NICE: Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems

Chun-Jen Chung, *Student Member, IEEE,* Pankaj Khatkar, *Student Member, IEEE,* Tianyi Xing, Jeongkeun Lee, *Member, IEEE,* and Dijiang Huang *Senior Member, IEEE*

Abstract—Cloud security is one of most important issues that has attracted a lot of research and development effort in past few years. Particularly, attackers can explore vulnerabilities of a cloud system and compromise virtual machines to deploy further large-scale Distributed Denial-of-Service (DDoS). DDoS attacks usually involve early stage actions such as multi-step exploitation, low frequency vulnerability scanning, and compromising identified vulnerable virtual machines as zombies, and finally DDoS attacks through the compromised zombies. Within the cloud system, especially the Infrastructure-as-a-Service (IaaS) clouds, the detection of zombie exploration attacks is extremely difficult. This is because cloud users may install vulnerable applications on their virtual machines. To prevent vulnerable virtual machines from being compromised in the cloud, we propose a multi-phase distributed vulnerability detection, measurement, and countermeasure selection mechanism called NICE, which is built on attack graph based analytical models and reconfigurable virtual network-based countermeasures. The proposed framework leverages OpenFlow network programming APIs to build a monitor and control plane over distributed programmable virtual switches in order to significantly improve attack detection and mitigate attack consequences. The system and security evaluations demonstrate the efficiency and effectiveness of the proposed solution.

Index Terms—Network Security, Cloud Computing, Intrusion Detection, Attack Graph, Zombie Detection.

1 Introduction

Recent studies have shown that users migrating to the cloud consider security as the most important factor. A recent Cloud Security Alliance (CSA) survey shows that among all security issues, abuse and nefarious use of cloud computing is considered as the top security threat [1], in which attackers can exploit vulnerabilities in clouds and utilize cloud system resources to deploy attacks. In traditional data centers, where system administrators have full control over the host machines, vulnerabilities can be detected and patched by the system administrator in a centralized manner. However, patching known security holes in cloud data centers, where cloud users usually have the privilege to control software installed on their managed VMs, may not work effectively and can violate the *Service Level Agreement* (SLA). Furthermore, cloud users can install vulnerable software on their VMs, which essentially contributes to loopholes in cloud security. The challenge is to establish an effective vulnerability/attack detection and response system for accurately identifying attacks and minimizing the impact of security breach to cloud users.

In [2], M. Armbrust *et al*. addressed that protecting "Business continuity and services availability" from service outages is one of the top concerns in cloud computing systems. In a cloud system where the infrastructure is shared by potentially millions of users, abuse and nefarious use of the shared infrastructure benefits attackers to exploit vulnerabilities of the cloud and use its resource to deploy attacks in more efficient ways [3]. Such attacks are more effective in the cloud environment since cloud users usually share computing resources, e.g., being connected through the same switch, sharing with the same data storage and file systems, even with potential attackers [4]. The similar setup for VMs in the cloud, e.g., virtualization techniques, VM OS, installed vulnerable software, networking, etc., attracts attackers to compromise multiple VMs.

In this article, we propose NICE (Network Intrusion detection and Countermeasure sElection in virtual network systems) to establish a defense-in-depth intrusion detection framework. For better attack detection, NICE incorporates attack graph analytical procedures into the intrusion detection processes. We must note that the design of NICE does not intend to improve any of the existing intrusion detection algorithms; indeed, NICE employs a reconfigurable virtual networking approach to detect and counter the attempts to compromise VMs, thus preventing zombie VMs.

In general, NICE includes two main phases: (1) deploy a lightweight mirroring-based network intrusion detection agent (NICE-A) on each cloud server to capture and analyze cloud traffic. A NICE-A periodically scans the virtual system vulnerabilities within a cloud server to establish Scenario Attack Graph (SAGs), and then based on the severity of identified vulnerability towards the collaborative attack goals, NICE will decide whether or not to put a VM in network inspection state. (2) Once a VM enters inspection state, Deep Packet Inspection (DPI)

- *C-J. Chung, P. Khatkar, T. Xing, and D. Huang are with the Department of Computer Science, Arizona State University, Tempe, AZ 85287. E-mail: {chun-jen.chung, pkhatkar, tianyi.xing, dijiang}@asu.edu*
- *J. Lee is with Hewlett-Packard Laboratories. Palo Alto, CA 94304. E-mail:jklee@hp.com*

Digital Object Indentifier 10.1109/TDSC.2013.8 1545-5971/13/$31.00 © 2013 IEEE

is applied, and/or virtual network reconfigurations can be deployed to the inspecting VM to make the potential attack behaviors prominent.

NICE significantly advances the current network IDS/IPS solutions by employing programmable virtual networking approach that allows the system to construct a dynamic reconfigurable IDS system. By using software switching techniques [5], NICE constructs a mirroring-based traffic capturing framework to minimize the interference on users' traffic compared to traditional bump-in-the-wire (i.e., proxy-based) IDS/IPS. The programmable virtual networking architecture of NICE enables the cloud to establish inspection and quarantine modes for suspicious VMs according to their current vulnerability state in the current SAG. Based on the collective behavior of VMs in the SAG, NICE can decide appropriate actions, for example DPI or traffic filtering, on the suspicious VMs. Using this approach, NICE does not need to block traffic flows of a suspicious VM in its early attack stage. The contributions of NICE are presented as follows:

- We devise NICE, a new multi-phase 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 services.
- NICE incorporates a software switching solution to quarantine and inspect suspicious VMs for further investigation and protection. Through programmable network approaches, NICE can improve the attack detection probability and improve the resiliency to VM exploitation attack without interrupting existing normal cloud services.
- NICE employs a novel attack graph approach for attack detection and prevention by correlating attack behavior and also suggests effective countermeasures.
- NICE optimizes the implementation on cloud servers to minimize resource consumption. Our study shows that NICE consumes less computational overhead compared to proxy-based network intrusion detection solutions.

The rest of paper is organized as follows. Section II presents the related work. System models are described in Section III, Section IV describes system design and implementation. The proposed security measurement, mitigation, and countermeasures are presented in Section V and Section VI evaluates NICE in terms of network performance and security. Finally, Section VII describes future work and concludes this paper.

2 Related Work

In this section, we present literatures of several highly related research areas to NICE, including: zombie detection and prevention, attack graph construction and security analysis, and software defined networks for attack countermeasures.

