Network Intrusion Detection and Counter Measure Selection in Virtual Network Systems

Mandeep Singh[1], Neetu Sharma[2]
[1,2] Department of Computer Science &Engineering,
Ganga Institute of Technology and Management,
Kablana, Jhajjar, Haryana, India

***Abstract* — Network Intrusion detection and Counter measure is a multiphase distributed network intrusion detection and prevention framework in a virtual networking environment that captures and inspects suspicious cloud traffic without interrupting users applications and cloud service. Cloud security is one of most important issues that has attracted a lot of research and development effort in past few years. The proposed solution utilizes a new network control approach called SDN, where networking functions can be programmed through software switch and OpenFlow protocol. NICE is a multiphase distributed network intrusion detection and prevention framework in a virtual networking environment that capture and inspects suspicious cloud traffic without interrupting users applications and cloud service. It employs a reconfigurable virtual networking approach to detect and counter the attempts to compromise VMs, thus preventing Zombie VMs. It optimizes the implementation on cloud servers to minimize resource consumption and consumes less computational overhead compared to proxy-based network intrusion detection solution.**

Abbreviations

1. QG - Queue Graph
2. DG - Dependency Graph
3. ACT - Attack Countermeasure Tree
4. BDD - Binary Decision Diagram
5. BAG - Bayesian Attack Graph
6. IaaS - Infrastructure as a Service
7. SAG - Scenario Attach Graph
8. ACG - Alert Correlation Graph
9. NIDS - Network Intrusion Detection System

I. INTRODUCTION

The area of detecting malicious behaviour has been well explored in the following approaches. *SPOT* focuses on the detection of compromised machines that have been recruited to serve as spam zombies. It is based on sequentially scanning outgoing messages while employing a statistical method *Sequential Probability Ratio Test (SPRT),* to quickly determine whether a host has been compromised. Bot Hunter detects compromised machines based on the fact that a thorough malware infection process has a number of well-defined stages that allow correlating the intrusion alarms triggered by inbound traffic with resulting outgoing communication patterns. Bot Sniffer exploits uniform spatial-temporal behavior characteristics of compromised machines to detect zombies by grouping flows according to server connections and searching for similar behavior in the flow. An *attack graph* is used to represent a series of exploits, called atomic attacks, that lead to an undesirable state. There are many automation tools to construct attack graph. A technique based on a modified symbolic model checking Nu SMV and *Binary Decision Diagrams* (BDDs) was pro-posed to construct attack graph. This model can generate all possible attack paths, but, the scalability is a big issue for this solution. The assumption of monotonicity was introduced, which states that the precondition of a given exploit is never invalidated by the successful pplication of another exploit, ie. attackers never need to backtrack. With this assumption, a concise, scalable graph representation for encoding attack tree can be obtained. An attack graph tool called Mul VAL, adopts a logic programming approach and uses Datalog language to model and analyze network system. The attack graph in the Mul VAL is constructed by accumulating true facts of the monitored network system. The attack graph construction process will terminate efficiently because the number of facts is polynomial in system.

The major problems for any IDS implementation are the false alarms and the large volume of raw alerts from IDS. Many attack graph-based alert correlation techniques have been proposed. An in memory structure, called queue graph (QG), was devised to trace alerts matching each exploit in the attack graph. However, the implicit correlations in this design make it difficult to use the correlated alerts in the graph for analysis of similar attack scenarios. A modified attack-graph-based correlation algorithm was proposed to create explicit correlations only by matching alerts to specific exploitation nodes in the attack graph with multiple mapping functions, and devised an alert dependencies graph (DG) to group related alerts with multiple correlation criteria. However, this algorithm involved all pairs shortest path searching and sorting in DG, which consumes considerable computing power. Several solutions have been proposed to select optimal counter measures based on the likelihood of the attack path and cost benefit analysis. An *attack countermeasure tree* (ACT) was proposed to consider attacks and countermeasures together in an attack tree structure. Here several objective functions based on greedy and branch and bound techniques were devised to minimize the number of countermeasure, reduce investment cost, and maximize the benefit from implementing a certain countermeasure set. In this design, each countermeasure optimization problem could be solved with and without probability assignments to the model. However, this solution focuses on a static attack scenario and predefined countermeasure for each attack. Another attack graph, *Bayesian attack graph* (BAG) was proposed to address

dynamic security risk management problem and applied a genetic algorithm to solve countermeasure optimization problem.

II. MODELS FOR COUNTERMEASURES

Threat Model : This protection model focuses on virtual network based attack detection and reconfiguration solutions to improve the resiliency to zombie explorations. The proposed solution can be deployed in an IaaS cloud networking system. Cloud service users are free to install whatever operating systems or applications they want, even if such action may introduce vulnerabilities to their controlled VMs.

Attack Graph Model : An attack graph is a modeling tool to illustrate all possible multistage, multihost attack paths that are crucial to understand threats and then to decide appropriate countermeasures. In an attack graph, each node represents either precondition or consequence of an exploit. Attack graph is helpful in identifying potential threats, possible attacks, and known *vulnerabilities* in a cloud system. If an event is recognized as a potential attack, specific countermeasures can be applied to mitigate its impact or take actions to prevent it from contaminating the cloud system. To represent the attack and the result of such actions, the notation of MulVAL logic attack graph is extended and is defined as *SAG*. An SAG is a tuple SAG = (V, E), where

• $V= N_C \cup N_D \cup N_R$ denotes a set of vertices that include three types namely

i. conjunction nodeN_C to represent exploit,
ii. disjunction node N_D to denote result of exploit,
iii. and root node N_R for showing initial step of an attack scenario.

