International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

NICE-D: A Modified Approach for Cloud Security

Nikita T. Ramteke[1], Dr. Yashwant V. Chavan[2]

[1]Department of Computer Engineering, Savitribai Phule Pune University, Pune-411038, Maharashtra, India

[2]Principal at Padmabhushan Vasantdada Patil Institute of Technology,
Savitribai Phule Pune University, Bavdhan, Pune-411021, Maharashtra, India

Abstract: ***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). 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 [1], 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. Existing NICE model uses signature based IDS i.e. SNORT [1], In this project work we improved the intrusion detection accuracy of NICE by using dynamic intrusion detection system (NICE-D). Dynamic IDS monitors the incoming traffic flow and anomalous time slot, and accordingly generates the new signature to identify the future intrusions over cloud system. The system and security evaluations demonstrate the efficiency and effectiveness of the proposed solution.***

Keyword: DDos, Iaas, NICE, SNORT

1. Introduction

The main aim of this project is to prevent the vulnerable virtual machines from being compromised in the cloud server using multi-phase distributed vulnerability detection, measurement, and countermeasure selection mechanism called NICE-D.

In 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, 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 [3]. 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 a cloud system [8] where the infra-structure 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.

2. Related Work

Cloud users can install vulnerable software on their VMs, [1] 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 [1]. 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.

Problems with existing system:

1) No detection and prevention framework in a virtual networking environment.
2) Not accuracy in the attack detection from attackers.
3) Use of signature based intrusion detection i.e. SNORT. Which will never detect and generate the alarm for newly created attack over cloud?

3. Proposed System

In this article, we propose NICE-D (Network Intrusion detection and Countermeasure selection in virtual network systems with Dynamic IDS) to establish a defense-in-depth intrusion detection framework. For better attack detection, NICE-D incorporates attack graph analytical procedures into the intrusion detection processes. In NICE-D, we improve existing intrusion detection algorithms; NICE-D employs a reconfigurable virtual networking approach to detect and counter the attempts to compromise VMs, thus preventing zombie VMs.

The earlier contributions of NICE [1] 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.

Advanced Contributions:

1. Use of Dynamic intrusion detection system.
2. We will be able to improve the false alarm rate.
3. Proposed system will detect the future intrusions as it is based on creating rules at operational environment.

4. System Architecture

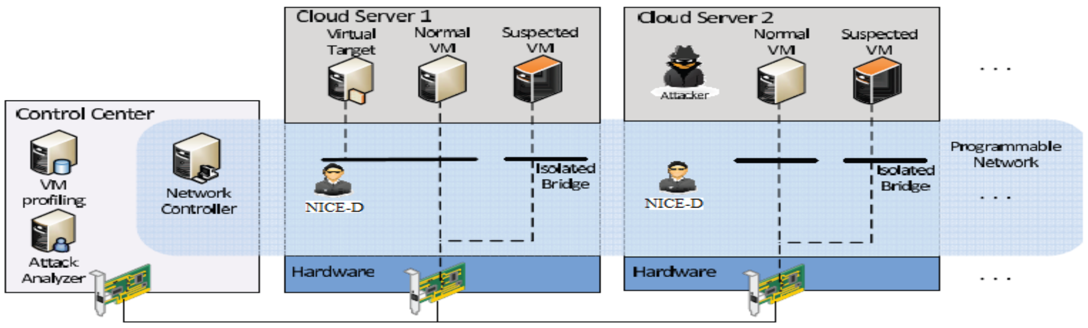

Figure 1: System architecture

5. Algorithm Used

5.1 Dynamic IDS

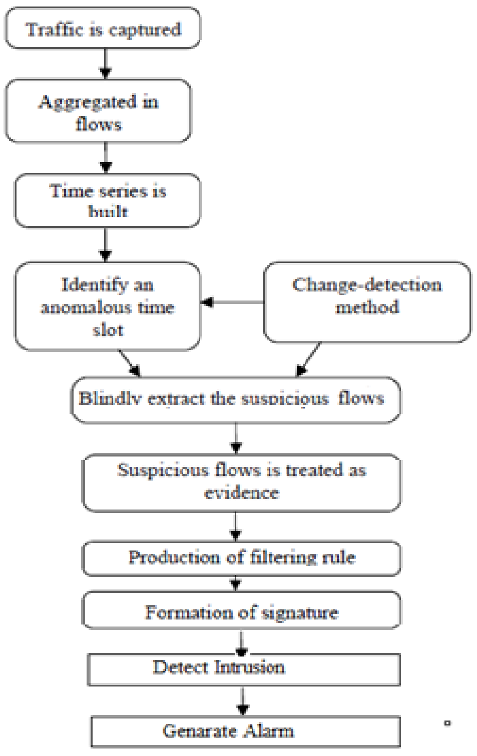
Traffic is captured
Aggregated in flows
Time series is built
Identify an anomalous time slot
Change-detection method
Blindly extract the suspicious flows
Suspicious flows is treated as evidence
Production of filtering rule
Formation of signature
Detect Intrusion
Genarate Alarm