The area of detecting malicious behavior has been well explored. The work by Duan *et al*. [6] focuses on the detection of compromised machines that have been recruited to serve as spam zombies. Their approach, SPOT, is based on sequentially scanning outgoing messages while employing a statistical method Sequential Probability Ratio Test (SPRT), to quickly determine whether or not a host has been compromised. BotHunter [7] 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. BotSniffer [8] 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 able to represent a series of exploits, called *atomic attacks*, that lead to an undesirable state, for example a state where an attacker has obtained administrative access to a machine. There are many automation tools to construct attack graph. O. Sheyner *et al*. [9] proposed a technique based on a modified symbolic model checking NuSMV [10] and Binary Decision Diagrams (BDDs) to construct attack graph. Their model can generate all possible attack paths, however, the scalability is a big issue for this solution. P. Ammann *et al*. [11] introduced the assumption of monotonicity, which states that the precondition of a given exploit is never invalidated by the successful application of another exploit. In other words, attackers never need to backtrack. With this assumption, they can obtain a concise, scalable graph representation for encoding attack tree. X. Ou *et al.* proposed an attack graph tool called MulVAL [12], which adopts a logic programming approach and uses Datalog language to model and analyze network system. The attack graph in the MulVAL 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. In order to provide the security assessment and alert correlation features, in this paper, we modified and extended MulVAL's attack graph structure.

Intrusion Detection System (IDS) and firewall are widely used to monitor and detect suspicious events in the network. However, the false alarms and the large volume of raw alerts from IDS are two major problems for any IDS implementations. In order to identify the source or target of the intrusion in the network, especially to detect multi-step attack, the alert correction is a must-have tool. The primary goal of alert correlation is to provide system support for a global and condensed view of network attacks by analyzing raw alerts [13].

Many attack graph based alert correlation techniques have been proposed recently. L. Wang *et al*. [14] devised an in-memory structure, called *queue graph* (QG), 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. Roschke *et al*. [15] proposed a modified attack-graph-based correlation algorithm 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. Each path in DG represents a subset of alerts that might be part of an attack scenario. However, their algorithm involved all pairs shortest path searching and sorting in DG, which consumes considerable computing power.

After knowing the possible attack scenarios, applying countermeasure is the next important task. Several solutions have been proposed to select optimal countermeasures based on the likelihood of the attack path and cost benefit analysis. A. Roy *et al*. [16] proposed an *attack countermeasure tree* (ACT) to consider attacks and countermeasures together in an attack tree structure. They devised several objective functions based on greedy and branch and bound techniques to minimize the number of countermeasure, reduce investment cost, and maximize the benefit from implementing a certain countermeasure set. In their design, each countermeasure optimization problem could be solved with and without probability assignments to the model. However, their solution focuses on a static attack scenario and predefined countermeasure for each attack. N. Poolsappasit *et al*. [17] proposed a *Bayesian attack graph* (BAG) to address dynamic security risk management problem and applied a genetic algorithm to solve countermeasure optimization problem.

Our solution utilizes a new network control approach called SDN [18], where networking functions can be programmed through software switch and OpenFlow protocol [19], plays a major role in this research. Flow-based switches, such as OVS [5] and OpenFlow Switch (OFS) [19], support fine-grained and flow-level control for packet switching [20]. With the help of the central controller, all OpenFlow-based switches can be monitored and configured. We take advantage of flow-based switching (OVS) and network controller to apply the selected network countermeasures in our solution.

3 NICE Models

In this section, we describe how to utilize attack graphs to model security threats and vulnerabilities in a virtual networked system, and propose a VM protection model based on virtual network reconfiguration approaches to prevent VMs from being exploited.

3.1 Threat Model

In our attack model, we assume that an attacker can be located either outside or inside of the virtual networking system. The attacker's primary goal is to exploit vulnerable VMs and compromise them as zombies. Our protection model focuses on virtual-network-based attack detection and reconfiguration solutions to improve the resiliency to zombie explorations. Our work does not involve host-based IDS and does not address how to handle encrypted traffic for attack detections.

Our proposed solution can be deployed in an Infrastructure-as-a-Service (IaaS) cloud networking system, and we assume that the Cloud Service Provider (CSP) is benign. We also assume that 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. Physical security of cloud server is out of scope of this paper. We assume that the hypervisor is secure and free of any vulnerabilities. The issue of a malicious tenant breaking out of DomU and gaining access to physical server have been studied in recent work [21] and are out of scope of this paper.

3.2 Attack Graph Model

An attack graph is a modeling tool to illustrate all possible multi-stage, multi-host attack paths that are crucial to understand threats and then to decide appropriate countermeasures [22]. In an attack graph, each node represents either precondition or consequence of an exploit. The actions are not necessarily an active attack since normal protocol interactions can also be used for attacks. Attack graph is helpful in identifying potential threats, possible attacks and known vulnerabilities in a cloud system.

Since the attack graph provides details of all known vulnerabilities in the system and the connectivity information, we get a whole picture of current security situation of the system where we can predict the possible threats and attacks by correlating detected events or activities. If an event is recognized as a potential attack, we can apply specific countermeasures 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, we extend the notation of MulVAL logic attack graph as presented by X. Ou *et al.* [12] and define as Scenario Attack Graph (SAG).

Definition 1 (Scenario Attack Graph)**.** *An Scenario Attack Graph is a tuple SAG=(V, E), where*

1. $V = N_C \cup N_D \cup N_R$ *denotes a set of vertices that include three types namely conjunction node* N_C *to represent exploit, disjunction node* N_D *to denote result of exploit, and root node* N_R *for showing initial step of an attack scenario.*
2. $E = E_{pre} \cup E_{post}$ *denotes the set of directed edges. An edge* $e \in E_{pre} \subseteq N_D \times N_C$ *represents that* N_D *must be satisfied to achieve* N_C*. An edge* $e \in E_{post} \subseteq N_C \times N_D$ *means that the consequence shown by* N_D *can be obtained if* N_C *is satisfied.*

Node $v_c \in N_C$ is defined as a three tuple $(Hosts, vul, alert)$ representing a set of IP addresses, vulnerability information such as CVE [23], and alerts

related to v_c, 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 scenario attack graph.

For correlating the alerts, we refer to the approach described in [15] and define a new Alert Correlation Graph (ACG) to map alerts in ACG to their respective nodes in SAG. To keep track of attack progress, we track the source and destination IP addresses for attack activities.

