IJERT
ISSN : 2278-0181
International Journal of Engineering Research & Technology
Publish & Find Papers @ www.ijert.org
BROWSE
OPEN ACCESS
Call for Papers

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

BrindhaKundavaran
Computer Science and Engineering
Sri Venkateshwara College of Engineering For Women(SVEW)
Tirupati-517501

G . TagoreSai Prasad
Computer Science and Engineering
Sri Venkateshwara College of Engineering For Women(SVEW)
Tirupati-517501

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

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

1 INTRODUCTION

Recent studies have shown that users migrating tothe 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 vulner-abilities 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 sys-tem administrator in a centralized manner. 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 [3].Such attacks are more effective in the cloud environment In this article, we propose NICE (Network Invasion detection and Countermeasure sElection in virtual net-work systems) to establish a defense-in-depth invasion detection framework. For better attack detection, NICE incorporates attack graph analytical procedures into the invasion detection processes. We must note that the design of NICE does not intend to improve any of the existing invasion 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 invasion detection agent (NICE-A) on each cloud server to capture and analyze cloud traffic. (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 con-struct 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. 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 invasion detection and prevention frameworkin a virtual networking environment

• NICE incorporates a software switching solutionto quarantine and inspect suspicious VMs for further investigation and protection.

• NICE employs a novel attack graph approach for attack detection and prevention by correlating attack Behavior.

• NICE optimizes the implementation on cloud servers to minimize resource consumption.

2 RELATED WORK

In this section, we present literatures of several highly related research areas to NICE, including: zombie detection and prevention, attack graph construction and security analysis, and software defined networks for attack countermeasures. The area of detecting malicious behavior has been well explored. 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. A technique based on a modified sym-bolic model checking NuSMV [10] and Binary Decision Diagrams (BDDs) to construct attack graph. This model can generate all possible attack paths, however, the scal-ability is a big issue for this solution. The assumption of monotonicity, which states that the precondition of a given R exploit is never invalidated by the successful application of another exploit. In other words, attackers never need to backtrack. Invasion 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 invasion 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. 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. 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.

After knowing the possible attack scenarios, applying countermeasure is the next important task. Several solutions have been proposed to select optimal coun-termeasures based on the likelihood of the attack path and cost benefit analysis.An attack countermeasure tree (ACT) to consider attacks and countermeasures together in an attack tree structure. 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. 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 moni-tored and configured. We take advantage of flow-based switching (OVS) and network controller to apply the selected network countermeasures in our solution.

3. NICE MODELS

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

3.1 Threat Model

In our attack model, we assume that an attacker can be located either outside or inside of the virtual networking system. Our protection model focuses on virtual-network-based attack detection and reconfiguration solutions to improve the resiliency to zombie explorations. Our work

does not involve host-based IDS and does not address how to handle encrypted traffic for attack detections. Our proposed solution can be deployed in an Infrastructure-as-a-Service (IaaS) cloud networking system, and we assume that the Cloud Service Provider (CSP) is benign. We also assume that cloud service users are free to install whatever operating systems or applications they want, even if such action may intro-duce vulnerabilities to their controlled VMs. The issue of a malicious tenant breaking out of DomU and gaining access to physical server have been studied in recent work [21] and are out of scope

3. 2 Attack Graph Model

An attack graph is a modeling tool to illustrate all possible multi-stage, multi-host attack paths that are crucial to understand threats and then to decide appropriate countermeasures [22]. In an attack graph, each node represents either precondition or consequence of an exploit. The actions are not necessarily an active attack since normal protocol interactions can also be used for attacks. Attack graph is helpful in identifying potential threats, possible attacks and known vulnerabilities in a cloud system.We extend the notation of MulVAL logic attack graph as presented by X. Ou et al. [12] and define as Scenario Attack Graph (SAG).

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

1. V = NC ∪ND ∪NR denotes a set of vertices that include three types namely conjunction node NC to represent exploit, disjunction node ND to denote result of exploit, and root node NR for showing initial step of an attack scenario.

2. E = Epre∪Epost denotes the set of directed edges.An edge e ∈Epre⊆ ND × NC represents that ND must be satisfied to achieve NC . An edge e ∈Epost⊆ NC × ND means that the consequence shown by ND can be obtained if NC is satisfied.

Node vc∈ NC is defined as a three tuple (Hosts, vul, alert) representing a set of IP addresses, vulnerability information such as CVE [23], and alerts related to vc, respectively. ND behaves like a logical OR operation and contains details of the results of actions. NR represents the root node of the scenario attack graph.