• E = Epre ∪ Epost denotes the set of directed edges. An edge $e \in Epre \subseteq N_D \times N_C$ represents that N_D must be satisfied to achieve N_C. An edge $e \in Epost \subseteq N_C \times N_D$ means that the consequence shown by N_D can be obtained if N_C is satisfied. Node $vc \in N_C$ is defined as a three tuple *(Hosts, vul, alert*) representing a set of IP addresses, vulnerability information such as CVE , and alerts related to vc, respectively. N_D behaves like a logical OR operation and contains details of the results of actions. N_R represents the root node of the SAG. A new *Alert Correlation Graph (ACG)* is defined to map alerts in ACG to their respective nodes in SAG. To keep track of attack progress, the source and destination IP addresses are tracked for attack activities. An ACG is a three tuple *ACG = (A,E,P*), where

i. A is a set of aggregated alerts. An alert $a \in A$ is a data structure *(src,dst,cls,ts)* representing source IP address, destination IPaddress, type of the alert, and time stamp of the alert respectively.
ii. Each alert *a* maps to a pair of vertices (v_c,v_d) in SAG using map(a) function, ie. $map(a) : a \rightarrow \{(v_c,v_d)| (a.src \in v_c.Hosts) \wedge (a.dst \in v_d.Hosts) \wedge (a.cls=v_c.vul)\}$
iii. E is a set of directed edges representing correlation between two alerts (a,a') if criteria below satisfied:
   a. $(a.ts < a'.ts) \wedge (a'.ts - a.ts<threshold)$.
   b. $\exists (v_d,v_d) \in E_{pre} : (a.dst \in v_d.Hosts \wedge a'.src \in v_c.Hosts)$.

P is set of paths in ACG. A path Si | P is a set of related alerts in chronological order.

A contains aggregated alerts rather than raw alerts. Raw alerts having same source and destination IP addresses, attack type, and time stamp within a specified window are aggregated as *Meta Alerts*. Each ordered pair (a, a') in ACG maps to two neighbor vertices in SAG with time stamp difference of two alerts within a predefined threshold. ACG shows dependency of alerts in chronological order and related alerts are found in the same attack scenario by searching the alert path in ACG. A set P is used to store all paths from root alert to the target alert in the SAG, and each path Si | P represents alerts that belong to the same attack scenario. Alert Correlation algorithm is followed for every alert detected and returns one or more paths Si. For every alert ac that is received from the IDS, it is added to ACG if it does not exist. For this new alert ac, the corresponding vertex in the SAG is found by using function map(ac). For this vertex in SAG, alert related to its parent vertex of type NC is then correlated with the current alert ac. This creates a new set of alerts that belong to a path Si in ACG or splits out a new path Si+1 from Si with subset of Si before the alert a and appends ac to Si+1. In the end of this algorithm, the ID of ac will be added to alert attribute of the vertex in SAG. Algorithm 1 returns a set of attack paths S in ACG.

Algorithm 1. Alert Correlation

```
Require: alert ac, SAG, ACG
if (ac is a new alert) then
   create node ac in ACG
   n1 ← vc ∈ map(ac)
   for all n2 ∈ parent(n1) do
      create edge (n2.alert,ac)
      for all Si containing a do
         if a is the last element in Si then
            append ac to Si
         else
            create path Si+1 = { subset(Si,a),ac}
         end if
      end for
      add ac to n1.alert
   end for
end if
return S
```

VM Protection Model :The VM protection model of NICE consists of a VM profiler, a security indexer, and a state monitor. Security index is specified for all the VMs in the network based on various factors like connectivity, the number of vulnerabilities present and their impact scores. The impact score of a vulnerability helps to judge the confidentiality, integrity, and availability impact of the vulnerability being exploited. Connectivity metric of a VM is decided by evaluating incoming and outgoing connections. VM states can be defined as follows:

i. *Stable*: There does not exist any known vulnerability on the VM.

ii. *Vulnerable*: Presence of one or more vulnerabilities on a VM, which remains unexploited.
iii. *Exploited*: At least one vulnerability has been exploited and the VM is compromised.
iv. *Zombie*: VM is under control of attacker.

III. System Design and Implementation

The NICE framework is illustrated in Fig. 1. Major components in this framework are distributed and light-weighted NICE-A on each physical cloud server, a network controller, a VM profiling server, and an attack analyzer. The latter three components are located in a centralized control center connected to software switches on each cloud server.

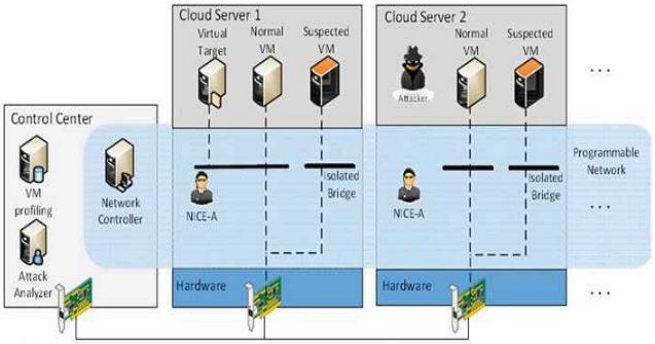

Fig. 1. NICE architecture within one cloud server cluster.