Definition 2 (Alert Correlation Graph)**.** *An ACG is a three tuple $ACG = (A, E, P)$, where*

1. *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 IP address, type of the alert, and timestamp of the alert respectively.*
2. *Each alert a maps to a pair of vertices (v_c, v_d) in SAG using $map(a)$ function, i.e., $map(a) : a \mapsto \{(v_c, v_d) | (a.src \in v_c.Hosts) \wedge (a.dst \in v_d.Hosts) \wedge (a.cls = v_c.vul)\}$.*
3. *E is a set of directed edges representing correlation between two alerts (a, a') if criteria below satisfied:*
   i. *$(a.ts < a'.ts) \wedge (a'.ts - a.ts < threshold)$*
   ii. *$\exists (v_d, v_c) \in E_{pre} : (a.dst \in v_d.Hosts \wedge a'.src \in v_c.Hosts)$*
4. *P is set of paths in ACG. A path $S_i \subset P$ is a set of related alerts in chronological order.*

We assume that A contains aggregated alerts rather than raw alerts. Raw alerts having same source and destination IP addresses, attack type and timestamp within a specified window are aggregated as *Meta Alerts*. Each ordered pair (a, a') in ACG maps to two neighbor vertices in SAG with timestamp difference of two alerts within a predefined *threshold*. ACG shows dependency of alerts in chronological order and we can find related alerts 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 $S_i \subset P$ represents alerts that belong to the same attack scenario.

We explain a method for utilizing SAG and ACG together so as to predict an attacker's behavior. *Alert_Correlation* algorithm is followed for every alert detected and returns one or more paths S_i. For every alert a_c that is received from the IDS, it is added to ACG if it does not exist. For this new alert a_c, the corresponding vertex in the *SAG* is found by using function $map(a_c)$ (line 3). For this vertex in *SAG*, alert related to its parent vertex of type N_C is then correlated with the current alert a_c (line 5). This creates a new set of alerts that belong to a path S_i in ACG (line 8) or splits out a new path S_{i+1} from S_i with subset of S_i before the alert a and appends a_c to S_{i+1} (line 10). In the end of this algorithm, the ID of a_c 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 a_c, SAG, ACG
1: **if** (a_c is a new alert) **then**
2: create node a_c in ACG
3: $n_1 \leftarrow v_c \in map(a_c)$
4: **for all** $n_2 \in parent(n_1)$ **do**
5: create edge $(n_2.alert, a_c)$
6: **for all** S_i containing a **do**
7: **if** a is the last element in S_i **then**
8: append a_c to S_i
9: **else**
10: create path $S_{i+1} = \{subset(S_i, a), a_c\}$
11: **end if**
12: **end for**
13: add a_c to $n_1.alert$
14: **end for**
15: **end if**
16: **return** S

3.3 VM Protection Model

The VM protection model of NICE consists of a VM profiler, a security indexer and a state monitor. We specify security index for all the VMs in the network depending upon various factors like connectivity, the number of vulnerabilities present and their impact scores. The impact score of a vulnerability, as defined by the CVSS guide [24], helps 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.

Definition 3 (VM State)**.** *Based on the information gathered from the network controller, VM states can be defined as following:*

1. Stable*: there does not exist any known vulnerability on the VM.*
2. Vulnerable*: presence of one or more vulnerabilities on a VM, which remains unexploited.*
3. Exploited*: at least one vulnerability has been exploited and the VM is compromised.*
4. Zombie*: VM is under control of attacker.*

4 NICE System Design

In this section, we first present the system design overview of NICE and then detailed descriptions of its components.

4.1 System design overview

The proposed NICE framework is illustrated in figure 1. It shows the NICE framework within one cloud server cluster. 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 (i.e., virtual switches

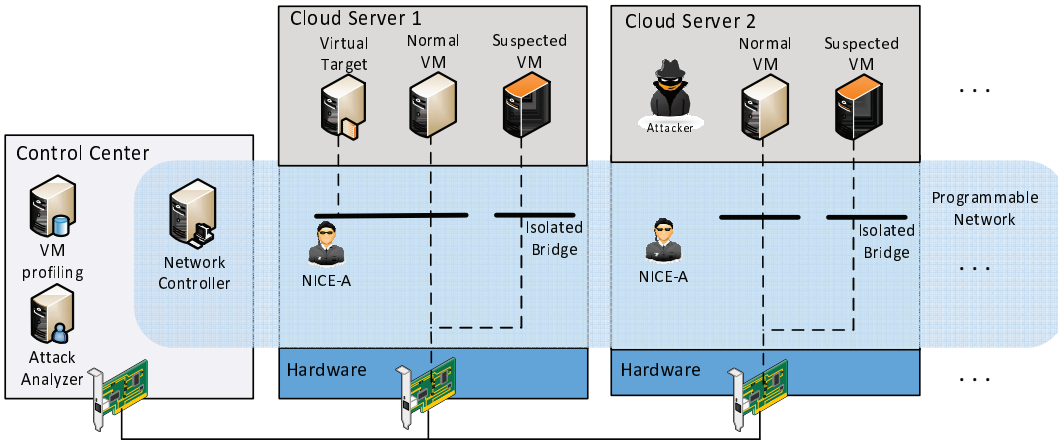


Fig. 1. NICE architecture within one cloud server cluster.

built on one or multiple Linux software bridges). 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.

In the following description, our terminologies are based on the XEN virtualization technology. NICE-A is a network intrusion detection engine that can be installed in either Dom0 or DomU of a XEN cloud server to capture and filter malicious traffic. 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 realtime vulnerability scans. Offline scanning can be done by running penetration tests and online realtime vulnerability scanning can be triggered by the network controller (e.g., when new ports are opened and identified by OpenFlow switches) 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 OpenFlow switches.

4.2 System Components

In this section we explain each component of NICE.

4.2.1 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. In our experiment, Snort is used to implement NICE -A in Dom0. It will sniff a mirroring port on each virtual bridge in the Open vSwitch. 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. The NICE-A sniffing rules have been custom defined to suite our needs. 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's more efficient to scan the traffic in Dom0 since all traffic in the cloud server needs go through it, however our design is independent to the installed VM. In the performance evaluation section, we will demonstrate the trade-offs of installing NICE-A in Dom0 and DomU.

