NICE: NETWORK INTRUSION DETECTION AND COUNTERMEASURE SELECTION IN VIRTUAL NETWORK SYSTEMS

NAGA RAMANI K 1*, SUBHANI SHAIK 2*

1. M.Tech-Student, Dept of CSE, SMWEC, Guntur, AP.
2. Assoc. Prof, Dept of CSE, SMWEC, Guntur, AP.

Abstract: One of the most important concerns is Cloud Security, which has attracted a lot of development and research over the past few years. Notably, vulnerabilities can be discovered by the attackers and virtual machines can be compromised to further deploy a Large scale Distributed Denial of Service (DDoS). Usually, DDoS attacks involve early stage actions such as low frequency vulnerable scanning, multistep exploitation and sacrificing vulnerable virtual machine identified as zombies, and ultimately DDoS attacks through the Sacrificed zombies. The identification of zombie exploration attacks is most difficult task within a cloud system. To avoid compromising of vulnerable virtual machines in the cloud, we proposed a Multi-phase distributed vulnerability detection, measurement and countermeasure selection system known as NICE, designed on analytical models based upon attack graph and re-configurable countermeasures based on virtual network. To build a maintenance and control plane for distributed programmable virtual switches, the propound framework supports to considerably enhance the identification of attacks and alleviate attack consequences. The assessment of system and security describe the effectiveness and efficiency of the propound solution.

Keywords: NICE, Cloud Security, DDoS, Distributed vulnerability detection, Identification of attacks

INTRODUCTION

Recent research has revealed that the users, who migrate towards the cloud, take security as the most important issue. The survey of Cloud Security Alliance (CSA) demonstrates that the exploitation and nefarious use of the cloud-computing environment is considered the top security threat [1] where attackers can abuse vulnerabilities in the cloud system and manipulate cloud system resources to deploy attacks. Vulnerabilities can be identified and patched by the administrators in a

consolidated manner in conventional data centres. Still, the patching aware security holes in the data centre where cloud users have a control over software installed on their managed Virtual Machines may not function efficiently and may violate the Service Level Agreement (SLA). Further, cloud users can set up a vulnerable program on their Virtual Machines, which supports loopholes in cloud security. The challenge here is to create an effective Attack/Vulnerability detection and response system to accurately detect the attacks and lower the effect of security breach in the cloud system.

For better attack detection, we proposed Network Intrusion Detection and Countermeasure selection in virtual network systems (NICE) which incorporates attack graph analytical procedures into the intrusion detection processes. 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 toward the collaborative attack goals, NICE will decide whether to put a VM in network inspection state. 2) Once a VM enters inspection state, Deep Packet Inspection (DPI) is applied, and/or virtual network reconfigurations can be deployed to the inspecting VM to make the potential attack behaviours prominent.

In this work, we proposed the following contributions for the Novel Network Intrusion detection and Countermeasures Election in virtual network systems:

1. 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.

2. NICE employs a novel attack graph approach for attack detection and prevention by correlating attack behavior and suggests effective countermeasures. NICE optimizes
the implementation on cloud servers to minimize resource consumption.

In Part 2 of this document, we presented the concerned work, Part 3 describes the System design and Part 4 describes the effectuation and design of the mechanism. Part 5 describes NICE security measurement, Attack Mitigation, and Countermeasures. Performance evaluation is performed in Part 6 and Part 7 describes Conclusion and future work.

RELATED WORK

This section describes various research areas concerned with NICE, which includes identification of Zombies and prevention, construction of attack graph and security analysis, software defined networks for attack prevention measures. The area of detecting malicious behaviour has been well explored. The work by Duan et al. [2] 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 a host has been compromised.

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. Ou et al. [3] proposed an attack graph tool called MulVAL, 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. 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.

Many attack graph-based alert correlation techniques have been proposed recently. Wang et al. [4] 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. [5] 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.

To countermeasure the attacks according to the likelihood and severity of attacks, several literatures are proposed. Poolsappasit et al. [6] proposed a Bayesian attack graph to address dynamic security risk management problem and applied a genetic algorithm to solve countermeasure optimization problem.

**NICE SYSTEM DESIGN**

The proposed NICE framework is illustrated in Fig. 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 centre connected to software switches on each cloud server (i.e., virtual switches built on one or multiple Linux software bridges).

NICE-A is a software agent implemented in each cloud server connected to the control centre through a dedicated secure channel, which is separated from the normal data packets using Open Flow tunnelling or VLAN approaches. The network controller is responsible for deploying attack countermeasures based on decisions made by the attack analyzer.

Our nomenclatures are based on XEN virtualization technology. Based on offline and online scanning of vulnerabilities, an attack graph is created. Offline scanning can be done by running penetration tests and online real-time vulnerability scanning can be triggered by the network controller (e.g., when new ports are opened and identified by OFSs) or when new alerts are generated by the NICE-A.