NICE-A is a software agent implemented in each cloud server connected to the control center through a dedicated and isolated secure channel, which is separated from the normal data packets using *OpenFlow tunneling* or *VLAN* approaches. The network controller is responsible for deploying attack countermeasures based on decisions made by the attack analyzer. Intrusion detection alerts are sent to control center when suspicious or anomalous traffic is detected. After receiving an alert, attack analyzer evaluates the severity of the alert based on the attack graph, decides what countermeasure strategies to take, and then initiates it through the network controller. An attack graph is established according to the vulnerability information derived from both offline and real-time vulnerability scans. Offline scanning can be done by running penetration tests and online real-time vulnerability scanning can be triggered by the network controller or when new alerts are generated by the NICE-A. Once new vulnerabilities are discovered or countermeasures are deployed, the attack graph will be reconstructed. Countermeasures are initiated by the attack analyzer based on the evaluation results from the cost benefit analysis of the effectiveness of countermeasures. Then, the network controller initiates countermeasure actions by reconfiguring virtual or physical OFSs. The components of NICE are as follows:

NICE-A : The NICE-A is a *Network-based Intrusion Detection System (NIDS)* agent installed in either Dom0 or DomU in each cloud server. It scans the traffic going through Linux bridges that control all the traffic among VMs and in/out from the physical cloud servers. Here Snort is used to implement NICE-A in Dom0. It will sniff a mirroring port on each virtual bridge in the Open vSwitch (OVS). Each bridge forms an isolated subnet in the virtual network and connects to all related VMs. The traffic generated from the VMs on the mirrored software bridge will be mirrored to a specific port on a specific bridge using SPAN, RSPAN, or ERSPAN methods. Dom0 in the Xen environment is a privilege domain, that includes a virtual switch for traffic switching among VMs and network drivers for physical network interface of the cloud server. It is more efficient to scan the traffic in Dom0 because all traffic in the cloud server needs go through it. The alert detection quality of NICE-A depends on the implementation of NICE-A, which uses Snort. The individual alert detections false alarm rate does not change. However, the false alarm rate could be reduced through this architecture design.

VM Profiling : Virtual machines in the cloud can be profiled to get precise information about their state, services
running, open ports, and so on. Any VM that is connected to more number of machines is more crucial than the one connected to fewer VMs because the effect of compromise of a highly connected VM can cause more damage. An attacker can use port-scanning program to perform an intense
examination of the network to look for open ports on any VM. So information about any open ports on a VM and the history of opened ports plays a significant role in determining how vulnerable the VM is. All these factors combined will form the VM profile. VM profiles are maintained in a database and contain comprehensive information about vulnerabilities, alert, and traffic. The data
comes from:

• *Attack graph generator*. While generating the attack graph, every detected vulnerability is added to its corresponding VM entry in the database.
• *NICE-A*. The alert involving the VM will be recorded in the VM profile database.
• *Network controller*. The traffic patterns involving the VM are based on five tuples *(source MAC address, destination MAC address, source IP address, destination IP address, protocol)*. There can be traffic pattern, where packets emanate from a single IP and are delivered to multiple destination IP addresses, and vice versa.

Attack Analyzer : The major functions of NICE system are performed by attack analyzer, which includes procedures such as attack graph construction and update, alert correlation, and countermeasure selection. The process of constructing and utilizing the SAG consists of three phases: Information gathering, attack graph construction, and potential exploit path analysis. With this information, attack paths can be modeled using SAG. Each node in the attack graph represents an exploit by the attacker. Each path from an initial node to a goal node represents a successful attack. NICE attack graph is constructed based on the following information:

• Cloud system information is collected from the node controller (i.e., Dom0 in XenServer).The information includes the number of VMs in the cloud server, running

services on each VM, and VMs Virtual Interface (VIF) information.

• Virtual network topology and configuration information is collected from the network controller, which includes virtual network topology, host connectivity, VM connectivity, every
VMs IP address, MAC address, port information, and traffic flow information.

• Vulnerability information is generated by both on demand vulnerability scanning (i.e., initiated by the network controller and NICE-A) and regular penetration testing using the well-known vulnerability databases. The attack analyzer also handles alert correlation and analysis operations. This component has two major functions: 1) constructs ACG, and 2) provides threat information and appropriate countermeasures to network controller for virtual network reconfiguration. After receiving an alert from NICE-A, alert analyzer matches the alert in the ACG. If the alert already exists in the graph and it is a known attack, the attack analyzer performs countermeasure selection procedure according to the Countermeasure Selection algorithm, and then notifies network controller immediately to deploy countermeasure or mitigation actions. If the alert is new, attack analyzer will perform alert correlation and analysis according to Algorithm 1, and updates ACG and SAG. This algorithm correlates each new alert to a matching alert correlation set (i.e., in the same attack scenario). A selected countermeasure is applied by the network controller based on the severity of evaluation results. If the alert is a new vulnerability and is not present in the NICE attack graph, the attack analyzer adds it to attack graph and then reconstructs it.

Network Controller : The network controller is a key component to support the programmable networking capability to realize the virtual network reconfiguration feature based on OpenFlow protocol. The network controller is responsible for collecting network information of current OpenFlow network and provides input to the attack analyzer to construct attack graphs. Through the cloud internal discovery modules that use DNS, DHCP, LLDP, and flow initiations, network controller is able to discover the network connectivity information from OVS and OFS. This information includes current data paths on each switch and detailed flow information associated with these paths, such as TCP/IP and MAC header. The network flow and topology change information will be automatically sent to the controller and then delivered to attack analyzer to reconstruct attack graphs. Another important function of the network controller is to assist the attack analyzer module. According to the OpenFlow protocol, when the controller receives the first packet of a flow, it holds the packet and
checks the flow table for complying traffic policies. In NICE, the network control also consults with the attack analyzer for the flow access control by setting up the filtering rules on the corresponding OVS and OFS. Once a traffic flow is admitted, the following packets of the flow are not handled by the network controller, but monitored by the
NICE-A. Network controller is also responsible for applying the countermeasure from attack analyzer. Based on VM Security Index (VSI) and severity of an alert, countermeasures are selected by NICE and executed by the network controller. If a severe alert is triggered and identifies some known attacks, or a VM is detected as a zombie, the network controller will block the VM immediately. An alert with medium threat level is triggered by a suspicious compromised VM. Countermeasure in such case is to put the suspicious VM with exploited state into quarantine mode and redirect all its flows to NICE-A DPI mode. An alert with a minor threat level can be generated due to the presence of a vulnerable VM. For this case, to intercept the VMs normal traffic, suspicious traffic to/from the VM will be put into inspection mode, in which actions such as restricting its flow bandwidth and changing network configurations will be taken to force the attack exploration behavior to stand out.