We must note that the alert detection quality of NICE-A depends on the implementation of NICE-A which uses Snort. We do not focus on the detection accuracy of Snort in this article. Thus, the individual alert detection's false alarm rate does not change. However, the false alarm rate could be reduced through our architecture design. We will discuss more about this issue in the later section.

4.2.2 VM Profiling

Virtual machines in the cloud can be profiled to get precise information about their state, services running, open ports, etc. One major factor that counts towards a VM profile is its connectivity with other VMs. 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. Also required is the knowledge of services running on a VM so as to verify the authenticity of alerts pertaining to that VM. 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 5 tuples (source MAC address, destination MAC address, source IP address, destination IP address, protocol). We can have traffic pattern where packets emanate from a single IP and are delivered to multiple destination IP addresses, and vice-versa.

4.2.3 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 Scenario Attack Graph (*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.

In summary, 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 VM's 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 VM's 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, such as Open Source Vulnerability Database (OSVDB)[25], Common Vulnerabilities and Exposures List (CVE)[23], and NIST National Vulnerability Database (NVD) [26].

The Attack Analyzer also handles alert correlation and analysis operations. This component has two major functions: (1) constructs Alert Correlation Graph (*ACG*), (2) provides threat information and appropriate countermeasures to network controller for virtual network reconfiguration.

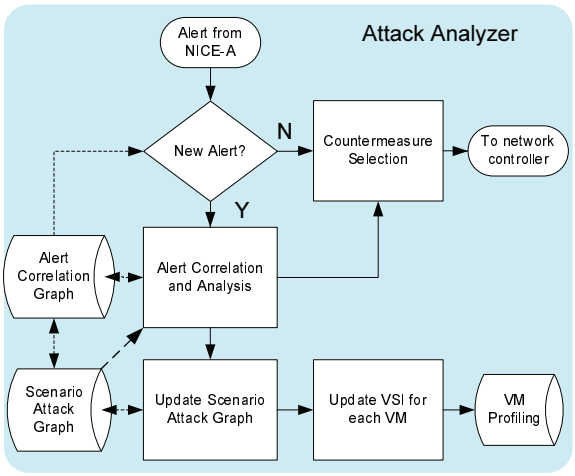


Fig. 2. Workflow of Attack Analyzer.

Figure 2 shows the workflow in the attack analyzer component. 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 (i.e., matching the attack signature), the attack analyzer performs countermeasure selection procedure according to Algorithm 2, 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.

4.2.4 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 [20]. In NICE, within each cloud server there is a software switch, for example, Open vSwitch (OVS) [5], which is used as the edge switch for VMs to handle traffic in & out from VMs. The communication between cloud servers (i.e., physical servers) is handled by physical OpenFlow-capable Switch (OFS). In NICE, we integrated the control functions for both OVS and OFS into the network controller that allows the cloud system to set security/filtering rules in an integrated and comprehensive manner.

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 [27],

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 [20], 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* 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 Deep Packet Inspection (DPI) mode. An alert with a minor threat level can be generated due to the presence of a vulnerable VM. For this case, in order to intercept the VM's 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.

5 NICE Security Measurement, Attack Mitigation and Countermeasures

In this section, we present the methods for selecting the countermeasures for a given attack scenario. When vulnerabilities are discovered or some VMs are identified as suspicious, several countermeasures can be taken to restrict attackers' capabilities and it's 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.

5.1 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 [28] is a good choice. In order 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. We use G_V to denote the priori risk probability for the root node of the graph and usually the value of G_V 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 CVSS (Common Vulnerability Scoring System). The base score as shown in (1) [24], is calculated by the impact and exploitability factor of the vulnerability. Base score can be directly obtained from National Vulnerability Database [26] by searching for the vulnerability CVE id.

$$BS = (0.6 \times IV + 0.4 \times E - 1.5) \times f(IV), \qquad (1)$$

where,

$$IV = 10.41 \times (1 - (1 - C) \times (1 - I) \times (1 - A)),$$

$$E = 20 \times AC \times AU \times AV,$$

and

$$f(IV) = \begin{cases} 0 & \text{if } IV = 0, \\ 1.176 & \text{otherwise.} \end{cases}$$

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 our attack graph, we assign each internal node with its BS value divided by 10, as shown in (2).

$$G_M[e] = Pr(e = T) = BS(e)/10, \forall e \in N_C. \qquad (2)$$

In the attack graph, the relations between exploits can be disjunctive or conjunctive according to how they are related through their dependency conditions [29]. 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. We propose the following probability derivation relations:

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

$$Pr(n|W) = G_M[n] \times \prod_{s \in W} Pr(s|W); \qquad (3)$$

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

$$Pr(n|W) = 1 - \prod_{s \in W} (1 - Pr(s|W)). \quad (4)$$

Once conditional probabilities have been assigned to all internal nodes in SAG, we can merge risk values from all predecessors to obtain the *cumulative risk probability* or *absolute risk probability* for each node according to (5) and (6). Based on derived conditional probability assignments on each node, we can then derive an effective security hardening plan or a mitigation strategy:

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

$$Pr(n) = Pr(n|W) \times \prod_{s \in W} Pr(s); \quad (5)$$

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

$$Pr(n) = 1 - \prod_{s \in W} (1 - Pr(s)). \quad (6)$$

5.2 Mitigation Strategies

Based on the security metrics defined in the previous subsection, NICE is able to construct the mitigation strategies in response to detected alerts. First, we define the term *countermeasure pool* as follows:

Definition 4 (Countermeasure Pool)**.** *A countermeasure pool* $CM = \{cm_1, cm_2, \ldots, cm_n\}$ *is a set of countermeasures. Each* $cm \in CM$ *is a tuple cm =* (cost, intrusiveness, condition, effectiveness*), where*

1. cost *is the unit that describes the expenses required to apply the countermeasure in terms of resources and operational complexity, and it is defined in a range from 1 to 5, and higher metric means higher cost;*
2. intrusiveness *is the negative effect that a countermeasure brings to the Service Level Agreement (SLA) and its value ranges from the least intrusive (1) to the most intrusive (5), and the value of intrusiveness is 0 if the countermeasure has no impacts on the SLA;*
3. condition *is the requirement for the corresponding countermeasure;*
4. effectiveness *is the percentage of probability changes of the node, for which this countermeasure 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. Without losing the generality, several common virtual-networking-based countermeasures are listed in table 1. The optimal countermeasure selection is a multi-objective 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 (including tunnels that can be established between two bridges) and VLANs are main component in cloud's virtual networking system to connect two VMs directly. A virtual bridge is an entity that attaches Virtual Interfaces (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. For example, vulnerabilities due to Arpspoofing [30] attacks are not possible when the suspicious VM is isolated to a different bridge. As a result, this countermeasure disconnects an attack path in the attack graph causing the attacker to explore an alternate attack path. 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.