After identification of vulnerabilities, the attack analyzer employs the effective countermeasures according to analysis of their effectiveness and the attack graph is recreated. The network controller starts countermeasures by reconfiguring the physical or virtual OFSs.
System Components: In this section, we explain each component of NICE.

**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 (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. 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 is more efficient to scan the traffic in Dom0 because 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 tradeoffs of installing NICE-A in Dom0 and DomU.
VM Profiling

Virtual machines in the cloud can be profiled to get precise information about their state, services running, open ports, and so on. One major factor that counts toward 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 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. Therefore, 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.

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. 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 Xen Server). 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) [9], Common Vulnerabilities and Exposures List (CVE) [8], and NIST National Vulnerability Database (NVD) [10].
Network Controller:

The network controller is responsible for collecting network information of the current Open Flow 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 [11], 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 Open Flow protocol [7], 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.

NICE Security Measurements and Countermeasures

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 [12] 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, NR ⊆ ND), the prior probability is assigned on the likelihood of a threat source becoming active and the difficulty of the vulnerability to be exploited. We use GV to denote the prior 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.

In the attack graph, the relations between exploits can be disjunctive or conjunctive according to how they are...
related through their dependency conditions [13]. 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. 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.

For any attack step node, \( n \in N_c \) with immediate predecessors set \( W=\text{parent} (n) \)

\[
\text{Pr}(n) = \text{Pr} \left( \frac{\pi}{W} \right) \times \prod_{s \in W} \text{Pr}(s);
\]

**COUNTERMEASURE SELECTION**

The figure below shows the algorithm for 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 \( CM \). Definition 3 (Countermeasure Pool). A countermeasure pool \( CM = \{ cm_1, cm_2, \ldots, cm_m \} \) is a set of countermeasures. Each \( cm \in CM \) is a tuple \( cm = (\text{cost}, \text{intrusiveness}, \text{condition}, \text{effectiveness}) \).

Algorithm 1. Countermeasure Selection:

**Require:** Alert, \( G (E, V) \), CM

Let \( \nu_{\text{Alert}} = \text{Source node of the Alert} \)

if Distance_to_target(\( \nu_{\text{Alert}} \)) > threshold then

Update_ACG

Return

End if

Let \( T = \text{Descendent} (\nu_{\text{Alert}}) \cup \nu_{\text{Alert}} \)

Set \( \text{Pr}(\nu_{\text{Alert}}) = 1 \)

Calculate_Risk_Probe (\( T \))

Let benefit \( [ |T|, |CM| ] = \emptyset \)

For each \( t \in T \) do

For each \( cm \in CM \) do

If \( cm.\text{condition} (t) \) then

\[
\text{Pr}(t) = \text{Pr}(t) \times (1 - \text{cm.\text{effectiveness}})
\]

Calculate_Risk_Probe (Descendent (\( t \)))

Benefit \( [t, cm] = \Delta \text{Pr}(\text{target_node}) \).

End if

End for

End for

Let ROI \( [ |T|, |CM| ] = \emptyset \)

For each \( t \in T \) do

For each \( cm \in CM \) do

\[
\text{ROI} [t, cm] = \frac{\text{benefit}[t, cm]}{\text{cost}. cm + \text{intruness}. cm}.
\]

End if

End for

Update_SAG and Update_ACG

Return Select_Optimal_CM (ROI)

The above algorithm describe the following procedure: 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

---

**International Journal of Computers, Electrical and Advanced Communications Engineering**

Vol.1 (11), ISSN: 2250-3129, JAN – JUL' 2017

PP: 89 - 100
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.

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 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. 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.

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.

From this experiment, we expected to prove the proposed solution, thus achieving our goal “establish
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 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.

CONCLUSION AND FUTURE WORK

We proposed NICE framework for identification and extenuation of threats in a cloud virtual system environment. To perform discovery and identification of attacks, we used attack graph model in NICE. The detection accuracy is increased by investigating the use of programmability of software switches based solutions in the proposed framework. The proposed solution can prevent the risk of exploitation of cloud system and abuse by internal and external attackers according to the system performance evaluation. In this work, NICE discovers the IDS approach of network to counteract exploitation attacks of zombies. We need host based IDS solutions to be comprised and to cover the whole IDS spectrum in the cloud environment to further improve the accuracy of detection. Our investigation in the future work includes incorporating of host based IDS solution and the scalability of the system proposed by exploring the model of attack analysis and network control according to the current work.

ACKNOWLEDGEMENT

Authors are thankful to Dr. O. Nagaraju, Professor, Dept. of computer science, Macharla Degree College for giving continues support and encouragement to carry out this work.
Authors are also thankful to the reviewer for critically going through the manuscript and giving valuable suggestions for the improvement of manuscript.

REFERENCES