IV. COUNTERMEASURE SELECTION

When vulnerabilities are discovered or some VMs are identified as suspicious, several countermeasures can be taken to restrict attackers capabilities and it is important to differentiate between compromised and suspicious VMs. The countermeasure serves the purpose of: 1) protecting the target VMs from being compromised, and 2) making attack behavior stand prominent so that the attackers actions can be identified.

Security Measurement Metrics : The issue of security metrics has attracted much attention and there has been significant effort in the development of quantitative security metrics in recent years. Among different approaches, using attack graph as the security metric model for the evaluation of security risks is a good choice. To assess the network security risk condition for the current network configuration, security metrics are needed in the attack graph to measure risk likelihood. After an attack graph is constructed, vulnerability information is included in the graph. For the initial node or external node (i.e., the root of the graph, $N_R \subseteq N_D$), the priori probability is assigned on the likelihood of a threat source becoming active and the difficulty of the vulnerability to be exploited. GV is used to denote the *priori risk probability* for the root node of the graph and usually the value of GV is assigned to a high probability, e.g., from 0.7 to 1. For the internal exploitation node, each attack-step node
$e \in N_C$ will have a probability of vulnerability exploitation denoted as $G_M[e]$. $G_M[e]$ is assigned according to the Base Score (BS) from Common Vulnerability Scoring System (CVSS). The BS is calculated by the impact and exploitability factor of the vulnerability. BS can be directly obtained from National Vulnerability Database by searching for the vulnerability CVE id

$$BS = (0.6 \times IV + 0.4 \times E - 1.5) \times f(IV),$$

where

$$IV = 10.41 \times (1 - (1 - C) \times (1 - I) \times (1 - A)),$$
$$E = 20 \times AC \times AU \times AV ,$$

and

f(IV)= 0 if IV = 0,
1.176 otherwise.

The impact value (IV) is computed from three basic parameters of security namely *confidentiality* (C), *integrity* (I), and *availability* (A). The *exploitability* (E) score consists of *access vector* (AV), *access complexity* (AC), and *authentication instances* (AU). The value of BS ranges from 0 to 10. In the attack graph, each internal node is assigned with its BS value divided by 10, as shown in

$$GM=Pr(e=T)=BS(e)/10, \forall e \in N_C$$

In the attack graph, the relations between exploits can be disjunctive or conjunctive according to how they are related through their dependency conditions. Such relationships can be represented as conditional probability, where the risk probability of current node is determined by the relationship with its predecessors and their risk probabilities. Given below are the probability derivation relations:

- for any attack-step node $n \in NC$ with immediate predecessors set W = parent(n),

$$P_r(n \mid W) = G_M[n] \times \prod_{s \varepsilon W} P_r(s \mid W);$$

- for any privilege node $n \in N_D$ with immediate predecessors set W= parent(n), and then

$$P_r(n \mid W) = 1 - \prod_{s \varepsilon W}(1 - P_r(s \mid W)).$$

Once conditional probabilities have been assigned to all internal nodes in SAG, risk values from all predecessors can be merged to obtain the cumulative risk probability or absolute risk probability for each node. Based on derived conditional probability assignments on each node, an effective security hardening plan or a mitigation strategy can be derived:

- for any attack-step node $n \in N_C$ with immediate predecessor set W = parent(n),

$$P_r(n) = Pr(n \mid W) \times \prod_{s \varepsilon W} P_r(s);$$

- for any privilege node $n \in N_D$ with immediate predecessor set W= parent(n), and then

$$P_r(n) = 1 - \prod_{s \varepsilon W}(1 - P_r(s)).$$

Mitigation Strategies **:** Based on the security metrics defined in the previous section, NICE is able to construct the mitigation strategies in response to detected alerts. *Countermeasure pool* can be defined as follows:

A countermeasure pool CM = cm1, cm2, ... , cm2 is a set of countermeasures. Each cm is a tuple *cm*=(cost, intrusiveness, condition, effectiveness), where

i. cost is the unit that describes the expenses required to apply the countermeasure in termsof resources and operational complexity, and it is defined in a range from 1 to 5, and higher metric means higher cost;
ii. intrusiveness is the negative effect that a countermeasure brings to the SLA and the value of intrusiveness is 0 if the countermeasure has no impacts on the SLA;
iii. condition is the requirement for the corresponding countermeasure;
iv. effectiveness is the percentage of probability changes of the node, for which this counter measure is applied.

