Inferring the Stealthy Bridges between Enterprise Network Islands in Cloud Using Cross-Layer Bayesian Networks

Xiaoyan Sun[1], Jun Dai[1], Anoop Singhal[2], and Peng Liu[1]

[1] Penn State University, University Park, PA 16802, USA,
[2] National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
{xzs5052,jqd5187}@ist.psu.edu, anoop.singhal@nist.gov, pliu@ist.psu.edu

Abstract. Enterprise networks are migrating to the public cloud to acquire computing resources for promising benefits in terms of efficiency, expense, and flexibility. Except for some public services, the enterprise network islands in cloud are expected to be absolutely isolated from each other. However, some "stealthy bridges" may be created to break such isolation due to two features of the public cloud: virtual machine image sharing and virtual machine co-residency. This paper proposes to use cross-layer Bayesian networks to infer the stealthy bridges existing between enterprise network islands. Prior to constructing cross-layer Bayesian networks, cloud-level attack graphs are built to capture the potential attacks enabled by stealthy bridges and reveal hidden possible attack paths. The result of the experiment justifies the cross-layer Bayesian network's capability of inferring the existence of stealthy bridges given supporting evidence from other intrusion steps in a multi-step attack.
Key words: cloud, stealthy bridge, Bayesian network, attack graph

1 Introduction

Enterprises have begun to move parts of their networks (such as web server, mail server, etc.) from traditional infrastructure into cloud computing environments. Cloud providers such as Amazon Elastic Compute Cloud (EC2) [1], Rackspace [2], and Microsoft's Azure cloud platform [3] provide virtual servers that can be rented on demand by users. This paradigm enables cloud customers to acquire computing resources with high efficiency, low cost, and great flexibility. However, it also introduces some security issues that are yet to be solved.

A public cloud can provide virtual infrastructures to many enterprises. Except for some public services, enterprise networks are expected to be like isolated islands in the cloud: connections from the outside network to the protected internal network should be prohibited. Consequently, an attack path that shows the multi-step exploitation sequence in an enterprise network should also be confined inside this island. However, as enterprise networks migrate into the cloud and replace traditional physical hosts with virtual machines, some "stealthy bridges" could be created between the isolated enterprise network islands, as shown in Fig. 1. Moreover, with the stealthy bridges, the attack path confined inside an enterprise network is able to traverse to another enterprise network in cloud.

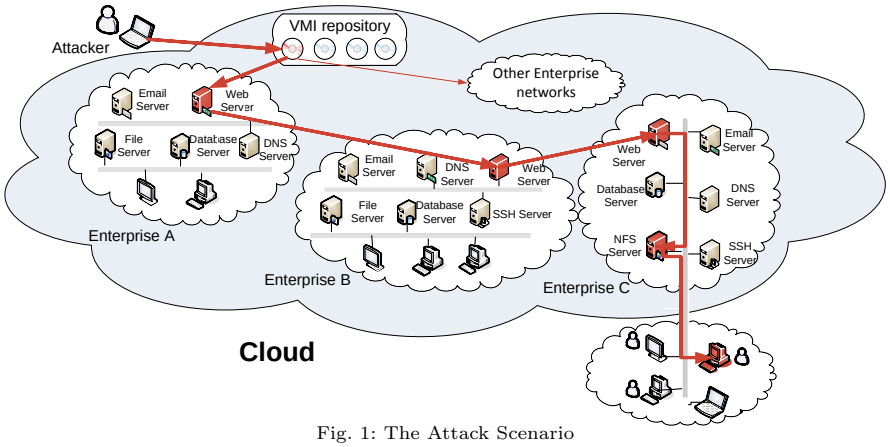


Fig. 1: The Attack Scenario

The creation of such "stealthy bridges" is enabled by two unique features of the public cloud. First, cloud users are allowed to create and share virtual machine images (VMIs) with other users. Besides, cloud providers also provide VMIs with pre-configured software, saving users' efforts of installing the software from scratch. These VMIs provided by both cloud providers and users form a large repository. For convenience, users can take a VMI directly from the repository and instantiate it with ease. The instance virtual machine inherits all the security characteristics from the parent image, such as the security configurations and vulnerabilities. Therefore, if a user instantiates a *malicious* VMI, it's like moving the attacker's machine directly into the internal enterprise network, without triggering the Intrusion Detection Systems (IDSs) or the firewall. In this case, a "stealthy bridge" can be created via security holes such as backdoors. For example, in Amazon EC2, if an attacker intentionally leaves his public key unremoved when publishing an AMI (Amazon Machine Image), the attacker can later login into the running instances of this AMI with his own private key.

