composed of start/end events, a task *check user request* followed by an exclusive gateway that directs to two tasks *request policy files* and *request customer databases*, which depends on the message passed from the task *send out request* in pool 1. In each of the decision task branch, there is an exclusion decision gateway(named *file available* and *data available* respectively) followed by tasks that either send the data(*policy file* or *customer medical records* through the message passing) back to the clients or reject the customer's requests otherwise. Pool 3 is a business process used to describe IaaS user services, which is mainly composed of three tasks *encrypt data in VM1*, *resize VM2*, and *install and run program in VM2* that are the exclusive decisions of the task *check IaaS user request*. For each of the three business processes in the three pools, we consider the last task(s) before the end event as the business process mission(s).

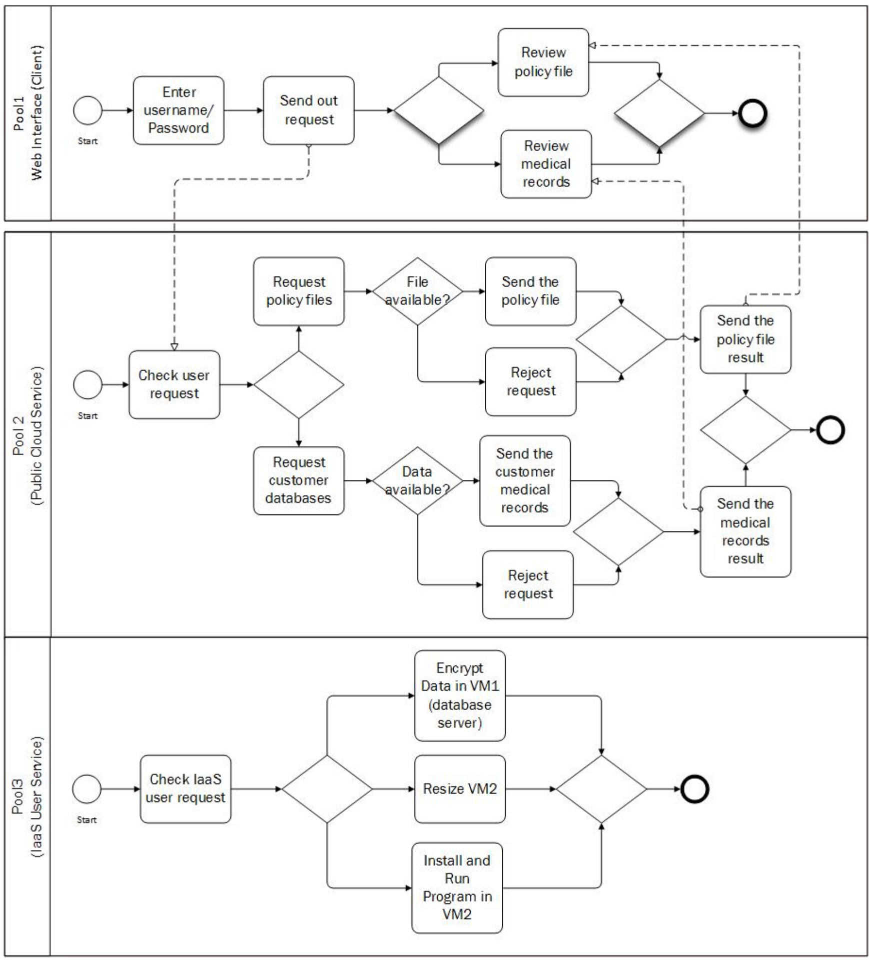


Figure 3: The BPD of the experimental network

2. The Logical Evidence Graph

Table 2 shows evidence of the SQL injection attack with Snort alerts and database 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'* to attack the customers' database

Table 2: 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

in the database server. Our IDS failed in capturing the DoS attack launched by exploiting the vulnerability CVE-2015-3241 in OpenStack Nova services. Because OpenStack application programing interface (API) logs provide users' operations of running instances (we illustrate some of these API logs in Figure 4, where we use bold font to show the users' operations), we used them to conclude that the IaaS user in VM2 (the attacker in our experiment) kept re-sizing and deleting the instance VM2 that co-resided in the same physical machine as the database server (VM1), which caused the DoS attack toward the database server.