In general, there are many countermeasures that can be applied to the cloud virtual networking system depending on available countermeasure techniques that can be applied. The optimal countermeasure selection is a multiobjective optimization problem, to calculate MIN (impact, cost) and MAX(benefit). In NICE, the network reconfiguration strategies mainly involve two levels of action: Layer-2 and layer-3. At layer-2, virtual bridges and VLANs are main component in clouds virtual networking system to connect two VMs directly. A virtual bridge is an entity that attaches VIFs. Virtual machines on different bridges are isolated at layer 2. VIFs on the same virtual bridge but with different VLAN tags cannot communicate to each other directly. Based on this layer-2 isolation, NICE can deploy layer-2 network reconfiguration to isolate suspicious VMs. Layer-3 reconfiguration is another way to disconnect an attack path. Through the network controller, the flow table on each OVS or OFS can be modified to change the network topology. Using the virtual network reconfiguration approach at lower layer has the advantage in that upper layer applications will experience minimal impact. Especially, this approach is only possible when using software switching approach to automate the reconfiguration in a highly dynamic networking environment. Countermeasures such as traffic isolation can be implemented by utilizing the traffic engineering capabilities of OVS and OFS to restrict the capacity and reconfigure the virtual network for a suspicious flow. When a suspicious activity such as network and port scanning is detected in the cloud system, it is important to determine whether the detected activity is indeed malicious or not. For example, attackers can purposely hide their scanning behavior to prevent the NIDS from identifying their actions. In such situation, changing the network configuration will force the attacker to perform more explorations, and in turn will make their attacking behaviour stand out.

Countermeasure Selection Algorithm **:** Algorithm2 presents how to select the optimal countermeasure for a given attack scenario. Input to the algorithm is an alert, attack graph G, and a pool of countermeasures CM. The algorithm starts by selecting the node vAlert that corresponds to the alert generated by a NICE-A. Before selecting the countermeasure, the distance of vAlert to the target node is counted. If the distance is greater than a threshold value, countermeasure selection is not performed, but the ACG is updated to keep track of alerts in the system. For the source node vAlert, all the reachable nodes (including the source node) are collected into a set T. Because the alert is generated only after the attacker has performed the action, the probability of vAlert is set to 1 and calculate the new probabilities for all of its child (downstream) nodes in the set T. Now, for all $t \in T$ the applicable countermeasures in CM are selected and new

probabilities are calculated according to the effectiveness of the selected countermeasures. The change in probability of target node gives the benefit for the applied countermeasure. In the next double for-loop, the Return of Investment(ROI)
is computed for each benefit of the applied countermeasure. The countermeasure which when applied on a node gives the least value of ROI, is regarded as the optimal countermeasure. Finally, SAG and ACG are also updated before terminating the algorithm. The complexity of Algorithm 2 is O(| V | × | CM |,where | V | is the number of vulnerabilities and | CM | represents the number of counter measures.

Algorithm 2. Countermeasure Selection

Require: Alert,G(E,V),CM
Let v_{Alert} = Source node of the Alert
if Distance to Target(v_{Alert})>threshold then
  Update_ACG
  return
end if
Let T=Descendant(v_{Alert}) ∪ v_{Alert}
Set $P_r(v_{Alert})=1$
Calculate Risk Prob(T)

Let benefit[| T |, | C |] =∅

for each t ∈ T do
  for each cm ∈ CM do
    if cm.condition(t) then
      Pr(t)= Pr(t) ∗ (1 - cm.effectiveness)
      Calculate Risk Prob(Descendant(t))
      benefit[t,cm]= ∆ Pr(target node)
    end if
  end for
end for

Let ROI[| T |, | CM |] =∅

for each t ∈ T do
  for each cm ∈ CM do
    ROI[t,cm]= benefit[t,cm]/(cost.cm + intrusiveness.cm)
  end for
end for
Update_SAG and Update_ACG
return Select_Optimal_CM(ROI)

V. PERFORMANCE EVALUATION

The performance evaluation is conducted in two directions: The *security performance*, and *the system computing and network reconfiguration overhead due to introduced security mechanism.*

Security Performance Analysis : The security performance of NICE is demonstrated by creating a virtual network testing environment consisting of all the components of NICE.

To evaluate the security performance, a demonstrative virtual cloud system consisting of public (public virtual servers) and private (VMs) virtual domains is established as shown in Fig. 2. Cloud Servers 1 and 2 are connected to Internet through the external firewall. In the DMZ on Server 1, there is one Mail server, one DNS server and one web server. Public network on Server 2 houses SQL server and NAT Gateway Server.

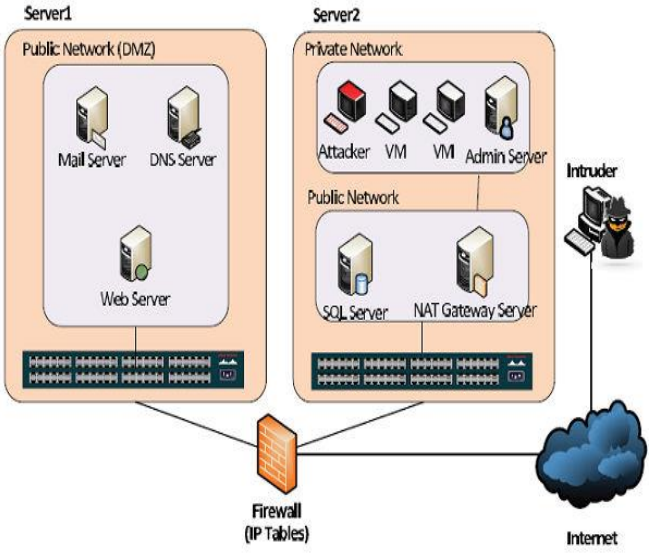

Fig.2 Virtual network topology for security evaluation.

Remote access to VMs in the private network is controlled through SSHD (i.e., SSH Daemon) from the NAT Gateway Server. Table 1 shows the vulnerabilities present in this network and Table 2 shows the corresponding network connectivity that can be explored based on the identified vulnerabilities.