Second, virtual machines owned by different tenants may co-reside on the same physical host machine. To achieve high efficiency, customer workloads are multiplexed onto a single physical machine utilizing virtualization. Virtual machines on the same host may belong to unrelated users, or even rivals. Thus co-resident virtual machines are expected to be absolutely isolated from each other. However, current virutalization mechanisms cannot ensure perfect isolation. The co-residency relationship can still enable security problems such as information leakage, performance interference [4], or even co-resident virtual machine crashing. Previous work [5] has shown that it is possible to identify on which physical host a target virtual machine is likely to reside, and then intentionally place an attacker virtual machine onto the same host in Amazon EC2. Once the co-residency is achieved, a "stealthy bridge" can be further established, such as a side-channel for passively observing the activities of the target machine to extract information for credential recovering [6], or a covert-channel for actively sending information from the target machine [8].

Stealthy bridges are stealthy information tunnels existing between disparate networks in cloud, that are unknown to security sensors and should have been forbidden. Stealthy bridges are developed mainly by exploiting *vulnerabilities that are unknown* to vulnerability scanners. Isolated enterprise network islands are connected via these stealthy tunnels, through which information (data, commands, etc.) can be acquired, transmitted or exchanged maliciously. Therefore stealthy bridges pose very severe threats to the security of public cloud. However, the stealthy bridges are inherently unknown or hard to detect: they either exploit unknown vulnerabilities, or cannot be easily distinguished from authorized activities by security sensors. For example, side-channel attacks extract information by passively observing the activities of resources shared by the attacker and the target virtual machine (e.g. CPU, cache), without interfering the normal running of the target virtual machine. Similarly, the activity of logging into an instance by leveraging intentionally left credentials (passwords, public keys, etc.) also hides in the authorized user activties.

The stealthy bridges can be used to construct a multi-step attack and facilitate subsequent intrusion steps across enterprise network islands in cloud. The stealthy bridges per se are difficult to detect, but the intrusion steps before and after the construction of stealthy bridges may trigger some abnormal activities. Human administrators or security sensors like IDS could notice such abnormal activities and raise corresponding alerts, which can be collected as the evidence of attack happening[1]. So our approach has two insights: 1) It is quite straightforward to build a cloud-level attack graph to capture the potential attacks enabled by stealthy bridges. 2) To leverage the evidence collected from other intrusion steps, we construct a cross-layer Bayesian Network (BN) to infer the existence of stealthy bridges. Based on the inference, security analysts will know where stealthy bridges are most likely to exist and need to be further scrutinized.

The main contributions of this paper are as follows:

First, a cloud-level attack graph is built by crafting new interaction rules in *MulVAL* [18], an attack graph generation tool. The cloud-level attack graph can capture the potential attacks enabled by stealthy bridges and reveal possible hidden attack paths that are previously missed by individual enterprise network attack graphs.

Second, based on the cloud-level attack graph, a cross-layer Bayesian network is constructed by identifying four types of uncertainties. The cross-layer Bayesian network is able to infer the existence of stealthy bridges given supporting evidence from other intrusion steps.

2 Cloud-level Attack Graph Model

A Bayesian network is a probabilistic graphical model that is applicable for real-time security analysis. Prior to the construction of a Bayesian Network, an attack graph should be built to reflect the attacks enabled by stealthy bridges.

[1] In our trust model, we assume cloud providers are fully trusted by cloud customers. In addition to security alerts generated at cloud level, such as alerts from hypervisors or cache monitors, the cloud providers also have the privilege of accessing alerts generated by customers' virtual machines.

2.1 Logical Attack Graph

An attack graph is a valuable tool for network vulnerability analysis. Current network defenders should not only understand how attackers could exploit a specific vulnerability to compromise one single host, but also clearly know how the security holes can be combined together for achieving an attack goal. An attack graph is powerful for dealing with the combination of security holes. Taking vulnerabilities existing in a network as the input, attack graph can generate the possible attack paths for a network. An attack path shows a sequence of potential exploitations to specific attack goals. For instance, an attacker may first exploit a vulnerability on Web Server to obtain the root privilege, and then further compromise Database Server through the acquired privilege. A variety of attack graphs have been developed for vulnerability analysis, mainly including state enumeration attack graphs [12, 13, 14] and dependency attack graphs [15, 16, 17]. The tool *MulVAL* employed in this paper is able to generate the logical attack graph, which is a type of dependency attack graph.