2017-07-18 07:52:00.237 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] Running cmd (subprocess): **mv /opt/stack/data/nova/instances/bd1dac18-1c e2-44b5-93ee-967fec640ff3= /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3_resize from (pid=41737)** execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:344

2017-07-18 07:52:00.253 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] CMD **"mv /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3 /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3_resize"** returned: 0 in 0.016s from (pid=41737) execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:374

2017-07-18 07:52:00.254 DEBUG oslo_concurrency.processutils [req-f79c7911-04ed-4a0c-adbe-0ae0a487c0f7 admin admin] Running cmd (subprocess): **mkdir –p /opt/stack/data/nova/instances/bd1dac18-1ce2-44b5-93ee-967fec640ff3 from (pid=41737)** execute /usr/local/lib/python2.7/dist-packages/oslo_concurrency/processutils.py:344

Figure 4: OpenStack Nova API call logs

```
/* 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).
```

Figure 5: Prolog predicates for SQL injection

In order to use the forensic analysis tool mentioned in Section 3.2, we converted system configu-

```
/* 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).
```

Figure 6: Prolog predicates for DoS attack

rations and the evidence for the SQL injection attack and DoS attack to Prolog predicates shown in Figures 5, 6. The output LEGs produced by the tool are shown in Figure 7 and Figure 8 respectively with node names in Tables 3, 4. The two LEGs are not grouped together due to distinct attacker locations and privileges. Consider an example attack step (Nodes $3, 7, 8 \rightarrow 2 \rightarrow 1$ in Figure 8). Facts of LEGs are shown in Nodes 7, 8, modeling pre-attack configurations and vulnerabilities. The consequence fact node (Node 1) shows post-attack evidence derived by applying a rule (an ellipse Node 2 connecting Nodes 3, 7, 8 to the post attack status, Node 1) to the parent facts (Nodes 7, 8) and parent consequence fact (Node 3 that is obtained from a prior stepping stone step).

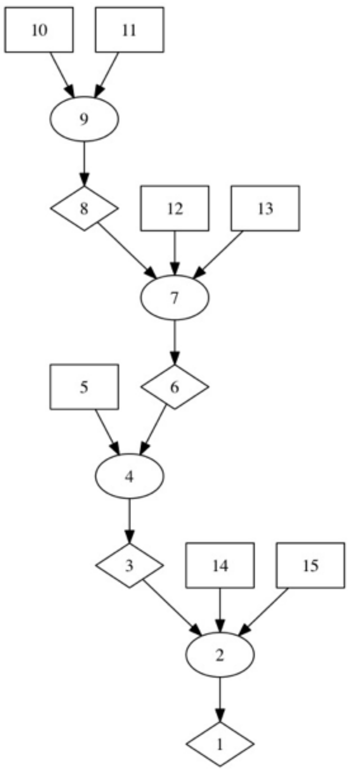


Figure 7: The LEG of SQL injection attack toward the database

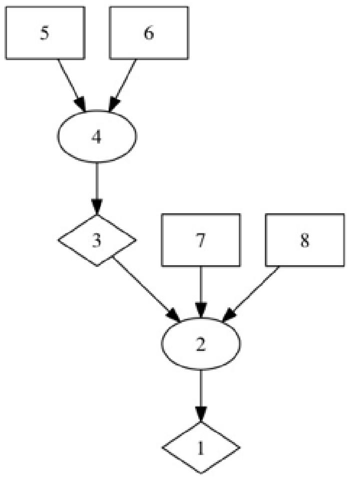


Figure 8: The LEG of DoS attack toward the database server

Table 3: Names of nodes in Figure 7

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)

Table 4: Names of nodes in Figure 8

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)

3. The Object Dependency Graph

Due to the lack of IDS alerts and logs for the side-channel and social engineering attacks, we could not reconstruct the two attack scenarios in the form of LEGs as we did for the SQL injection and DoS attacks. Therefore, we turned to the corresponding system calls we captured during the attacks and used the method mentioned in Section 3.3 to construct ODGs for a forensic analysis.

Figure 9 and Figure 10 are a fraction of the system calls captured from VM1 (the database server) and VM2 (the attacker's VM) during the side-channel attack. The system call in Figure 9 shows that a file from VM1 (named *message.txt*) was encrypted by using GnuPG 1.4.12. System calls in Figure 10 show that a probe program *bin/probe* (Line 1) in VM2 used *mmap2* (Line 3) to force the underlying system to share memory addresses (Lines 5, 6, 7...) with the probing process; hence the probing process could read data from the shared memory addresses between VM1 and VM2 for a later malicious analysis (Lines 10, 11, 12 and Line 1 show the data was written to a file named *out.txt*, and our continuous captured system calls show, later, a Python program was used to extract and analyze the information in *out.txt*). Figure 11 is a fraction of the system calls captured from the file server, where the *read/write* system call trace shows that the *test.txt* in *FileServer* has been modified. Because the corresponding sshd log in the *FileServer* recorded the users' access, it was easy to judge that the attacker stole a legitimate user's credentials to modify the policy file (the sshd log is omitted).

```
execve("/home/flush-reload-master/build_gpg/gnupg-1.4.12/bin/gpg", ["/home/flush-reload-"..., "--yes", "-
-sign", "message.txt"], [/* 61 vars */]) = 0
brk(0)                          = 0x942b000
……
```