For correlating the alerts,

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

1. A is a set of aggregated alerts. An alert a ∈ A is a data structure (src, dst, cls, ts) representing source IPaddress, destination IP address, type of the alert, and timestamp of the alert respectively.
2. Each alert a maps to a pair of vertices (vc, vd) in SAG using map(a) function, i.e.,

map(a) : a→{(vc, vd)|(a.src∈vc.Hosts) ∧ (a.dst∈vd.Hosts) ∧ (a.cls = vc.vul)}.
3. E is a set of directed edges representing correlation between two alerts (a, a′) if criteria below satisfied:

i. (a.ts< a′.ts) ∧ (a′.ts − a.ts< threshold) ii. ∃(vd, vc) ∈Epre : (a.dst∈vd.Hosts∧ a′.src∈vc.Hosts)
4. P is set of paths in ACG. A path Si ⊂ P is a set of related alerts in chronological order.

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 Si+1 from Si with subset of Si before the alert a and appends ac to Si+1 (line 10). In the end of this algorithm, the ID of ac will be added to alert attribute of the vertex in SAG. Algorithm 1 returns a set of attack paths S in ACG.

Algorithm 1 Alert_Correlation

Require: alert a_c, SAG, ACG

```
1:  if (a_c is a new alert) then
2:      create node a_c in ACG
3:      n_1 ← v_c ∈ map(a_c)
4:      for all n_2 ∈ parent(n_1) do
5:          create edge (n_2.alert, a_c)
6:          for all S_i containing a do
7:              if a is the last element in S_i then
8:                  append a_c to S_i
9:              else
10:                 create path S_{i+1} = {subset(S_i, a), a_c}
11:             end if
12:         end for
13:         add a_c to n_1.alert
14:     end for
15: end if
16: return S
```

3.3 VM Protection Model

The VM protection model of NICE consists of a VM profiler, a security indexer and a state monitor. We specify security index for all the VMs in the network depend-ing upon various factors like connectivity, the number of vulnerabilities present and their impact scores.

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

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

4. NICE SYSTEM DESIGN

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

4.1 System design overview

The proposed NICE framework is illustrated in figure 1. It shows the NICE framework within one cloud server cluster. Major components in this framework are dis-tributed 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 soft-ware switches on each cloud serve. NICE-A is a software agent implemented in each cloud server connected to the control center through a dedicated and isolated secure channel, which is separated from the normal data packets using OpenFlow tunneling or VLAN approaches. The network controller is responsible for deploying attack countermeasures based on decisions made by the attack analyzer.

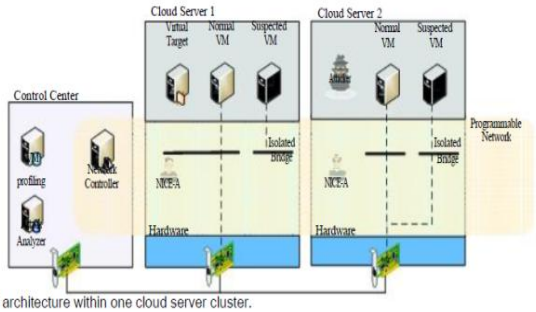

Fig. 1. NICE architecture within one cloud server cluster.

4.2 System Components

In this section we explain each component of NICE.

4.2.1 NICE-A

The NICE-A is a Network-based Invasion Detection System (NIDS) agent installed in either Dom0 or DomU in each cloud server. It scans the traffic going through Linux bridges that control all the traffic among VMs and in/out from the physical cloud servers. In our experiment, Snort is used to implement NICE -A in Dom0. It will sniff a mirroring port on each virtual bridge in the Open vSwitch. Each bridge forms an isolated subnet in the virtual network and connects to all related VMs. The traffic generated from the VMs on the mirrored software bridge will be mirrored to a specific port on a specific bridge using SPAN, RSPAN, or ERSPAN methods.We must note that the alert detection quality of NICE-A depends on the implementation of NICE-A which uses Snort. We do not focus on the detection accuracy of Snort in this article. Thus, the individual alert detection's false alarm rate does not change. However, the false alarm rate could be reduced through our architecture design.