TABLE 1
Vulnerabilities in the Virtual Networked System

Host	Vulnerability	Node	CVE	Base Score
VM group	LICQ buffer overflow	10	CVE 2001-0439	0.75
	MS Video ActiveX Stack buffer overflow	5	CVE 2008-0015	0.93
	GNU C Library loader flaw	22	CVE-2010-3847	0.69
Admin Server	MS SMV service Stack buffer overflow	2	CVE 2008-4050	0.93
Gateway server	OpenSSL uses predictable random variable	15	CVE 2008-0166	0.78
	Heap corruption in OpenSSH	4	CVE 2003-0693	1
	Improper cookies handler in OpenSSH	9	CVE 2007-4752	0.75
Mail server	Remote code execution in SMTP	21	CVE 2004-0840	1
	Squid port scan	19	CVE 2001-1030	0.75
Web server	WebDAV vulnerability in IIS	13	CVE 2009-1535	0.76

TABLE 2
Virtual Network Connectivity

From	To	Protocol
Internet	NAT Gateway server	SSHD
	Mail server	IMAP, SMTP
	Web server	HTTP
Web server	SQL server	SQL
NAT Gateway server	VM group	Basic network protocols
	Admin server	Basic network protocols
VM Group	NAT Gateway server	Basin network protocols
	Mail server	IMAP, SMTP
	SQL server	SQL
	Web server	HTTP
	DNS server	DNS

Attack Graph and Alert Correlation : The attack graph can be generated by utilizing network topology and the vulnerability information, and it is shown in Fig. 3. As the attack progresses, the system generates various alerts that can be related to the nodes in the attack graph. Creating an attack graph requires knowledge of network connectivity, running services, and their vulnerability information. This information is provided to the attack graph generator as the input. Whenever a new vulnerability is discovered or there are changes in the network connectivity and services running through them, the updated information is provided to attack graph generator and old attack graph is updated to a new one. SAG provides information about the possible paths that an attacker can follow. ACG serves the purpose of confirming attackers behavior, and helps in determining false positive and false negative. ACG can also be helpful in predicting attackers next steps.

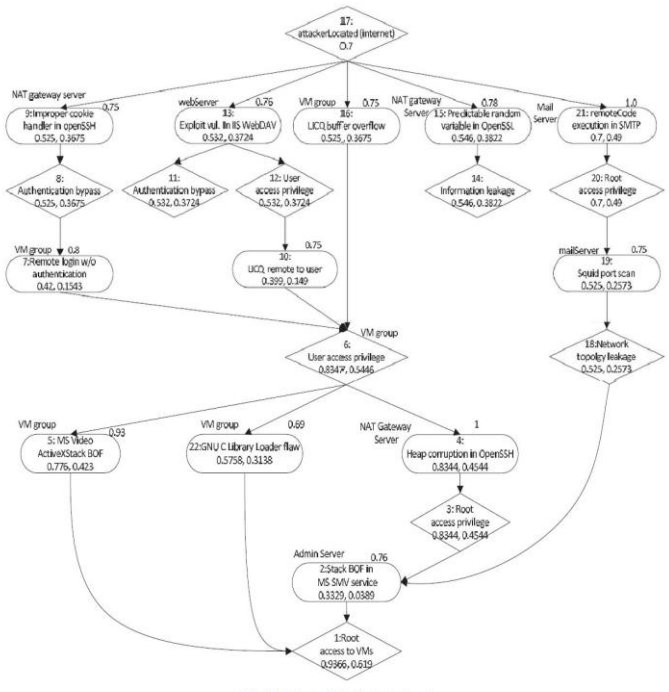

Fig. 3. Attack graph for the test network

Countermeasure Selection : To illustrate how NICE works, consider an example where an alert is generated for node 16 (v_{Alert}=16) when the system detects LICQ Buffer overflow. After the alert is generated, the cumulative probability of node 16 becomes 1 because that attacker has already compromised that node.This triggers a change in cumulative probabilities of child nodes of node 16. Now, the next step is to select the countermeasures from the pool of countermeasures CM. If the countermeasure CM4: Create filtering rules is applied to node 5 and assuming that this countermeasure has effectiveness of 85 percent, the probability of node 5 will change to 0.1164, which causes change in probability values of all child nodes of node 5 thereby accumulating to a decrease of 28.5 percent for the target node 1. Following the same approach for all possible countermeasures that can be applied, the percentage change in the cumulative probability of node 1, i.e., benefit computed are shown in Fig. 4. Apart from calculating the benefit measurements, the evaluation based on ROI is done and represent a comprehensive evaluation considering benefit, cost, and intrusiveness of countermeasure. Fig. 5 shows the ROI evaluations for presented countermeasures. Results show that countermeasures CM2 and CM8 on node 5 have the maximum benefit evaluation; however, their cost and intrusiveness scores indicate that they might not be good candidates for the optimal countermeasure and ROI evaluation results confirm this. The ROI evaluations demonstrate that CM4 on node 5 is the optimal solution.

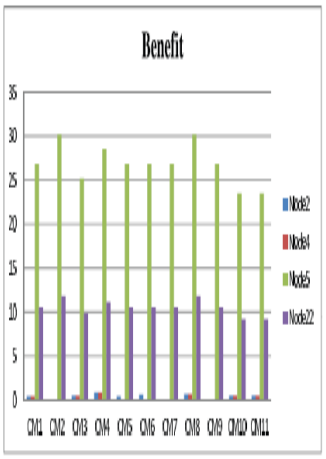

Fig. 4. *Benefit* evaluation chart.