Fig. 2 shows part of an exemplar logical attack graph. There are two types of nodes in logical attack graph: derivation nodes (also called rule nodes, represented with ellipse), and fact nodes. The fact nodes could be further classified into primitive fact nodes (in rectangles), and derived fact nodes (in diamonds). Primitive fact nodes are typically objective conditions of the network, including network connectivity, host configuration, and vulnerability information. Derived fact nodes represent the facts inferred from logical derivation. Derivation nodes represent the interaction rules used for derivation. The directed edges in this graph represent the causality relationship between nodes. In a logical dependency attack graph, one or more fact nodes could serve as the preconditions of a derivation node and cause it to take effect. One or more derivation nodes could further cause a derived fact node to become true. Each derivation node represents the application of an interaction rule given in [19] that yields the derived fact.

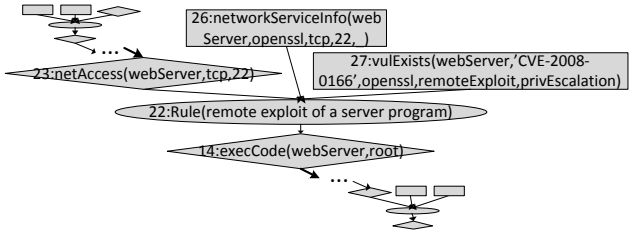


Fig. 2: A Portion of an Example Logical Attack Graph

For example, in Fig. 2, Node 26, 27 (primitive fact nodes) and Node 23 (derived fact node) are three fact nodes. They represent three preconditions respectively: Node 23, the attacker has access to the Web Server; Node 26, Web Server provides *OpenSSL* service; Node 27, Openssl has a vulnerability *CVE-2008-0166*. With the three preconditions satisfied simultaneously, the rule of Node 22 (derivation node) can take effect, meaning the remote exploit of a server program could happen. This derivation rule can further cause Node 14 (derived fact node) to be valid, meaning attacker can execute code on Web Server.

2.2 Cloud-level Attack Graph

In the cloud, each enterprise network can scan its own virtual machines for existing vulnerabilities and then generate an attack graph. The individual attack graph shows how attackers could exploit certain vulnerabilities and conduct a sequence of attack steps inside the enterprise network. However, such individual attack graphs are confined to the enterprise networks without considering the potential threats from cloud environment. The existence of stealthy bridges could activate the prerequisites of some attacks that are previously impossible in traditional network environment and thus enable new attack paths. These attack paths are easily missed by individual attack graphs. For example, in Fig. 1, without assuming the stealthy bridge existing between enterprise A and B, the individual attack graph for enterprise B can be incomplete or even not established due to lack of exploitable vulnerabilities. Therefore, a cloud-level attack graph needs to be built to incorporate the existence of stealthy bridges in the cloud. By considering the attack preconditions enabled by stealthy bridges, the cloud-level attack graph can reveal hidden potential attack paths that are missed by individual attack graphs.

The cloud-level attack graph should be modeled based on the cloud structure. Due to the VMI sharing feature and the co-residency feature of cloud, a public cloud has the following structural characteristics. First, virtual machines can be created by instantiating VMIs. Therefore virtual machines residing on different hosts may actually be instances of the same VMI. In simple words, they could have the same VMI parents. Second, virtual machines belong to one enterprise network may be assigned to a number of different physical hosts that are shared by other enterprise networks. That is, the virtual machines employed by different enterprise networks are likely to reside on the same host. As shown in Fig. 3, the vm_{11} on host 1 and vm_{2j} on host 2 may be instances of the same VMI, while vm_{12} and vm_{2k} could belong to the same enterprise network. Third, the real enterprise network could be a hybrid of a cloud network and a traditional network. For example, the servers of an enterprise network could be implemented in the cloud, while the personal computers and workstations could be in the traditional network infrastructure.

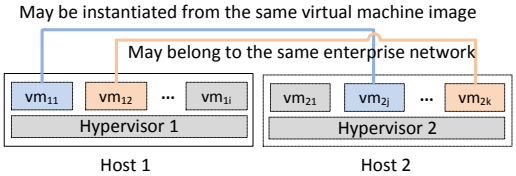


Fig. 3: Features of the Public Cloud Structure

Due to the above characteristics of cloud structure, the model for the cloud-level attack graph should have the following corresponding characteristics.

1) The cloud-level attack graph is a cross-layer graph that is composed of three layers: virtual machine layer, VMI layer, and host layer, as shown in Fig. 4.

