ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Detecting and Mitigating Collaborative Attacks using NICE in Virtual Network systems

Y.Srilakshmi
M.Tech Student,
Department of CSE,
School of Technolog,
GITAM University.

Mr.S.D.Vara Prasad
Assistant Professor,
Department of CSE,
School of Technolog,
GITAM University.

ABSTRACT:
The cloud computing has increased in many organizations. It provides many benefits in terms of low cost and accessibility of data. Ensuring the security of cloud computing is a major factor in the cloud computing environment, as users often store sensitive information with cloud storage providers but these providers may be untrusted. In this project we propose anIntrusion Detection and Countermeasure in Virtual Network Systemsmechanism called NICE to prevent vulnerable virtual machines from being compromised in the cloud. NICE detects and mitigates collaborative attacks in the cloud virtual networking environment. 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..

INDEX TERMS:
Cloud Computing, Intrusion Detection, Network Security, Zombie Detection.

I.INTRODUCTION :
Cloud Security Alliance (CSA) survey shows that among all security issues, abuse and nefarious use of cloud computing is considered as the top securitythreat, in which attackers can exploit vulnerabilities in clouds and utilize cloud system resources to deploy at-tacks. In traditional data centers, where system admin- istrators have full control over the host machines, vul- nerabilities 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).

[4] Furthermore, cloud users can install vulner- able software on their VMs, which essentially contrib- utes 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.To establish a defense-in-depth Intrusion Detection Framework, We Propose NICE. 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. [5] 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.Actually, 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 vulner- abilities within a cloud server to establish Scenario At- tack Graph (SAGs), and then based on the severity of identified vulnerability towards the collaborative at- tack 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 re- configurations can be deployed to the inspecting VM to make the potential attack behaviors prominent.

2. NORMALIZATION TRADEOFFS:

When designing traffic normalizes, we are faced with a set of tradeoffs, which can be arranged along several axes:

• Extent of normalization vs. protection

• impact on end-to-end semantics (service models)

• impact on end-to-end performance amount of state held

• work offloaded from the NIDS

Generally speaking, as we increase the degree of normalization and protection, we need to hold more state; performance decreases both for normalize and for end-to-end flows; and we impact end-to-end semantics more. Our goal is not to determine a single "sweet spot," but to understand the character of the tradeoffs, and, ideally, design a system that a site can tune to match their local requirements.

Normalization vs. protection:

As normalize is a "bump in the wire," the same box performing normalization can also perform firewall functionality. For example, normalize can prevent known attacks, or shut down access to internal machines from an external host when the NIDS detects a probe or an attack. In this paper we concentrate mainly on normalization functionality, but will occasionally discuss protective functionality for which normalize is well suited.

End-to-end semantics:

As much as possible, we would like normalize to preserve the end-to-end semantics of Well-behaved network protocols, whilst cleaning up misbehaving traffic. Some packets arriving at normalize simply cannot be correct according to the protocol specification, and for these there often is a clear normalization to apply. For example, if two copies of an IP fragment arrive with the same fragment offset, but containing differ- ent data, then dropping either of the fragments or dropping the whole packet won't undermine the cor- rect operation of the particular connection. Clearly the operation was already incorrect. [6]However, there are other packets that, while perfectly legal according to the protocol specifications, may still cause ambiguities for the NIDS. For example, it is perfectly legitimate for a packet to arrive at normalize with a low TTL. However, per the discussion in the Introduction, the NIDS cannot be sure whether the packet will reach the destination.

A possible normalization for such packets is to increase its TTL to a large value.1 for most traffic, this will have no adverse effect, but it will break diagnos- tics such as trace route, which rely on the semantics of the TTL field for their correct operation. Normaliza- tions like these, which erode but do not brutally vio- late the end-to-end protocol semantics, present a ba- sic tradeoff that each site must weigh as an individual policy decision, depending on its user community, [7] performance needs, and threat model. In our analysis of different normalizations, we place particular empha- sis on this tradeoff, because we believe the long-term utility of preserving end-to-end semantics is often un- derappreciated and at risk of being sacrificed for short- term expediency.

Impact on end-to-end performance:

Some normalization is performed by modifying packets in a way that removes ambiguities, but also adversely affects the performance of the protocol being normal- ized. There is no clear answer as to how much impact on performance might be acceptable, as this clearly depends on the protocol, local network environment, and threat model.