4.2.2 VM Profiling

Virtual machines in the cloud can be profiled to get precise information about their state, services running, open ports, etc. One major factor that counts towards a VM profile is its connectivity with other VMs. Any VM that is connected to more number of machines is more crucial than the one connected to fewer VMs because the effect of compromise of a highly connected VM can cause more damage.

• Attack graph generator: while generating the attack graph, every detected vulnerability is added to itscorresponding VM entry in the database.

• NICE-A: the alert involving the VM will be recordedin the VM profile database.

• Network controller: the traffic patterns involving the VM are based on 5 tuples.

4.2.3 Attack Analyzer

The major functions of NICE system are performed by attack analyzer, which includes procedures such as attack graph construction and update, alert correlation and countermeasure selection.The 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.

4.2.4 Network Controller

The network controller is a key component to support the programmable networking capability to realize the virtual network reconfiguration feature based on OpenFlow protocol [20]. In NICE, within each cloud server there is a software switch, for example, Open vSwitch (OVS) [5], which is used as the edge switch for VMs to handle traffic in & out from VMs. The communication between cloud

servers (i.e., physical servers) is handled by physical OpenFlow-capable Switch (OFS).

Another important function of the network controller is to assist the attack analyzer module. According to the OpenFlow protocol [20], when the controller receives the first packet of a flow, it holds the packet and checks the flow table for comply-ing traffic policies.Network controller is also responsible for applying the counter-measure from attack analyzer. Countermeasure

in such case is to put the suspicious VM with exploited state into quarantine mode and redirect all its flows to NICE-A Deep Packet Inspection (DPI) mode.

5 NICE SECURITY MEASUREMENT, ATTACK MITIGATION AND COUNTERMEASURES

In this section, we present the methods for selecting the countermeasures for a given attack scenario. When vulnerabilities are discovered or some VMs are identified as suspicious, several countermeasures can be taken to restrict attackers' capabilities and it's important to differentiate between compromised and suspicious VMs. The countermeasure serves the purpose of 1) protecting the target VMs from being compromised; and 2) making attack behavior stand prominent so that the attackers' actions can be identified.

5.1 Security Measurement Metrics

The issue of security metrics has attracted much attention and there has been significant effort in the devel- opment of quantitative security metrics in recent years.

The value of BS ranges from 0 to 10. In our attack graph, we assign each internal node with its BS value divided by 10, as shown in (2).

$$GM[e] = Pr(e = T) = BS(e)/10, \forall e \in NC \quad (2)$$

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

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

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

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

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

Once conditional probabilities have been assigned to all internal nodes in SAG, an effective security hardening plan or a mitigation strategy:

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

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

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

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

5.2 Mitigation Strategies

NICE is able to construct the mitigation strategies in response to detected alerts. First, we define the term countermeasure pool as follows:

Definition 4 (Countermeasure Pool). A countermeasure pool $CM = \{cm1, cm2, \ldots, cmn\}$ is a set of countermea-sures. Each $cm \in CM$ is a tuple $cm = (cost, intrusiveness, condition, effectiveness)$, where

1. cost is the unit that describes the expenses required to apply the countermeasure in terms of resources and

operational complexity, and it is defined in a range from1 to 5, and higher metric means higher cost;

2. intrusiveness is the negative effect that a countermea-sure brings to the Service Level Agreement (SLA) and its value ranges from the least intrusive (1) to the most intrusive (5), and the value of intrusiveness is 0 if the countermeasure has no impacts on the SLA;

3. condition is the requirement for the correspondingcountermeasure;

4. effectiveness is the percentage of probability changes of the node, for which this countermeasure is applied.

In general, there are many countermeasures that can be applied to the cloud virtual networking system de- pending on available countermeasure techniques that can be applied. Without losing the generality, several common virtual-networking-based countermeasures are listed in

table 1

TABLE 1
Possible Countermeasure Types

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

5.3 Countermeasure selection

Algorithm 2 presents how to select the optimal countermeasure for a given attack scenario.