TABLE 1
Possible Countermeasure Types

No.	Countermeasure	Intrusiveness	Cost
1	Traffic redirection	3	3
2	Traffic isolation	4	2
3	Deep Packet Inspection	3	3
4	Creating filtering rules	1	2
5	MAC address change	2	1
6	IP address change	2	1
7	Block port	4	1
8	Software patch	5	4
9	Quarantine	5	2
10	Network reconfiguration	0	5
11	Network topology change	0	5

We must note that 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 behavior stand out.

5.3 Countermeasure selection

Algorithm 2 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 v_{Alert} that corresponds to the alert generated by a NICE-A. Before selecting the countermeasure, we

count the distance of v_{Alert} to the $target\_node$. If the distance is greater than a threshold value, we do not perform countermeasure selection but update the ACG to keep track of alerts in the system (line 3). For the source node v_{Alert}, all the reachable nodes (including the source node) are collected into a set T (line 6). Because the alert is generated only after the attacker has performed the action, we set the probability of v_{Alert} to 1 and calculate the new probabilities for all of its child (downstream) nodes in the set T (line 7 & 8). 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 (line 13 & 14). The change in probability of $target\_node$ gives the $benefit$ for the applied countermeasure using (7). In the next double *for-loop*, we compute the *Return of Investment* (ROI) for each $benefit$ of the applied countermeasure based on (8). 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 $\mathcal{O}(|V| \times |CM|)$ where $|V|$ is the number of vulnerabilities and $|CM|$ represents the number of countermeasures.

Algorithm 2 Countermeasure_Selection

Require: $Alert, G(E, V), CM$

1: Let v_{Alert} = Source node of the $Alert$
2: **if** Distance_to_Target(v_{Alert}) $> threshold$ **then**
3: Update_ACG
4: **return**
5: **end if**
6: Let $T = Descendant(v_{Alert}) \cup v_{Alert}$
7: Set $Pr(v_{Alert}) = 1$
8: Calculate_Risk_Prob(T)
9: Let $benefit[|T|, |CM|] = \emptyset$
10: **for** each $t \in T$ **do**
11: **for** each $cm \in CM$ **do**
12: **if** $cm.condition(t)$ **then**
13: $Pr(t) = Pr(t) * (1 - cm.effectiveness)$
14: Calculate_Risk_Prob($Descendant(t)$)
15:

$$benefit[t, cm] = \Delta Pr(target\_node). \quad (7)$$

16: **end if**
17: **end for**
18: **end for**
19: Let $ROI[|T|, |CM|] = \emptyset$
20: **for** each $t \in T$ **do**
21: **for** each $cm \in CM$ **do**
22:

$$ROI[t, cm] = \frac{benefit[t, cm]}{cost.cm + intrusiveness.cm}. \quad (8)$$

23: **end for**
24: **end for**
25: Update_SAG and Update_ACG
26: **return** Select_Optimal_CM(ROI)

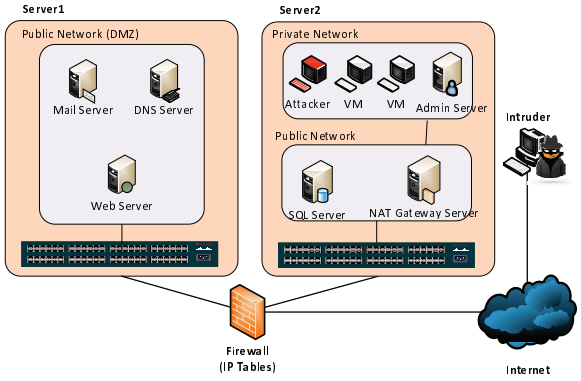


Fig. 3. Virtual network topology for security evaluation.

6 Performance Evaluation

In this section we present the performance evaluation of NICE. Our evaluation is conducted in two directions: the security performance, and the system computing and network reconfiguration overhead due to introduced security mechanism.

6.1 Security Performance Analysis

To demonstrate the security performance of NICE, we created a virtual network testing environment consisting of all the presented components of NICE.

6.1.1 Environment and Configuration

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 Figure 3. Cloud Servers 1 and 2 are connected to Internet through the external firewall. In the Demilitarized Zone (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. Remote access to VMs in the private network is controlled through SSHD (i.e., SSH Daemon) from the NAT Gateway Server. Table 2 shows the vulnerabilities present in this network and table 3 shows the corresponding network connectivity that can be explored based on the identified vulnerabilities.