State holding:

A NIDS system must hold state in order to understand the context of incoming information. One form of at- tack on a NIDS is a state holding attack, whereby the attacker creates traffic that will cause the NIDS to in- stantiate state (see _ 4.2 below). If this state exceeds the NIDS's ability to cope, the attacker may well be able to launch an attack that passes undetected. This is possible in part because a NIDS generally operates passively, and so "fails open." A normalize also needs to hold state to correct ambi- guities in the data flows. Such state mightinvolve keep- ing track of unacknowledged TCP segments, or holding IP fragments for reassembly in normalize. However, unlike the NIDS, normalize is in the forwarding path, and so has the capability to "fail closed" in the pres- ence of state holding attacks. Similarly, the normalize can perform "triage" amongst incoming flows:

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

if the normalize is near state exhaustion, it can shut down and discard state for flows that do not appear to be making progress, whilst passing and normalizing those that do make progress. [8] The assumption here is that without complicity from internal hosts (see below), it is difficult for an attacker to fake a large number of active connections and stress normalize state holding. But even given the ability to perform triage, if a normalize operates fail-closed then we must take care to assess the degree to which an attacker can exploit state hold- ing to launch a denial-of-service attack against a site, by forcing the normalize to terminate some of the site's legitimate connections.

Inbound vs. outbound traffic:

The design of the Bionetwork intrusion detection system assumes that it is monitoring a bi-directional stream of traffic, and that either the source or the destination of the traffic can be trusted [3]. However it is equally effective at detecting inbound or outbound attacks. The addition of normalize to the scenario potentially introduces an asymmetry due to its location— normalize protects the NIDS against ambiguities by processing the traffic before it reaches the NIDS (Figure 2).

Thus, an internal host attempting to attack an external host might be able to exploit such ambiguities to evade the local NIDS. If the site's threat model includes such attacks, either two normalize may be used, one on either side of the NIDS, or a NIDS integrated into a single normalize.

Finally, we note that if both internal and external hosts in a connection are compromised, there is little any NIDS or normalize can do to prevent the use of encrypted or otherwise covert channels be- tween the two hosts. Whilst a normalize will typically make most of its modifications to incoming packets, there may also be a number of normalizations it ap- plies to outgoing packets.

These normalizations are to ensure that the internal and external hosts' protocol state machines stay in step despite the normalization of the incoming traffic. It is also possible to normalize outgoing traffic to prevent unintended information about the internal hosts from escaping ([3], and see _ 5.1 below).

Protection vs. offloading work:

Although the primary purpose of normalize is to prevent ambiguous traffic from reaching the NIDS where it would either contribute to a state explosion or allow evasion, normalize can also serve to offload work from the NIDS. For example, if the normalize discards packets with bad checksums, then the NIDS needn't spend cycles verifying checksums.

3. REAL-WORLD CONSIDERATIONS:

Due to the adversarial nature of attacks, for security systems it is particularly important to consider not only the principles by which the system operates, but as much as possible also the "real world" details of operating the system. In this section, we discuss two such issues, the "cold start" problem, and attackers targeting the normalize operation.

3.1 Cold start:

It is natural when designing a network traffic analyzer to structure its analysis in terms of tracking the progression of each connection from the negotiation to begin it, through the connection's establishment and data transfer operations, to its termination. Unless carefully done, however, such a design can prove vulnerable to incorrect analysis during a cold start. That is, when the analyzer first begins to run, it is confronted with traffic from already-established connections for which the analyzer lacks knowledge of the connection characteristics negotiated when the connections were established.

For example, the TCP scrubber [1] requires a connection to go through the normal start-up handshake. However, if a valid connection is in progress, and the scrubber restarts or otherwise loses state, then it will terminate any connections in progress at the time of the cold start, since to its analysis their traffic exchanges appear to violate the protocol semantics that require each newly seen connection to begin with a start-up handshake. The cold-start problem also affects the NIDS itself. If the NIDS restarts, the loss of state can mean that pre- viously monitored connections are no longer monitor able because the state negotiated at connection set- up time is no longer available.

As we will show, tech- niques required to allow clean normalize restarts can also help a NIDS with cold start (_ 6.2). Finally, we note that could start is not an unlikely "corner case" to deal with, but instead an on-going issue for normalize and NIDS alike. First, an attacker might be able to force a cold start by exploiting bugs in either system. Second, from operational experience we know that one cannot avoid occasionally restarting a monitor system, for ex- ample to reclaim leaked memory or update configura- tion files. Accordingly, a patient attacker who keeps a connection open for a long period of time can ensure a high probability that it will span a cold start.