2) The virtual machine layer is the major layer in the attack graph stack. This layer reflects the causality relationship between vulnerabilities existing inside the virtual machines and the potential exploits towards these vulnerabilities. If

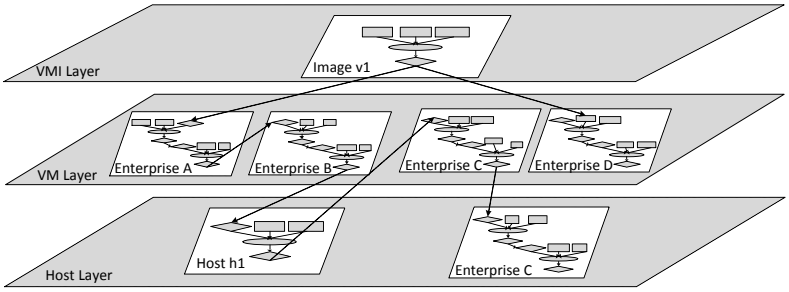


Fig. 4: An Example Cloud-level Attack Graph Model

stealthy bridges do not exist, the attack graph generated in this layer is scattered: each enterprise network has an individual attack graph that is isolated from others. The individual attack graphs can be the same as the ones generated by cloud customers themselves through scanning the virtual machines for known vulnerabilities. However, if stealthy bridges exist on the other two layers, the isolated attack graph could be connected, or even experience dramatic changes: some hidden potential attack paths will be revealed and the original attack graph is enriched. For example, in Fig. 4, without the stealthy bridge on *h1*, attack paths in enterprise network C will be missing or incomplete because no exploitable vulnerability is available as the entry point for attack.

3) The VMI layer mainly captures the stealthy bridges and corresponding attacks caused by VMI sharing. Since virtual machines in different enterprise networks may be instantiated from the same parent VMI, they could inherit the same security issues from parent image, such as software vulnerabilities, malware, or backdoors, etc. Evidence from [20] shows that 98% of Windows VMI and 58% of Linux VMIs in Amazon EC2 contain software with critical vulnerabilities. A large number of software on these VMIs are more than two years old. Since cloud customers take full responsibility for securing their virtual machines, many of these vulnerabilities remain unpatched and thus pose great risks to cloud. Once a vulnerability or an attack type is identified in the parent VMI, the attack graph for all the children virtual machine instances may be affected: a precondition node could be activated, or a new interaction rule should be constructed in attack graph generation tool.

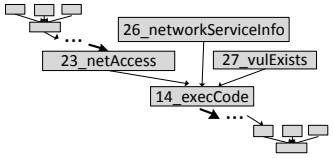
The incorporation of the VMI layer provides another benefit to the subsequent Bayesian network analysis. It enables the interaction between the virtual machine layer and the VMI layer. On one hand, the probability of a vulnerability existence on a VMI will affect the probability of the vulnerability existence on its children instance virtual machines. On the other hand, if new evidence is found regarding the vulnerability existence on the children instances, the probability change will in turn influence the parent VMI. If the same evidence is observed on multiple instances of the VMI, this VMI is very likely to be problematic.

4) The host layer is able to reason exploits of stealthy bridges caused by virtual machine co-residency. Exploits on this layer could lead to further penetrations on the virtual machine layer. In addition, this layer actually captures all attacks that could happen on the host level, including those on pure physical

hosts with no virtual machines. Hence it provides a good interface to hybrid enterprise networks that are implemented with partial cloud and partial traditional infrastructures. The potential attack paths identified on the cloud part could possibly extend to traditional infrastructures if all prerequisites for the remote exploits are satisfied, such as network access being allowed, and exploitable vulnerabilities existing, etc. As in Fig. 4, the attack graph for enterprise C extends from virtual machine layer to host layer.

3 Cross-layer Bayesian Networks

A Bayesian network (BN) is a probabilistic graphical model representing cause and effect relations. For example, it is able to show the probabilistic causal relationships between a disease and the corresponding symptoms. Formally, a Bayesian network is a Directed Acyclic Graph (DAG) that contains a set of nodes and directed edges. The nodes represent random variables of interest and the directed edges represent the causal influence among the variables. The strength of such influence is represented with a conditional probability table (CPT). For example, Fig. 5 shows a portion of a BN constructed directly from the attack graph in Fig. 2 by removing the rule Node 22. Node 14 can be associated with the CPT table as shown. This CPT means that if all of the preconditions of Node 14 are satisfied, the probability of Node 14 being true is 0.9. Node 14 is false in all other cases.