TABLE 2
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

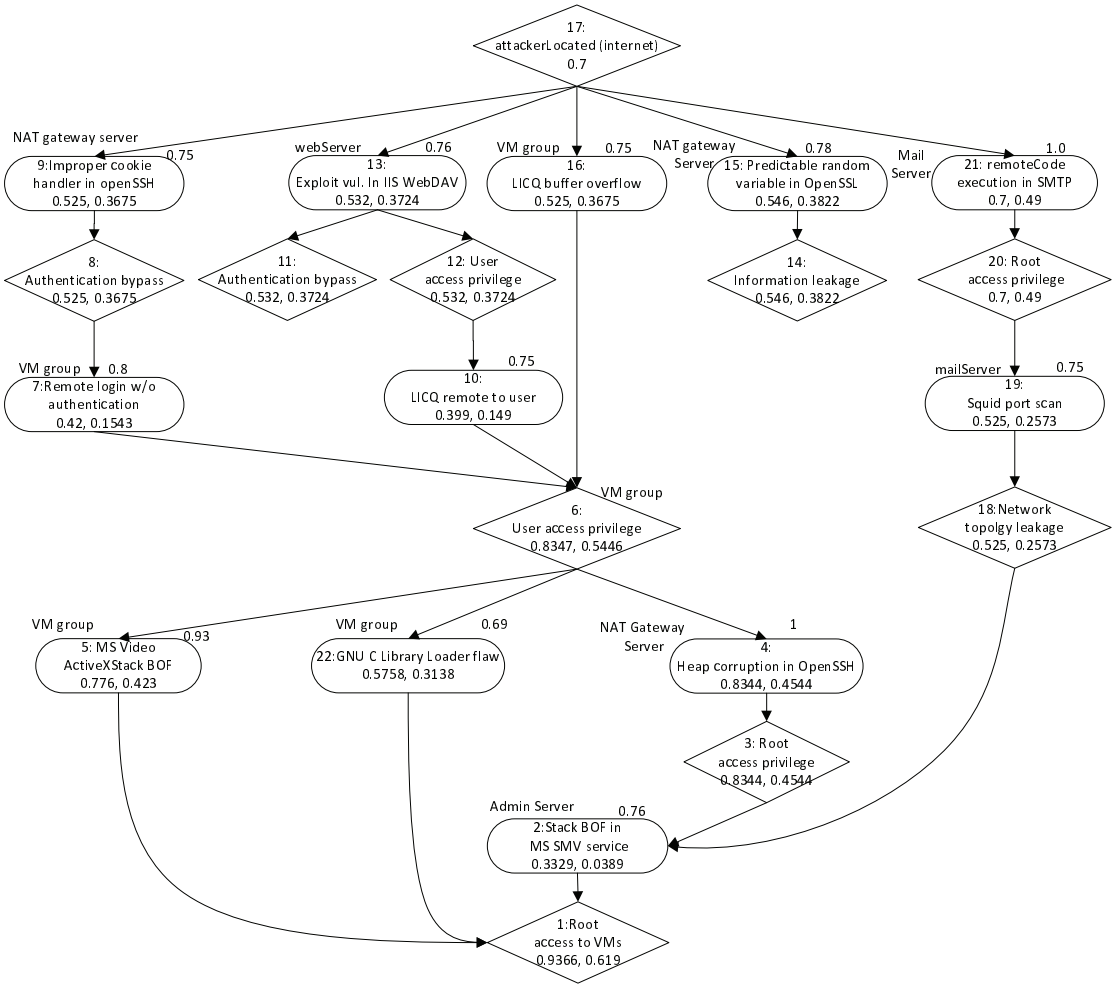


Fig. 4. Attack graph for the test network.

TABLE 3
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

6.1.2 Attack Graph and Alert Correlation

The attack graph can be generated by utilizing network topology and the vulnerability information, and it is shown in Figure 4. 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.

6.1.3 Countermeasure Selection

To illustrate how NICE works, let us consider for example, 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 we assume that this countermeasure has effectiveness of

85%, 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% 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 using (7) are shown in Figure 5.

Apart from calculating the *benefit* measurements, we also present the evaluation based on *Return of Investment (ROI)* using (8) and represent a comprehensive evaluation considering *benefit*, *cost* and *intrusiveness* of countermeasure. Figure 6 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.

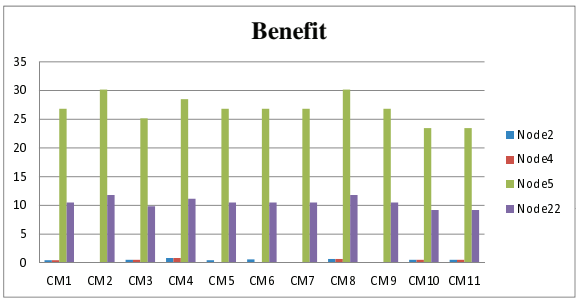


Fig. 5. *Benefit* evaluation chart.

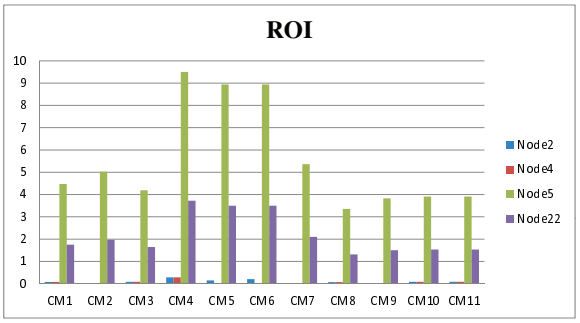


Fig. 6. *ROI* evaluation chart.

6.1.4 Experiment in Private Cloud Environment

Previously we presented an example where the attackers target is VM in the private network. For performance analysis and capacity test, we extended the configuration in figure 3 to create another test environment which includes 14 VMs across 3 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. We created penetration testing scripts with Metasploit framework [31] and Armitage [32] as attackers in our 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.

In order to evaluate security level of a VM, we define a VM Security Index (VSI) to represent the security level of each VM in the current virtual network environment. This VSI refers to the VEA-bility metric [33] 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. We now define VSI:

Definition 5 (VM Security Index)**.** *VSI for a virtual machine k is defined as $VSI_k = (V_k + E_k)/2$, where*

1. *V_k is vulnerability score for VM k. The score is the exponential average of base score from each vulnerability in the VM or a maximum 10, i.e., $V_k = \min\{10, \ln \sum e^{BaseScore(v)}\}$.*
2. *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., $E_k = (\min\{10, \ln \sum e^{ExploitabilityScore(v)}\}) \times (\frac{S_k}{NS_k})$. 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 base scores of all the vulnerabilities on a VM. The base score depicts how easy it is for an attacker to exploit the vulnerability and how much damage it may incur. The exponential addition of base scores allows the vulnerability score to incline towards higher base score 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 as defined in [33]. 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. It can be also used as an indicator to demonstrate the security status of a VM, i.e., a VM with higher value of VSI means is easier to be attacked. In order 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.

Figure 7 shows the plotting of *VSI* for these virtual machines before countermeasure selection and application. Figure 8 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.

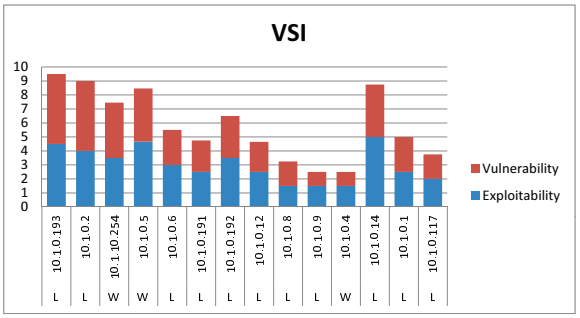


Fig. 7. VM Security Index.