3.2 Attacking the Normalize:

Inevitably we must expect the normalize itself to be the target of attacks. Besides complete subversion, which can be prevented only though good design and coding practice, two other ways normalize can be attacked are state holding attacks and CPU overload attacks.

State holding attacks:

Some normalization are stateless. For example, the TCP MSS option (Maximum Segment Size) is only allowed in TCP SYN packets. If a normalize sees a TCP packet with an MSS Option but no SYN flag, then this is illegal; but even so, it may be unclear to the NIDS what the receiving host will do with the option, since its TCP implementation might incorrectly still honor it. Because the use of the option is illegal, normalize can safely remove it (and adjust the TCP checksum) without needing to instantiate any state for this purpose. Other normalizations require normalize to hold state. For example, an attacker can create ambiguity by sending multiple copies of an IP fragment with different payloads. While normalize can remove fragment based ambiguities by reassembling all fragmented IP packets itself before forwarding them (and if necessary re-fragmenting correctly), to do this, normalize must hold fragments until they can be reassembled into a complete packet. An attacker can thus cause normalize to use up memory by sending many fragments of packets without ever sending enough to complete a packet. This particular attack is easily defended against by simply bounding the amount of memory that can be used for fragments, and culling the oldest fragments from the cache if the fragment cache fills up.

Because fragments tend to arrive together, this simple strategy means an attacker has to flood with a very high rate of fragments to cause a problem. Also,as IP packets are unreliable, there's no guarantee they arrive anyway, so dropping the occasional packet doesn't break any end- to-end semantics. More difficult to defend against is an attacker causing normalize to hold TCP state by flood- ing in, for example, the following ways:

1.Simple SYN flooding with SYNs for multiple connections to the same or to many hosts.

2.ACK flooding. A normalize receiving a packet for which it has no state might be designed to Then instantiate state (in order to address the "cold start" problem).

3.Initial window flooding. The attacker sends a SYN to a server that exists, receives a SYN-ACK, and then floods data without waiting for a response. A normalize would normally temporarily store unacknowledged text to prevent inconsistent retransmissions.

Our strategy for defending against these is twofold. First, normalize knows whether or not it's under attack by monitoring the amount of memory it is consuming. If it's not under attack, it can instantiate whatever state it needs to normalize correctly. If it believes it's under attack, it takes a more conservative strategy that is designed to allow it to survive, although some legitimate traffic will see degraded performance. In general our aim when under attack is to only instantiate TCP connection state when we see traffic from an internal (and hence trusted) host as this restricts state holding attacks on normalize to those actually involving real connections to internal hosts. Note here that normalize is explicitly not attempting to protect the internal hosts from denial-of-service attacks; only to protect itself and the NIDS.

CPU over load attacks:

An attacker may also attempt to overload the CPU on normalize. However, unlike state holding attacks, such an attack cannot cause normalize to allow an ambiguity to pass. Instead, CPU overload attacks can merely cause normalize to forward packets at a slower rate than it otherwise would.

In practice, we find that most normalization are rather cheap to perform (_ 7.2), so such attacks need to concentrate on the normalizations where the attacker can utilize computational complexity to their advantage. Thus, CPU utilization attacks will normally need to be combined with state holding attacks so that normalize performs an expensive search in a large state-space. Accordingly, we need to pay great attention to the implementation of such search algorithms, with extensive use of constant-complexity hash algorithms to locate matching state. An additional difficulty that arises is the need to be opportunistic about garbage collection, and to apply algorithms that are low cost at the possible expense of not being completely optimal in the choice of state that is reclaimed.

ALGORITHM:

When an Attacker Attacks the Server by using a User Account, Attacker can Deploy Multiple Levels of Malwares to the Server, If and only if he can Access to the Server, but in Existing System it's Hard to Detect the Attacker because of Server Cloud Service While in Proposed, When an Attacker Attacks the Server using User Account, the Attack Analyzer can Detect the Attacker and Send the Warning to Administrator that User[Attacked by the Zombie] try to Access to Other Users Account to Deploy the Multiple Levels of Malware and Admin waits for Maximum Attempts and then Admin Blocks him Permanently using Scenario Attack Graph.

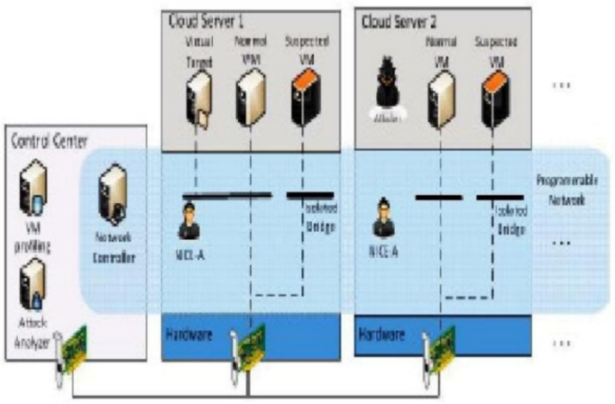


Fig 1: Designed NICE Architecture

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 model 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.