26	27	23	14
T	T	T	0.9
otherwise			0

Fig. 5: A Portion of Bayesian Network with associated CPT table

A Bayesian network can be used to compute the probabilities of variables of interest. It is especially powerful for diagnosis and prediction analysis. For example, in diagnosis analysis, given the symptoms being observed, a BN can calculate the probability of the causing fact (respresented with $\Pr(cause \mid symptom = True)$). While in prediction analysis, given the causing fact, a BN will predict the probability of the corresponding symptoms showing up ($Pr(symptom|cause = True)$). In the cybersecurity field, similar diagnosis and prediction analysis can also be performed, such as calculating the probability of an exploitation happening if related IDS alerts are observed($Pr(exploitation|IDSalert = True)$), or the probability of the IDS raising an alert if an exploitation already happened ($Pr(IDSalert|exploitation = True)$). This paper mainly carries out a diagnosis analysis that computes the probability of stealthy bridge existence by collecting evidence from other intrusion steps. Diagnosis analysis is a kind of "backward" computation. In the cause-and-symptom model, a concrete evidence about the symptom could change the posterior probability of the cause by computing $Pr(cause|symptom = True)$. More intuitively, as more evidence is collected regarding the symptom, the probability of the cause will become closer to reality if the BN is constructed properly.

3.1 Identify the Uncertainties

Inferring the existence of stealthy bridges requires real-time evidence being collected and analyzed. BN has the capability, which attack graphs lack, of performing such real-time security analysis. Attack graphs correlate vulnerabilities and potential exploits in different machines and enables *determinstic* reasoning. For example, if all the preconditions of an attack are satisfied, the attacker *should* be able to launch the attack. However, in *real-time* security analysis, there are a range of uncertainties associated with this attack that cannot be reflected in an attack graph. For example, has the attacker chosen to launch the attack? If he launched it, did he succeed to compromise the host? Are the Snort [22] alerts raised on this host related to the attack? Should we be more confident if we got other alerts from other hosts in this network? Such uncertainty aspects should be taken into account when performing real-time security analysis. BN is a valuable tool for capturing these uncertainties.

One non-trivial difficulty for constructing a well functioning BN is to identify and model the uncertainty types existing in the attack procedure. In this paper, we mainly consider four types of uncertainties related to cloud security.

Uncertainty of stealthy bridges existence. The presence of known vulnerabilities is usually deterministic due to the availability of vulnerability scanners. After scanning a virtual machine or a physical host, the vulnerability scanner such as Nessus [24] is able to tell whether a known vulnerability exists or not[2]. However, due to its unknown or hard-to-detect feature, effective scanners for stealthy bridges are rare. Therefore, the existence of stealthy bridges itself is a type of uncertainty. In this paper, to enable the construction of a complete attack graph, stealthy bridges are hypothesized to be existing when corresponding conditions are met. For example, if two virtual machines co-reside on the same physical host and one of them has been compromised by the attacker, the attack graph will be generated by making a hypothesis that a stealthy bridge can be created between these two virtual machines. This is enforced by crafting a new interaction rule as follows in *MulVAL*:

```
interaction rule(
   (stealthyBridgeExists(Vm_1,Vm_2, Host, stealthyBridge_id):-
      execCode(Vm_1,_user),
      ResideOn(Vm_1, Host),
      ResideOn(Vm_2, Host)),
    rule_desc('A stealthy bridge could be built between virtual machines co-residing on
 the same host after one virtual machine is compromised')).
```

Afterwards, the BN constructed based on the attack graph will infer the probability of this hypothesis being true.

Uncertainty of attacker action. Uncertainty of attacker action is first identified by [23]. Even if all the prerequsites for an attack are satisfied, the attack may not happen because attackers may not take action. Therefore, a kind of Attack Action Node (AAN) is added to the BN to model attackers' actions. An AAN node is introduced as an additional parent node for the attack. For example, the BN shown in Fig. 5 is changed to Fig. 6 after adding an AAN node.

[2] The assumption here is that a capable vulnerability scanner is able to scan out all the known vulnerabilities.

Correspondingly, the CPT table is modified as in Fig. 6. This means "attacker taking action" is another prerequisite to be satisfied for the attack to happen.

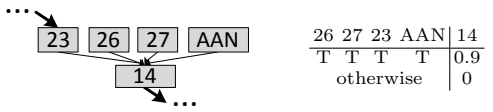


Fig. 6: A Portion of Bayesian Network with AAN node

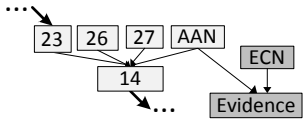
An AAN node is not added for all attacks. They are needed only for important attacks such as the very first intrustion steps in a multi-step attack, or attacks that need attackers' action. Since an AAN node represents the primitive fact of whether an attacker taking action and has no parent nodes, a prior probability distribution should be assigned to an AAN to indicate the likelihood of an attack. The posterior probability of AAN will change as more evidence is collected.