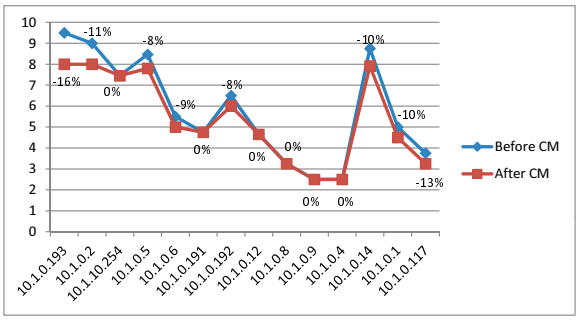


Fig. 8. Change in VM Security Index.

6.1.5 *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 our work 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.

Moreover, we cannot keep aside the case of zero-day attack where the vulnerability is discovered by the attacker but is not detected by vulnerability scanner. In such case, 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.

6.2 NICE System Performance

We evaluate system performance to provide guidance on how much traffic NICE can handle for one cloud server and use the evaluation metric to scale up to a large cloud system. In a real cloud system, traffic planning is needed to run NICE, which is beyond the scope of this paper. Due to the space limitation, we will investigate the research involving multiple cloud clusters in the future.

To demonstrate the feasibility of our solution, comparative studies were conducted on several virtualization approaches. We evaluated NICE based on Dom0 and DomU implementations with mirroring-based and proxy-based attack detection agents (i.e., NICE-A). In mirror-based IDS scenario, we established two virtual networks 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, we have deployed the NICE-A in Dom0 and it removes the traffic duplication function in mirroring and proxy-based solutions.

NICE-A running in Dom0 is more efficient since it can sniff the traffic directly on the virtual bridge. However, in DomU, the traffic need to be duplicated on the VM's virtual interface (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 we used four metrics namely CPU utilization, network capacity, agent processing capacity, and communication delay. We performed the evaluation on cloud servers with Intel quad-core Xeon 2.4Ghz CPU and 32G memory.

We used packet generator to mimic real traffic in the Cloud system. As shown in Figure 9, the traffic load, in form of packet sending speed, increases from 1 to 3000 packets per second. The performance at Dom0 consumes less CPU and the IPS mode consumes the maximum CPU resources. It can be observed that when the packet rate reaches to 3000 packets per second; the CPU utilization of IPS at DomU reaches its limitation, while the IDS mode at DomU only occupies about 68%.

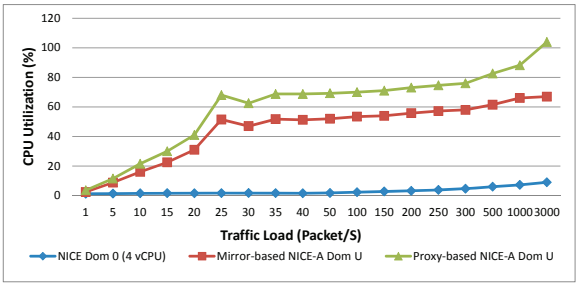


Fig. 9. CPU utilization of NICE-A.

Figure 10, represents the performance of NICE-A in terms of percentage of successfully analyzed packets,i.e., the number of the analyzed packets divided by the total number of packets received. The higher this value is, more packets this agent can handle. It can be observed from the result that IPS agent demonstrates 100% performance because every packet captured by the IPS is

cached in the detection agent buffer. However, 100% 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.

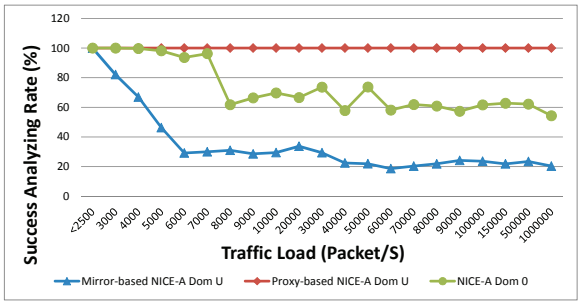


Fig. 10. NICE-A Success Analyzing Rate.

In Figure 11, the communication delay with the system under different NICE-A is presented. We generated 100 consecutive normal packets with the speed of 1 packet per second to test the end-to-end delay of two VMs compared by using NICE-A running in mirroring and proxy modes in DomU and NICE running in Dom0. We record the minimal, average, and maximum communication delay in the comparative study. Results show that the delay of proxy-based NICE-A is the highest because every packet has to pass through it. Mirror-based NICE-A at DomU and NICE-A at Dom0 do not have noticeable differences in the delay. In summary, the NICE-A at Dom0 and Mirror-based NICE-A at DomU have better performance in terms of delay.

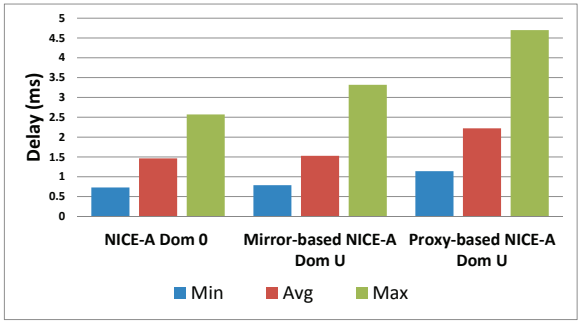


Fig. 11. Network Communication Delay of NICE-A.

From this test we expected to prove the proposed solution, thus achieving our goal "establish a dynamic defensive mechanism based software defined networking approach that involves multiphase intrusion detections". The experiments prove that for a small-scale cloud system, our approach works well. The performance evaluation includes two parts. First, security performance evaluation. It shows that the our approach achieves the design security goals: to prevent vulnerable VMs from being compromised and to 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 us a benchmark for the given hardware setup and shows how much traffic can be handled by using a single detection domain. To scale up to a data center level intrusion detection system, a decentralized approach must be devised, which is scheduled in our future research.

7 Conclusion and Future Work

In this paper, we presented NICE, which is proposed 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. In order 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. Additionally, as indicated in the paper, we will investigate the scalability of the proposed NICE solution by investigating the decentralized network control and attack analysis model based on current study.

Acknowledgement

This work is supported by Hewlett-Packard Lab's Innovation Research Program (IRP) grant and Office of Naval Research (ONR) Young Investigator Program (YIP) award.