Algorithm 2 Countermeasure_Selection

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

Let v_{Alert} = Source node of the $Alert$
if Distance_to_Target(v_{Alert}) $> threshold$ **then**
  Update_ACG
  return
end if
Let $T = Descendant(v_{Alert}) \cup v_{Alert}$
Set $Pr(v_{Alert}) = 1$
Calculate_Risk_Prob(T)
Let $benefit[|T|, |CM|] = \emptyset$
for each $t \in T$ **do**
  for each $cm \in CM$ **do**
    if $cm.condition(t)$ **then**
      $Pr(t) = Pr(t) * (1 - cm.effectiveness)$
      Calculate_Risk_Prob($Descendant(t)$)

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

    end if
  end for
end for
Let $ROI[|T|, |CM|] = \emptyset$
for each $t \in T$ **do**
  for each $cm \in CM$ **do**

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

  end for
end for
Update_SAG and Update_ACG
return Select_Optimal_CM(ROI)

6 PERFORMANCE EVALUATION

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

6.1 Security Performance Analysis

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

6.1.1 Environment and Configuration

To evaluate the security performance, a demonstrative virtual cloud system consisting of public (public virtual servers) and private (VMs) virtual domains is established.

6.1.2 Attack Graph and Alert Correlation

The attack graph can be generated by utilizing network topology and the vulnerability information Creating an attack graph requires knowledge of network connectivity, running services and their vulnerability information.

6.1.3 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.

Apart from calculating the benefit measurements, we also present the evaluation based on Return of Investment(ROI)

6.1.4 Experiment in Private Cloud Environment

Definition 5 (VM Security Index). VSI for a virtual machine k is defined as V SIk = (Vk + Ek)/2, where

1. Vk is vulnerability score for VM k. The score is the exponential average of base score from each vulnerability in Σe VM or a maximum 10, i.e., Vk = min{10, lneBaseScore(v)}.

2. Ek is exploitability score for VM k. It is the exponential average of exploitability score for allvulnerabilities or a maximum 10 multiplied by the ratio of netΣrk services on the VM, i.e., Ek = (min{10, lnSk).Sk represents the number of services provided by VM k. NSk represents the number of network services the VM k can connect to.

6.1.5 False Alarms

A cloud system with hundreds of nodes will have huge amount of alerts raised by Snort. Not all of these alerts can be relied upon, and an effective mechanism is needed to verify if such alerts need to be addressed. Since Snort can be programmed to generate alerts with CVE id, one approach that our work provides is to match. if the alert is actually related to some vulnerability being exploited.

6.2 NICE System Performance

We evaluate system performance to provide guidance on how much traffic NICE can handle for one cloud server and use the evaluation metric to scale up to a large cloud system. In a real cloud system, traffic planning is needed to run NICE, which is beyond the scope of this paper

7 CONCLUSION AND FUTURE WORK

In this paper, we presented NICE, which is proposed to detect and mitigate collaborative attacks in the cloud virtual networking environment. NICE utilizes the attack graph model to conduct attack detection and prediction. The proposed solution investigates how to use the programmability of software switches based solutions to improve the detection accuracy and defeat victim exploitation phases of collaborative attacks.The system performance evaluation

demonstrates the feasibility of NICE and shows that the proposed solution can signif- icantly 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 toover 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 ICE solution by investigating the decentralized network control and attack analysis model based on current study.

REFERENCES

[1] CoudSercurity Alliance, "Top threats to cloud computing v1.0,"https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf,March 2010.
[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia,"A view of cloud computing," ACM Commun., vol. 53, no. 4, pp. 50-58, Apr. 2010.
[3] B. Joshi, A. Vijayan, and B. Joshi, "Securing cloud computing environment against DDoS attacks," IEEE Int'l Conf. Computer Communication and Informatics (ICCCI '12), Jan. 2012.
[4] H. Takabi, J. B. Joshi, and G. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security & Privacy, vol. 8, no. 6, pp. 24-31, Dec. 2010.
[5] "Open vSwitch project," http://openvswitch.org, May2012.
[6] Z. Duan, P. Chen, F. Sanchez, Y. Dong, M. Stephenson, andJ. 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 nn. Network and Distributed Sytem Security Symp. (NDSS '08), Feb. 2008.
[9] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing,"Automated generation and analysis of attack graphs," Proc. IEEE Symp. on Security and Privacy, 2002, pp. 273-284.
[10] "NuSMV: A new symbolic model checker," http://afrodite.itc.it:1024/~nusmv. Aug. 2012.
[11] S. H. Ahmadinejad, S. Jalili, and M. Abadi, "A hybrid model for correlating alerts of known and unknown attack scenarios and updating attack graphs," Computer Networks, vol. 55, no. 9, pp. 2221-2240, Jun. 2011.
[12] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: a logic-based network security analyzer," Proc. of 14th USENIX SecuritySymp., 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 invasion alerts," Computer Communications, vol. 29, no. 15, pp. 2917-2933, Sep. 2006.