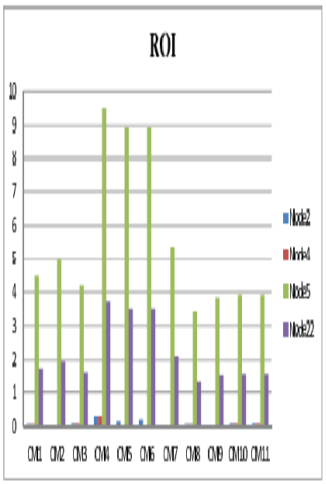

Fig. 5. *ROI* evaluation chart.

Experiment in Private Cloud Environment : For performance analysis and capacity test, configuration in Table 3 is extended to create another test environment, which includes 14 VMs across three cloud servers and configured each VM as a target node to create a dedicated SAG for each VM. These VMs consist of Windows (W) and Linux(L) machines in the private network 172.16.11.0/24, and contains more number of vulnerabilities related to their OSes and applications. Penetration testing scripts are created with Metasploit framework and Armitage as attackers in the test environment. These scripts emulate attackers from different places in the internal and external sources, and launch diversity of attacks based on the vulnerabilities in each

VM. To evaluate security level of a VM, a VSI is defined to represent the security level of each VM in the current virtual network environment. This VSI refers to the VEAbility metric and utilizes two parameters that include Vulnerability and Exploitability as security metrics for a VM. The VSI value ranges from 0 to 10, where lower value means better security.VSI for a virtual machine k is defined as

$$VSI_k = (V_k+E_k)/2$$

where

i. V_k is vulnerability score for VM k. The score is the exponential average of BS from each vulnerability in the VM or a maximum 10, i.e.,V_k= min{10, $\ln\Sigma e^{BaseScore(v)}$}

ii. E_k is exploitability score for VM k. It is the exponential average of exploitability score for all vulnerabilities or a maximum 10 multiplied by the ratio of network services on the VM, i.e.,E_k = (min{10, $\ln\Sigma\ e^{ExploitabilityScore(v)}$}) × S_k/NS_k. S_k represents the number of services provided by VM k. NS_k represents the number of network services the VM k can connect to.

Basically, vulnerability score considers the BSs of all the vulnerabilities on a VM. The BS depicts how easy it is for an attacker to exploit the vulnerability and how much damage it may incur. The exponential addition of BSs allows the vulnerability score to incline toward higher BS values and increases in logarithm-scale based on the number of vulnerabilities. The exploitability score on the other hand shows the accessibility of a target VM, and depends on the ratio of the number of services to the number of network services. Higher exploitability score means that there are many possible paths for that attacker to reach the target. VSI can be used to measure the security level of each VM in the virtual network in the cloud system. A VM with higher value of VSI means is easier to be attacked. To prevent attackers from exploiting other vulnerable VMs, the VMs with higher VSI values need to be monitored closely by the system (e.g., using DPI) and mitigation strategies may be needed to reduce the VSI value when necessary. Fig. 6 shows the plotting of VSI for these virtual machines before countermeasure selection and application. Fig. 7 compares VSI values before and after applying the countermeasure CM4, i.e., creating filtering rules. It shows the percentage
change in VSI after applying countermeasure on all of the VMs. Applying CM4 avoids vulnerabilities and causes VSI to drop without blocking normal services and ports.

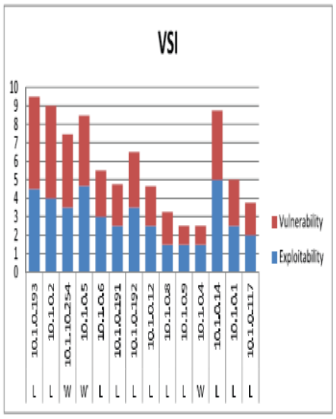

Fig.6. VM security index.

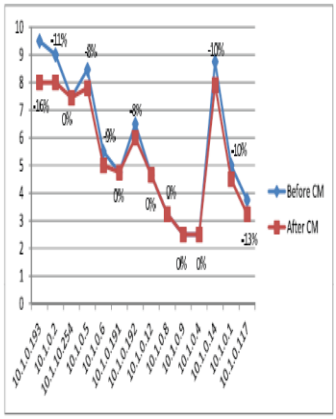

Fig.7. Change in VSI.

False Alarms **:** A cloud system with hundreds of nodes will have huge amount of alerts raised by Snort. Not all of these alerts can be relied upon, and an effective mechanism is needed to verify if such alerts need to be addressed. Since Snort can be programmed to generate alerts with CVE id, one approach that NICE provides is to match if the alert is actually related to some vulnerability being exploited. If so, the existence of that vulnerability in SAG means that the alert is more likely to be a real attack. Thus, the false positive rate will be the joint probability of the correlated alerts, which will not increase the false positive rate compared to each individual false positive rate. In the case of zero-day attack, where the vulnerability is discovered by the attacker but is not detected by vulnerability scanner, the alert being real will be regarded as false, given that there does not exist corresponding node in SAG. Thus, current research does not address how to reduce the false negative rate. It is important to note that vulnerability scanner should be able to detect most recent vulnerabilities and sync with the latest vulnerability database to reduce the chance of Zero-day attacks.