Figure 9: Filtered system calls of the side-channel attack from VM1

```
1.  execve("bin/probe", ["bin/probe", "/home/flush-reload-"..., "docs/addr/osx.txt", "out.txt", "5"], [/*
    61 vars */]) = 0
    …
2.  fstat64(4, {st_mode=S_IFREG|0664, st_size=78, ...}) = 0
3.  mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
    1, 0) = 0xb7702000
4.  read(4, "4\n0x00000000000967c7\n0x000000000"..., 4096) = 78
5.  write(1, "Probing 4 addresses:\n", 21)  = 21
6.  write(1, "0x967c7\n", 8)             = 8
7.  write(1, "0x95f5d\n", 8)             = 8
    …
8.  write(1, "Started spying\n", 15)       = 15
    …
9.  mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
    1, 0) = 0xb7701000
10. write(5, "0 0 22690\n0 1 460\n0 2 1159\n0 3 6"..., 4096) = 4096
11. write(5, " 1 217\n113 2 223\n113 3 214\n114 0"..., 4096) = 4096
12. write(5, "520\n215 3 232\n216 0 490\n216 1 26"..., 4096) = 4096
    …
```

Figure 10: Filtered system calls of the side-channel attack from VM2

Using the dependency rules listed in the left column of Table 1 and corresponding analysis on the system calls as shown in Figures 9, 10 and 11, we constructed two ODGs, showing the attacker from VM2 read the shared cache between VM1 and VM2, and the attacker from Internet used a legitimate user's credentials to access the file server and modified a policy file. We mapped both ODGs to the LEGs in Figure 7 and Figure 8. The integrated LEGs show that: (1) the attacker from the Internet launched two attacks including using stolen credentials to modify a policy file and stealing all customers' medical records by using a SQL injection attack; (2) the attacker who was an IaaS user launched two other attacks including a DoS attack and a side-channel attack to the database server.

```
write(9, “v”, 1) = 1
read(11, “v”, 16384) = 1
write(3, “\0\0\0\20\331\255\275\264c\2173)z2j\32\255n\2007d\366m\21\316
\2648\240\207\31\211” . . . , 36) = 36
read(3,“\0\0\0\20\240\253\341\227\321xU\305\347\226\246\361\316\242S                =
\30\341QT\231\n\343\314\343\307\f\361”. . . , 16384) = 36
write(9, “i”, 1) = 1
read(11, “i”, 16384) = 1
write(3,“\0\0\0\20\177\352\313\332\373yjM\3416l\230\215\10\220p\252g\375\365
\1\f\335\361\r\273\374\357”. . . , 36) = 36
read(3,“\0\0\0\20\27\334?\201x\300\16\356\346, \0379\32\220{\372)\366\4\v\1      =
\347\263\311\250k\353” . . . , 16384) = 36
write(9, “ ”, 1) = 1
read(11, “ ”, 16384) = 1
write(3,“\0\0\0\20′i\321\344\220\313\322\254S\252o\201\225; 6v\243\205\10gŝ
\253\237\325\375\332v” . . . , 36) = 36
read(3, “\0\0\0\20\5\27k; \254\301\24\n\\ZN\267\260\336\323′\323\32\345\2b\
226 − \271|[B\21” . . . , 16384) = 36
write(9, “t”, 1) = 1
read(11, “t”, 16384) = 1
read(3,“\0\0\0\20\325\261\7\254\211(\201\331\272\344[\355\200\\u4\357G\347
\232\276 : \201\376\342\202\201.” . . . , 16384) = 36
write(3,“\0\0\0\20\320\254\#\312\211_\3022\n\227u\16I\372\202\347\37\252T
\257\220
\210E\343\222\342\24S” . . . , 36) = 36
write(9, “e”, 1) = 1
read(11, “e”, 16384) = 1
...
write(9, “\t”, 1) = 1
read(11, “st.txt ”, 16384) = 7
. . .
```

Figure 11: Filtered system calls of modifying a file from the file server

Table 5: The CVSS impact scores

Attack Name	CVE Entry	Symbol Representation	Attack Impact
SQL injection	CWE-89	N_1	0.9
DoS attack	CVE-2015-3241	$N_1\prime$	0.69
Social Engineering		N_s	0.5
side-channel attack	CVE-2013-4576	N_{sc}	0.29

4.3 Mission impact computation in our case study

The impact score of each attack step in all LEGs and ODGs is shown in Table 5, where the three impact scores of *CWE-89*, *CVE-2015-3241*, *CVE-2013-4576* were obtained from NIST’s NVD-CVSS and the impact score of the social engineering attack was from our expert knowledge. The impact scores in NIST’s NVD-CVSS are based on a [0, 10] scale, which we converted to a [0,1] interval scale.