Uncertainty of exploitation success. Uncertainty of exploitation success goes to the question of "did the attacker succeed in this step?". Even if all the prerequisites are satisfied and the attacker indeed launches the attack, the attack is not guarenteed to succeed. The success likelihood of an attack mainly depends on the exploit difficulty of vulnerabilities. For some vulnerabilities, usable exploit code is already publicly available. While for some other vulnerabilities, the exploit is still in the proof-of-concept stage and no successful exploit has been demonstrated. Therefore, the exploit difficulty of a vulnerability can be used to derive the CPT table of an exploitation. For example, if the exploit difficulty for the vulnerability in Fig. 5 is very high, the probability for Node 14 when all parent nodes are true could be assigned as very low, such as 0.3. If in the future a public exploit code is made available for this vulnerability, the probability for Node 14 may be changed to a higher value accordingly. The National Vulnerability Database (NVD) [25] maintains a CVSS [26] scoring system for all CVE [27] vulnerabilities. In CVSS, Access Complexity (AC) is a metric that describes the exploit complexity of a vulnerability using values of "high", "medium", "low". Hence the AC metric can be employed to derive CPT tables of exploitations and model the uncertainty of exploitation success.

Uncertainty of evidence. Evidence is the key factor for BN to function. In BN, uncertainties are indicated with probability of related nodes. Each node describes a real or hypothetical event, such as "attacker can execute code on Web Server", or "a stealthy bridge exists between virtual machine A and B", etc. *Evidence is collected to reduce uncertainty* and calculate the probabilities of these events. According to the uncertainty types mentioned above, evidence is also classified into three types: evidence for stealthy bridges existence, evidence for attacker action, and evidence for exploitation success. Therefore, whenever a piece of evidence is observed, it is assigned to one of the above evidence types to support the corresponding event. This is done by adding evidence as the children nodes to the event nodes related to uncertainty. For example, an IDS alert about a large number of login attempts can be regarded as evidence of attacker action, showing that an attacker could have tried to launch an attack. This evidence is then added as the child node to an AAN, as exemplified in Fig. 7. For another example, the alert "system log is deleted" given by Tripwire [28] can be the

child of the node "attacker can execute code", showing that an exploit has been successfully achieved.

However, evidence per se contain uncertainty. The uncertainty is twofold. First, the support of evidence to an event is uncertain. For analogy, a symptom of coughing cannot completely prove the presence of lung disease. In the above examples, could the multiple login attempts testify that attackers have launched the attack? How likely is it that attackers have succeeded in compromising the host if a system log deletion is observed? Second, evidence from security sensors is not 100% accurate. IDS systems such as Snort, Tripwire, etc. suffer a lot from a high false alert rate. For example, an event may trigger an IDS to raise an alert while actually no attack happens. In this case, the alert is a false positive. The reverse case is a false negative, that is, when an IDS should have raised an alarm but doesn't. Therefore, we propose to model the uncertainty of evidence with an Evidence-Confidence(EC) pair as shown in Fig. 7. The EC pair has two nodes, an Evidence node and an Evidence Confidence Node (ECN). An ECN is assigned as the parent of an Evidence node to model the confidence level of the evidence. If the confidence level is high, the child evidence node will have larger impact on other nodes. Otherwise, the evidence will have lower impact on others. An example CPT associated with the evidence node is given in Fig. 7. Whenever new evidence is observed, an EC pair is attached to the supported node. A node can have several EC pairs attached with it if multiple instances of evidence are observed. With ECN nodes, security experts can tune confidence levels of evidence with ease based on their domain knowledge and experience. This will greatly enhance the flexibility and accuracy of BN analysis.



AAN	True					False				
ECN	VeryHigh	High	Medium	Low	None	VeryHigh	High	Medium	Low	None
True	0.95	0.8	0.6	0.55	0.5	0.05	0.2	0.4	0.45	0.5
False	0.05	0.2	0.4	0.45	0.5	0.95	0.8	0.6	0.55	0.5

Fig. 7: The Evidence-Condidence Pair and Associated Exemplar CPT

4 Implementation

4.1 Cloud-level Attack Graph Generation

This paper uses *MulVAL* [19] as the attack graph generation tool. To construct a cloud-level attack graph, new primitive fact nodes and interaction rules have to be crafted in *MulVAL* on the VMI layer and host layer to model the existence of stealthy bridges. Each virtual machine has an ID tuple *(Vm_id, VMI_id, H_id)* associated with it, which represents the ID for the virtual machine itself, the VMI it was derived from, and the host it resides on. The VMI layer mainly focuses on the model of VMI vulnerability inheritance and the VMI backdoor problems. The host layer mainly focuses on modeling the virtual machine co-residency problems. Table 1 provides a sample set of newly crafted interaction rules that are incorporated into *MulVAL* for cloud-level attack graph generation.