NICE System Performance

NICE was evaluated based on Dom0 and DomU implementations with mirroring-based and proxy-based attack detection agents (i.e., NICE-A). In mirrorbased IDS scenario, two virtual networks were established in each cloud server: Normal network and monitoring network. NICE-A is connected to the monitoring network. Traffic on the normal network is mirrored to the monitoring network using Switched Port Analyzer (SPAN) approach. In the proxy-based IDS solution, NICE-A interfaces two VMs and the traffic goes through NICE-A. Additionally, the NICE-A have been deployed in Dom0 and it removes the traffic duplication function in mirroring and proxy-based solutions. NICE-A running in Dom0 is more efficient because it can sniff the traffic directly on the virtual bridge. However, in DomU, the traffic need to be duplicated on the VMs VIF, causing overhead. When the IDS is running in Intrusion Prevention System (IPS) mode, it needs to intercept all the traffic and perform packet checking, which consumes more system resources as compared to IDS mode. To demonstrate performance evaluations, four

metrics namely CPU utilization, network capacity, agent processing capacity, and communication delay were used. The evaluation on cloud servers with Intel quad-core Xeon 2.4-GHz CPU and 32-G memory was performed. Packet generator was used to mimic real traffic in the cloud system. As shown in Fig. 8, the traffic load, in the form of packet sending speed, increases from 1 to 3,000 packets per second. The performance at Dom0 consumes less CPU and the IPS mode consumes the maximum CPU resources. When the packet rate reaches to 3,000 packets per second, the CPU utilization of IPS at DomU reaches its limitation, while the IDS mode at DomU only occupies about 68 percent.

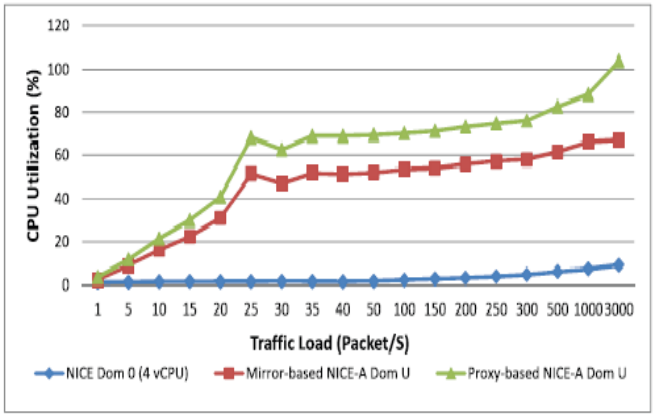

Fig. 8. CPU utilization of NICE-A.

Fig. 8, represents the performance of NICE-A in terms of percentage of successfully analyzed packets. The higher this value is, more packets this agent can handle. IPS agent demonstrates 100 percent performance because every packet captured by the IPS is cached in the detection agent buffer. However, 100 percent success analyzing rate of IPS is at the cost of the analyzing delay. For other two types of agents, the detection agent does not store the captured packets and, thus, no delay is introduced. However, they all experience packet drop when traffic load is huge. For a small-scale cloud system this approach works well. The performance evaluation includes two parts. First, security performance evaluation. It shows that the system helps to prevent vulnerable VMs from being compromised and do so in less intrusive and cost effective manner. Second, CPU and throughput performance evaluation. It shows the limits of using the proposed solution in terms of networking throughputs based on software switches and CPU usage when running detection engines on Dom 0 and Dom U. The performance results provide a benchmark for the given hardware setup and shows how much traffic can be handled by using a single detection domain.

VI. CONCLUSION

NICE is used to detect and mitigate collaborative attacks in the cloud virtual networking environment. NICE utilizes the attack graph model to conduct attack detection and prediction. The proposed solution investigates how to use the programmability of software switches-based solutions to improve the detection accuracy and defeat victim exploitation phases of collaborative attacks. The system performance evaluation demonstrates the feasibility of NICE and shows that the proposed solution can significantly reduce the risk of the cloud system from being exploited and abused
by internal and external attackers. NICE only investigates the network IDS approach to counter zombie explorative attacks.

VII. FUTURE SCOPE

To improve the detection accuracy, *host-based IDS* solutions are needed to be incorporated and to cover the whole spectrum of IDS in the cloud system. This should be investigated in the future work. The scalability of the proposed NICE solution can be investigated by investigating the decentralized network control and attack analysis model.

REFERENCES

[1] Chun-Jen Chung, Pankaj Khatkar, TianyiXing,Jeongkeun Lee and Dijiang Huang: Nice-Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems:IEEE Transactions on Dependable and Secure Computing, 10(4):198 211, July 2013.
[2] Cloud Sercurity Alliance: Top Threats to Cloud Computing v1.0:https://cloudsecurityalliance.org/topthreats/csathreats. v1.0.pdf, Mar. 2010.
[3] B. Joshi, A. Vijayan, and B. Joshi: Securing Cloud Computing Environment Against DDoS Attacks: Proc. IEEE Intl Conf. Computer Comm. and Informatics (ICCCI 12), Jan. 2012.
[4] Open vSwitch Project: http://openvswitch.org, May 2012
[5] Z. Duan, P. Chen, F. Sanchez, Y. Dong, M. Stephenson, and J.Barker: Detecting Spam Zombies by Monitoring Outgoing Messages: IEEE Trans. Dependable and Secure Computing, vol. 9, no. 2, pp. 198-210, Apr. 2012.
[6] X. Ou, S. Govindavajhala, and A.W. Appel: MulVAL: A Logic-Based Network Security Analyzer: Proc. 14th USENIX Security Symp., pp. 113-128, 2005.
[7] S. Roschke, F. Cheng, and C. Meinel: A New Alert Correlation Algorithm Based on Attack Graph: Proc. Fourth Intl Conf. Computational Intelligence in Security for Information Systems, pp. 58-67, 2011.
[8] M. Tupper and A. Zincir-Heywood: VEA-bility Security Metric: A Network Security Analysis Tool: Proc. IEEE Third Intl Conf. Availability, Reliability and Security (ARES 08), pp. 950-957, Mar.2008.