References

[1] Coud Sercurity Alliance, "Top threats to cloud computing v1.0," https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf, March 2010.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *ACM Commun.*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[3] B. Joshi, A. Vijayan, and B. Joshi, "Securing cloud computing environment against DDoS attacks," *IEEE Int'l Conf. Computer Communication and Informatics (ICCCI '12)*, Jan. 2012.

[4] H. Takabi, J. B. Joshi, and G. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24–31, Dec. 2010.

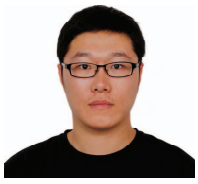
[5] "Open vSwitch project," http://openvswitch.org, May 2012.

[6] 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.

[7] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: detecting malware infection through IDS-driven dialog correlation," *Proc. of 16th USENIX Security Symp. (SS '07)*, pp. 12:1–12:16, Aug. 2007.
[8] G. Gu, J. Zhang, and W. Lee, "BotSniffer: detecting botnet command and control channels in network traffic," *Proc. of 15th Ann. Network and Distributed Sytem Security Symp. (NDSS '08)*, Feb. 2008.
[9] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," *Proc. IEEE Symp. on Security and Privacy*, 2002, pp. 273–284.
[10] "NuSMV: A new symbolic model checker," http://afrodite.itc.it:1024/~nusmv. Aug. 2012.
[11] S. H. Ahmadinejad, S. Jalili, and M. Abadi, "A hybrid model for correlating alerts of known and unknown attack scenarios and updating attack graphs," *Computer Networks*, vol. 55, no. 9, pp. 2221–2240, Jun. 2011.
[12] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: a logic-based network security analyzer," *Proc. of 14th USENIX Security Symp.*, pp. 113–128. 2005.
[13] R. Sadoddin and A. Ghorbani, "Alert correlation survey: framework and techniques," *Proc. ACM Int'l Conf. on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services (PST '06)*, pp. 37:1–37:10. 2006.
[14] L. Wang, A. Liu, and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Computer Communications*, vol. 29, no. 15, pp. 2917–2933, Sep. 2006.
[15] S. Roschke, F. Cheng, and C. Meinel, "A new alert correlation algorithm based on attack graph," *Computational Intelligence in Security for Information Systems*, LNCS, vol. 6694, pp. 58–67. Springer, 2011.
[16] A. Roy, D. S. Kim, and K. Trivedi, "Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees," *Proc. IEEE Int'l Conf. on Dependable Systems Networks (DSN '12)*, Jun. 2012.
[17] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Trans. Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, Feb. 2012.
[18] Open Networking Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, Apr. 2012.
[19] "Openflow." http://www.openflow.org/wp/learnmore/, 2012.
[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[21] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: virtualized cloud infrastructure without the virtualization," *Proc. of the 37th ACM ann. int'l symp. on Computer architecture (ISCA '10)*, pp. 350–361. Jun. 2010.
[22] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," *Proc. of the 13th ACM conf. on Computer and communications security (CCS '06)*, pp. 336–345. 2006.
[23] Mitre Corporation, "Common vulnerabilities and exposures, CVE," http://cve.mitre.org/. 2012.
[24] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system (CVSS)," http://www.first.org/cvss/cvss-guide.html, May 2010.
[25] O. Database, "Open source vulnerability database (OVSDB)," http://osvdb.org/. 2012
[26] NIST, "National vulnerability database, NVD," http://nvd.nist.gov. 2012
[27] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.
[28] X. Ou and A. Singhal, *Quantitative Security Risk Assessment of Enterprise Networks*. Springer, Nov. 2011.
[29] M. Frigault and L. Wang, "Measuring network security using bayesian Network-Based attack graphs," *Proc. IEEE 32nd ann. int'l conf. on Computer Software and Applications (COMPSAC '08)*, pp. 698–703. Aug. 2008.
[30] K. Kwon, S. Ahn, and J. Chung, "Network security management using ARP spoofing," *Proc. Int'l Conf. on Computational Science and Its Applications (ICCSA '04)*, LNCS, vol. 3043, pp. 142–149, Springer, 2004.
[31] "Metasploit," http://www.metasploit.com. 2012.
[32] "Armitage," http://www.fastandeasyhacking.com. 2012.
[33] M. Tupper and A. Zincir-Heywood, "VEA-bility security metric: A network security analysis tool," *Proc. IEEE Third Int'l Conf. on Availability, Reliability and Security (ARES '08)*, pp. 950–957, Mar. 2008.

Chun-Jen Chung received the MS degree in computer science from New York University. He is working toward the Ph.D. degree in School of Computing Informatics and Decision Systems Engineering (CIDSE) at Arizona State University. Prior to that, he worked as software developer in Microsoft and Oracle for several years. His current research interests include computer and network security, security in software defined networking, and trusted computing in mobile devices and cloud computing.

Pankaj Khatkar is currently a Ph.D. student, research associate in School of Computing Informatics and Decision Systems Engineering (CIDSE) at Arizona State University. He received B.E. Computer Engineering in 2010 from University of Mumbai, India. His research interests lie in the area of computer and network security, and mobile cloud computing with emphasis on cloud security.



Tianyi Xing is currently a Ph.D. student in School of Computing Informatics and Decision Systems Engineering (CIDSE) at Arizona State University. He received the B.E. in Telecommunications Engineering from Xidian University and the M.E. in Electronic Engineering from Beijing University of Posts & Telecommunications in 2007 and 2010, respectively. He has been working in Microsoft Research Asia as a research intern from July to December in 2009. His research interests are in secure networking design and implementation, software defined network, and Cloud Computing.

Jeongkeun Lee is a senior researcher in Networking and Communications Lab, HP Labs. He has a Ph.D. in Computer Science and Engineering from Seoul National University. His research interest covers cloud networking, software-defined networking, mobile and wireless communication systems.

Dijiang Huang received his B.S. degree from Beijing University of Posts & Telecommunications, China 1995. He received his M.S., and Ph.D. degrees from the University of Missouri–Kansas City, in 2001 and 2004, respectively. He is currently an Associate Professor in the School of Computing Informatics and Decision System Engineering at the Arizona State University. His current research interests are computer network security, mobile computing, and cloud computing. He is an associate editor of the Journal of Network and System Management (JNSM). He has served as an organizer for many International conferences and workshops. His research is supported by NSF, ONR, Navy, and Hewlett-Packard. He is a recipient of ONR Young Investigator Award 2010.