Figure 2: Dynamic Intrusion detection

Network attack detection is the very challenging task for the network operator in today's internet. It is being challenging task because network attack are moving targets, they are not Steady. Attacker may launch every time new attack, which is not seen previously. So there is the need of detection system that will be able to detect various attacks of different range and with variety of characteristic [3]. This detection system should use the very less amount of previous knowledge or no use of any type of information at all. In research literature and commercial detection systems there are two different approaches namely signature based detection and anomaly detection for detection of attack. Signature based detection relies on the use of specifically known pattern of unauthorized behavior. It depends on sniffing packet. It monitors and compares the packet with predetermined attack patterns which are also known as signature. That is signature based detection system is used to detect those attacks which they are program to alert on. This detection system cannot defend against unknown attack. On the contrary, Anomaly detection builds normal operation traffic profile which detects anomalies as the activities that deviate from the baseline [5]. Thus it uncover abnormal pattern of behavior. This detection system can detect new, previously unseen attack. But as it has to build normal operation profile and it require training to construct normal operation profile and hence it is being time consuming. The new services and applications are constantly emerging and this type of detection is being difficult. In this paper the approach is used to detect both known as well as unknown attack. This is done by the production of signature that determine the attack in an online basis. Algorithm that is being applied for characterizing attack. The algorithm follows in three consecutive stages. Firstly, using a temporal sliding-window approach, traffic is captured and it is aggregated in flows. This is done using different levels of traffic aggregation. For simple traffic metrics such as number of bytes, flows in each time slot, time series are built. And any change-detection method is applied to identify an anomalous time slot.

In the second stage unsupervised detection algorithm begins. It uses the output of first stage as the input. Filtering rule helps to provide characteristic of attack[8] .But when the network operator deal with the unknown attack, the characterization of attack may became much more difficult as it require good, easy, simple information as the input .To remove this issue, new traffic signature is developed by combining relevant filtering rules. This signature will detect the attack coming in future; this is the important step toward autonomous security.

5.2 Alert Correlation Algorithm

```
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 $S$
```

An attack graph is a modeling tool to illustrate all possible multistage, multi-host attack paths that are crucial to understand threats and then to decide appropriate countermeasures [2]. In an attack graph, each node represents either precondition or consequence of an exploit. The actions are not necessarily an active attack because 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 Ou et al. [12] and define as Scenario Attack Graph (SAG).

Definition 1 (SAG). An SAG 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 NR 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 (Host,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. NR represents the root node of the SAG.

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 (ACG). An ACG is a three tuple ACG =(A,E,P), where

1. A is a set of aggregated alerts. An alert a ϵ A is a data structure (*src*,*dst*,*cls*,*ts*) representing source IP address, destination IP address, type of the alert, and time stamp of the alert respectively.
2. Each alert a maps to a pair of vertices (v_c,vd) in SAG using map(a) function, i.e.,map(a): a→{(v_c,v_d) | (*a.src* ϵ v_c.Hosts) /\ (*a.dst* ϵ v_d.Hosts) /\ (*a.cls* = v_c.*vul*)) }
3. E is a set of directed edges representing correlation between two alerts ða; a0Þ if criteria below satisfied:
a. (a.ts<a’.ts) /\ (a’.ts-a.ts < threshold)

b.∃(v_d,v_c) ϵ E_{pre}:(a.dst ϵ v_d.Hosts /\ a’.src ϵ v_c.Hosts)

4. P is set of paths in ACG. A path S_i ⊂ 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 time stamp within a specified window are aggregated as Meta Alerts. Each ordered pair (*a,a’*) in ACG maps to two neighbor vertices in SAG with time stamp difference of two alerts within a predefined threshold. ACG shows dependency of alerts in chronological order and 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 Si ⊂ 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 Si. For every alert ac that is received from the IDS, it is added to ACG if it does not exist. For this new alert ac, the corresponding vertex in the SAG is found by using function map(ac) (line 3). For this vertex in SAG, alert related to its parent vertex of type NC is then correlated with the current alert ac (line 5). This creates a new set of alerts that belong to a path Si in ACG (line 8) or splits out a new path Siþ1 from Si with subset of Si before the alert a and appends ac to Siþ1 (line 10). In the end of this algorithm, the ID of ac will be added to alert attribute of the vertex in SAG.