Algorithm:

Alert Correlation

```
Require: alert ac, SAG, ACG

1:if (ac is a new alert) then

2:create node ac in ACG

3:n1  vc map (ac)

4:for all n2  parent (n1) does

5:create edge (n2.alert, ac)

6:for all Si containing a do

a is the last element in Si then ppend ac to Si
lsecreate path Si+1 = {sub set (Si, a), ac}

nd if end for
add ac to n1.alert

14:end for

15:end if

16: return S
```

above method for utilizing SAG and ACG together so as to predict an attackers 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.

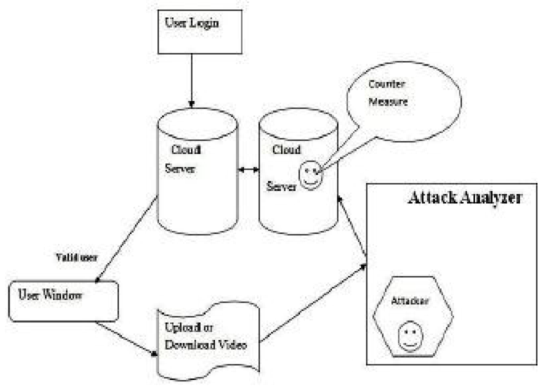
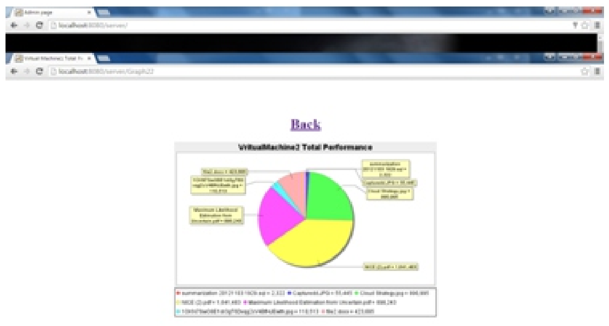


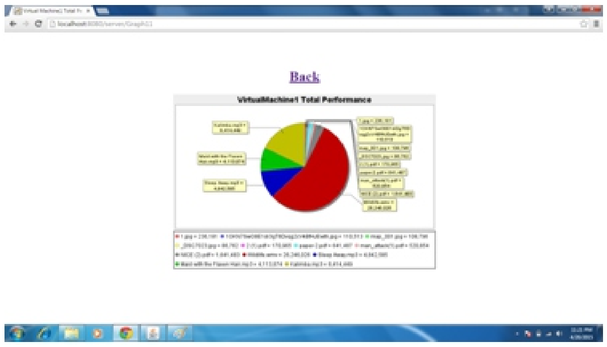
Fig 2: Counter-Measure Model

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 electing 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.

4. 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 solu- tion can significantly reduce the risk of the cloud sys- tem 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. Addi- tionally, as indicated in the paper, we will investigate the scalability of the proposed NICE solution by inves- tigating the decentralized network control and attack analysis model based on current study.





REFERENCES:

[1]G. R. Malan, D.Watson, F. Jahanian and P. Howell, “Transport and Application Protocol Scrubbing”, Proceedings of the IEEE INFOCOM 2000 Conference, Tel Aviv, Israel, Mar. 2000.
[2]V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time”, Computer Networks, 31(23-24), pp. 2435-2463, 14 Dec 1999.
[3]M. Smart, G.R. Malan and F. Jahanian, “Defeating TCP/IP Stack Fingerprinting,” Proc. USENIX Security Symposium, Aug. 2000.
[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]C. Kent and J. Mogul, “Fragmentation Considered Harmful,” Proc. ACM SIGCOMM, 1987.
[7]E. Kohler, R. Morris, B. Chen, J. Jannotti and M.F. Kaashoek, “The Click modular router,”ACM Transactions on Computer Systems, 18(3), pp. 263–297, Aug. 2000.
[8] G. R. Malan, D.Watson, F. Jahanian and P. Howell, “Transport and Application Protocol Scrubbing”, Proceedings of the IEEE INFOCOM 2000 Conference, Tel Aviv, Israel, Mar. 2000.