Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 8, August 2014, pg.372 – 378

RESEARCH ARTICLE

NICE

Renuka Devi, B.E, (M.Tech), **C.H.Kiran**, B.Tech, M.Tech
Computer Science and Engineering, JNTU, Hyderabad, India
renukamalige@gmail.com , kiran.30.aug@gmail.com

ABSTRACT- Recently, Cloud security is one of most important issues that have 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 Open Flow 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.

I. 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 Counter measures Election 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) 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.

II. LITERATURE SURVEY

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 *etal*. 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 Mul VAL'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.

III. EXISTING SYSTEM

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 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. 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. 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.

Disadvantage

1. No detection and prevention framework in a virtual networking environment.
2. Not accuracy in the attack detection from attackers.

IV. OBJECTIVES

Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective**. Main Modules:-**

1. User Module: In this module, Users are having authentication and security to access the detail which is presented in the ontology system. Before accessing or searching the details user should have the account in that otherwise they should register first.

2. Countermeasure Selection:

Countermeasure Selection To illustrate how NICE works, let us consider for example, an alert is generated for node 16 (*vAlert* = 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*.

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 ash can be modelled 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.

4. 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 zeroday 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.

V. PROPOSED SYSTEM

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.

Advantages:

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.

VI. SYSTEM ANALYSIS

NICE SYSTEM DESIGN

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

6.1 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 built on one or multiple Linux software bridges). NICEA 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 real time vulnerability scanning can be triggered by the network controller (e.g., when new ports are opened and identified by Open Flow 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.

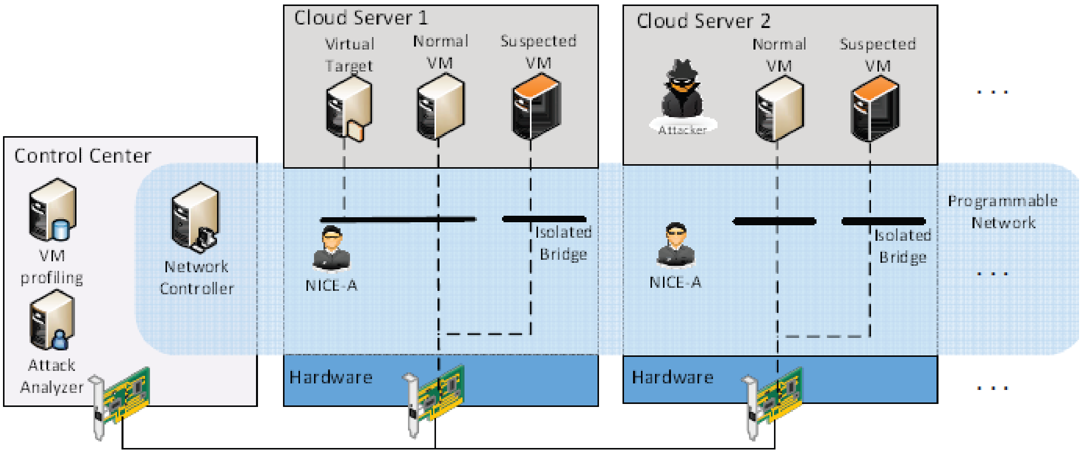

Fig. 1. NICE architecture within one cloud server cluster.

VII. CONCLUSION

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

REFERENCES

[1] 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 logicbased 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.