6. Countermeasure Selection Algorithm

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:
                $benefit[t, cm] = \Delta Pr(target\_node).$    (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:
            $ROI[t, cm] = \frac{benefit[t, cm]}{cost.cm + intrusiveness.cm}.$    (8)
23:     end for
24: end for
25: Update_SAG and Update_ACG
26: return Select_Optimal_CM($ROI$)
```

Countermeasure Selection:-

Above algorithm 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_{Aler}t that corresponds to the alert generated by a NICE-D. 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 (lines 7 and 8). Now, for all t ϵ T the applicable countermeasures in CM are selected and new probabilities are calculated according to the effectiveness of the selected countermeasures (lines 13 and 14). The change in probability of target node gives the benefit for the applied countermeasure. 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 O(|V| x|CM|), where |V |is the number of vulnerabilities and |CM| represents the number of countermeasures.

7. NICE System Component

- Nice-D
- VM Profiling
- Attack Analyzer
- Network Controller

7.1 Nice-D

The NICE-D is a Network-based Intrusion Detection System (NIDS) with dynamic IDS installed in each cloud server. It scans the traffic going through the bridges that control all the traffic among VMs and in/out from the physical cloud servers. 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. It's more efficient to scan the traffic in cloud server since all traffic in the cloud server needs go through it; however our design is independent to the installed VM. The false alarm rate could be reduced through our architecture design.

7.2 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. 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.

7.3 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. 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. NICE attack graph is constructed based on the following information: *Cloud system information*, *Virtual network topology and configuration information*, *Vulnerability information*

7.4 Network Controller

The network controller is a key component to support the programmable networking capability to realize the virtual network reconfiguration. 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 Open Flow network and provides input to the attack analyzer to construct attack graphs. 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. 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.

8. Conclusion

NICE-D for cloud system work better than the existing system NICE-A. NICE-A covered all possible work for the VM over cloud but did not considered the improvement to the intrusion detection algorithm. In this paper we proposed the new dynamic intrusion detection algorithm which performs better than the existing signature based SNORT.

References

[1] Chun-Jen Chung, Student Member, IEEE, Pankaj Khatkar, Student Member, IEEE, Tianyi Xing, Jeongkeun Lee, Member, IEEE, and Dijiang Huang S enior Member, IEEE-"NICE: Network Intrusion Detection and Countermeasure Selection in Virtual Network Systems"- IEEE TRANSACTIONS ON DEPEDABLE AND SECURE COMPUTING, 2013.
[2] Coud Sercurity Alliance, "Top Threats to Cloud Computing v1.0," https://cloudsecurityalliance.org/topthreats/csathreats. v1.0.pdf, Mar. 2010.
[3] 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 Comm., vol. 53, no. 4, pp. 50-58, Apr. 2010.
[4] B. Joshi, A. Vijayan, and B. Joshi, "Securing Cloud Computing Environment Against DDoS Attacks," Proc. IEEE Int'l Conf. Computer Comm. and Informatics (ICCCI '12), Jan. 2012.
[5] H. Takabi, J.B. Joshi, and G. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," IEEE Security and Privacy, vol. 8, no. 6, pp. 24-31, Dec. 2010.
[6] "Open vSwitch Project," http://openvswitch.org, May 2012.
[7] 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.
[8] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection through

Licensed Under Creative Commons Attribution CC BY

IDS-driven Dialog Correlation," Proc. 16th USENIX Security Symp. (SS '07), pp. 12:1-12:16, Aug. 2007.

[9] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic," Proc. 15th Ann. Network and Distributed Sytem Security Symp. (NDSS '08), Feb. 2008.

[10] Michael Armbrust, Armando Fox, Rean Griffith "Above the Clouds: A View of Cloud Computing" Tech. Rep. UCB/EECS-2009-28, EECS Department, U.C. Berkeley, Feb 2012.

[11] Ms. Parag K. Shelke, Ms. Sneha Sontakke, Dr. A. D. Gawande "Intrusion Detection System for Cloud Computing "International Journal of Scientific & Technology Research Volume 1, Issue 4, May 2012.

[12] J. Sasi Devi, R. Sugumar "Host Based Intrusion Detection to Prevent VirtualNetwork System from Intruders in Cloud" International Journal of Science and Research (IJSR) 2014

Licensed Under Creative Commons Attribution CC BY