We mapped all attacks from LEGs and ODGs to the BPD in Figure 3 and computed the mission impacts of the four attacks as shown in Table 6. Based on Table 6 and the BPD in Figure 3, the cumulative mission impacts are computed and listed in Table 7.

Table 6: The mission impact scores

Pool	**Task**	**Mapping Attack**	**Weight**	**Mission Impact**
Pool 1	Check username and password	CWE-89	1	$I_1 = 1 \times P(N_1) = 1 \times 0.9 = 0.9$
Pool 2	Data available	CVE-2015-3241	0.9	$I_2 = 0.9 \times P(N_1\prime) = 0.9 \times 0.69 = 0.621$
Pool 2	Request policy files	Social Engineering	1	$I_3 = 1 \times P(N_s) = 1 \times 0.5 = 0.5$
Pool 3	Encrypt data in VM1	CVE-2013-4576	1	$I_4 = 1 \times P(N_{sc}) = 1 \times 0.29 = 0.29$

Table 7: The cumulative mission impact

Pool	**Mission**	**Cumulative Mission Impact**
Pool 1	Review policy file	M =Max(I_3) $= Max(0.5) = 0.5$
Pool 1	Review medical records	M = Max(I_1, I_2) $= Max(0.9, 0.621) = 0.9$
Pool 3	Encrypt data in VM1	M = Max(I_4) $= Max(0.29) = 0.29$

4.4 Using mission impacts to reduce attack risks

In a complex infrastructure, different missions use connections and combinations of multiple services. Each service is supported by software and hardware assets that are usually the target of attackers. In such cases, a tool can determine the impacts of cyber-attacks on the missions. By correlating the attacks on lower level assets to the higher level business process diagram and using mission impact scores of those attacks provided by NIST's NVD-CVSS, our model shows attacks and computes their impacts on complex missions. The information provided by such an analysis can be used by forensic investigators and infrastructure system planners.

As an example, we show how the mission impacts can be used by the system planners to enhance our experimental network. First, the cumulative impact scores in Table 7 show that the attacks on the mission of *review medical records* have a higher impact because the customers' medical records could be stolen by using a SQL injection attack. This suggests the user input sanitization should be implemented to defeat SQL injections. Second, the attacks and their impact scores shown in Table 6 show the two attacks (a DoS attack and a side-channel attack) were caused by cloud services that have vulnerabilities; hence the corresponding services should be moved to a more stable cloud that has countermeasures to attacks that can be launched through the shared hypervisor or host. Third, Table 6 shows, though the mission impact on the insurance policy stored in the file sever might not be as bad as the SQL injection attack on the database server, it has a score "0.5" that can not be neglected. An easy solution would be limiting the file *write/modify* right to only administrators with local access.

5 Conclusion

We proposed a three-layered graphical model to quantify mission impacts of cyber-attacks in this work. We did so by reconstructing attacks based on available evidence from attack logs and system call sequences when logs missed requisite evidence to reconstruct attack steps. We used impact scores published in the NIST's NVD-CVSS and expert opinions when such numbers are unavailable as a base line to estimate attack impacts. 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. Our case study showed that this model can be used to mitigate the impacts of cyber-attacks in a network. In the future, we will conduct more experiments using attacks on the cloud infrastructure to determine how our framework should be used in order to help enterprises to reduce the security risks of their infrastructures.

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] P. Mell and T. Grance. "NIST definition of cloud computing". National Institute of Standards and Technology. October 7, 2009.

[4] K. Ruan, J. Carthy, T. Kechadi, and M. Crosbie. "Cloud forensics". In IFIP International Conference on Digital Forensics, pp. 35-46. Springer Berlin Heidelberg, 2011.

[5] 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.

[6] 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.

[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.

[15] Online Resource for Markup Language Technologies, retrieved from http://xml.coverpages.org/bpm.html#bpmi.

[16] L. Herbert, Specification, Verification and Optimization of Business Processes, A Unified Framework, Technical University of Denmark (2014).

[17] OpenStack Open Source Cloud Computing Software. Retrieved from https://www.openstack.org.

[18] Y. Zhang, A. Juels, M. K. Reiter and T. Ristenpart, Cross-vm side channels and their use to extract private keys, In Proceedings of the 2012 ACM Conference on Computer and Communications Security (New York, NY, USA, 2012), CCS '12, ACM, pp. 305–316.

[19] Yarom, Yuval, and Katrina Falkner. "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack." In USENIX Security Symposium, pp. 719-732. 2014.