4.2 Construction of Bayesian Networks

Deriving Bayesian networks from cross-layer attack graphs consists of four major components: removing rule nodes in the attack graph, adding new nodes, determining prior probabilities, and constructing CPT tables.

Table 1: a Sample Set of Interaction Rules

```
/***Model the Virtual Machine Image Vulnerability Inheritance***/
primitive(IsInstance(Vm_id, VMI_id))
primitive(ImageVulExists(VMI_id, vulID, _program, _range, _consequence))
derived(VulExists(Vm_id, vulID, _program,_range,_consequence)).

%remove vulExists from the primitive fact set
primitive(vulExists(_host, _vulID, _program, _range, _consequence)

interaction rule(
   (VulExists(Vm_id, vulID, _program, _range, _consequence):-
      ImageVulExists(VMI_id, vulID, _program, _range, _consequence),
      IsInstance(Vm_id, VMI_id)),
    rule_desc('A virtual machine instance inherits the vulnerability from the parent VMI')).

/***Model the Virtual Machine Image Backdoor Problem***/
primitive(IsThirdPartyImage(VMI_id)).
derived(ImageVulExists(VMI_id, sealthyBridge_id, _, _remoteExploit, privEscalation)).

interaction rule(
   (ImageVulExists(VMI_id,stealthyBridge_id, _, _remoteExploit, privEscalation):-
      IsThirdPartyImage(VMI_id)),
    rule_desc('A third party VMI could contain a stealthy bridge')).

interaction rule(
   (execCode(Vm_id, Perm):
      VulEixsts(Vm_id, stealthyBridge_id, _, _, privEscalation),
      netAccess(H, _Protocol, _Port)),
    rule_desc('remoteExploit of a stealthy bridge')).

/***Model the Virtual Machine Co-residency Problem***/
primitive(ResideOn(VM_id, H_id)).
derived(stealthyBridgeExists(Vm_1,Vm_2, H_id, stealthyBridge_id).

interaction rule(
   (stealthyBridgeExists(Vm_1,Vm_2, Host, stealthyBridge_id):-
      execCode(Vm_1,_user),
      ResideOn(Vm_1, Host),
      ResideOn(Vm_2, Host)),
   rule_desc('A stealthy bridge could be built between virtual machines co-residing on
 the same host after one virtual machine is compromised')).

interaction rule(
   (execCode(Vm_2,_user):-
      stealthyBridgeExists(Vm_1,Vm_2, Host, stealthyBridge_id)),
    rule_desc('A stealthy bridge could lead to privilege escalation on victim machine')).

interaction rule(
   (canAccessHost(Vm_2):-
      logInService(Vm_2,Protocol,Port),
      stealthyBridgeExists(Vm_1,Vm_2,Host,stealthyBridge_id)),
    rule_desc('Access a host through a log-in service by obtaining authentication
information through stealthy bridges')).
```

Remove rule nodes of attack graph. In an attack graph, the rule nodes imply how postconditions are derived from preconditions. The derivation is deterministic and contains no uncertainty. Therefore, these rule nodes have no effect on the reasoning process, and thus can be removed when constructing the BN. To remove a rule node, its preconditions are connected directly to its postconditions. For example, in Fig. 2, Node 26, 27, and 23 will be connected directly to Node 14 by removing Node 22.

Adding new nodes. New nodes are added to capture the uncertainty of attacker action and the uncertainty of evidence. To capture the uncertainty of

attacker action, each step has a separate AAN node as the parent, rather than sharing the same AAN among multiple steps. The AAN node models attacker action at the granularity of attack steps, and thus reflects the actual attack paths. To model the uncertainty of evidence, whenever new evidence is observed, an EC pair is constructed and attached to the supported node with uncertainty.

Determining prior probabilities. Prior probability distributions should be determined for all root nodes that have no parents, such as the vulnerability existence nodes, the network access nodes, or the AAN nodes.

Constructing CPT tables. Some CPT tables can be determined according to a standard, such as the the AC metric in CVSS scoring system. The AC metric describes the exploit complexity of vulnerabilities and thus can be used to derive the CPT tables for corresponding exploitations. Some other CPT tables may involve security experts' domain knowledge and experience. For example, the VMIs from a trusted third party may have lower probability of containing security holes such as backdoors, while those created and shared by individual cloud users may have higher probability.

The constructed BN should be robust against small changes in prior probabilities and CPT tables. To ensure such robustness, we use *SamIam* [33] for sensitivity analysis when constructing and debugging the BN. By specifying the requirements for an interested node's probability, *SamIam* will check the associated CPT tables and provide suggestions on feasible changes. For example, if we want to change $P(N5 = True)$ from 0.34 to 0.2, *SamIam* will provide two suggestions, either changing $P(N5 = True|N2 = True, N3 = True)$ from 0.9 to $<= 0.43$, or changing $P(N3 = True|N1 = True)$ from 0.3 to $<= 0.125$.

5 Experiment

5.1 Attack Scenario

Fig. 1 shows the network structure in our attack scenario. We have 3 major enterprise networks: A, B, and C. A and B are all implemented within the cloud, while C is implemented by partially cloud, and partially traditional network (the servers are located in the cloud and the workstations are in a traditional network). The attack includes several steps conducted by attacker Mallory.

Step 1, Mallory first publishes a VMI that provides a web service in the cloud. This VMI is malicious in that it contains a security hole that Mallory knows how to exploit. For example, this security hole could be an SSH user authentication key (the public key located in *.ssh/authorized_keys*) that is intentionally left in the VMI by Mallory. The leftover creates a backdoor that allows Mallory to login into any instances derived from this malicious VMI using his own private key. The security hole could also be an unknown vulnerability that is not yet publicly known. To make the attack scenario more generic, we choose a vulnerability *CVE-2007-2446* [29], existing in *Samba 3.0.0* [30], as the one imbedded in the malicious VMI, but assume it as *unknown* for the purpose of simulation.

Step 2, the malicious VMI is then adopted and instantiated as a web server by an innocent user from A. Mallory now wants to compromise the live instances, but he needs to know which instances are derived from his malicious VMI. [20] provides three possible ways for machine fingerprinting: ssh matching, service

matching, and web matching. Through ssh key matching, Mallory finds the right instance in A and completes the exploitation towards *CVE-2007-2446* [29].

Step 3, enterprise network B provides web services to a limited number of customers, including A. With the acquired root privilege from A's web server, Mallory is able to access B's web server, exploit one of its vulnerabilities *CVE-2007-5423* [31] from application *tikiwiki 1.9.8* [32], and create a reverse shell.

Step 4, Mallory notices that enterprise B and C has a special relationship: their web servers are implemented with virtual machines co-residing on the same host. C is a start-up company that has some valuable information stored on its CEO's workstation. Mallory then leverages the co-residency relationship of the web servers and launches a side-channel attack towards C's web server to extract its password. Mallory obtains user privilege through the attack. Mallory also establishes a covert channel between the co-resident virtual machines for convenient information exchange.

Step 5, the NFS server in C has a directory that is shared by all the servers and workstations inside the company. Normally C's web server should not have *write* permission to this shared directory. But due to a configuration error of the NFS export table, the web server is given *write* permission. Therefore, if Mallory can upload a Trojan horse to the shared directory, other innocent users may download the Trojan horse from this directory and install it. Hence Mallory crafts a Trojan horse *management_tool.deb* and uploads it into the shared NSF directory on web server.

Step 6, The innocent CEO from C downloads *management_tool.deb* and installs it. Mallory then exploits the Trojan horse and creats a unsolicited connection back to his own machine.

Step 7, Mallory's VMI is also adopted by several other enterprise networks, so Mallory compromises their instances using the same method in Step 2.

In this scenario, two stealthy bridges are established[3]: one is from Internet to enterprise network A through exploiting an unknown vulnerability, the other one is between enterprise network B and C by leveraging virtual machine co-residency. The attack path crosses over three enterprise networks that reside in the same cloud, and extends to C's traditional network.

5.2 Experiment Result

The purpose of our experiment is to check whether the BN-based tool is able to infer the existence of stealthy bridges given the evidence. The Bayesian network has two inputs: the network deployment (network connection, host configuration, and vulnerability information, etc.) and the evidence. The output of BN is the probability of specific events, such as the probability of stealthy bridges being established, or the probability of a web server being compromised. We view the attackers' sequence of attack steps as a set of ground truth. To evaluate the effectiveness of the constructed BN, we compare the output of the BN with the ground truth of the attack sequence. For example, given the ground truth that a

[3] The enterprise networks in Step 7 are not key players, so we do not analyze the stealthy bridges established in this step, but still use the raised alerts as evidence.

stealthy bridge has been established, we will check the corresponding probability provided by the BN to see whether the result is convincible.

For the attack scenario illustrated in Fig. 1, the cross-layer BN is constructed as in Fig. 8. By taking into account the existence of stealthy bridges, the cloud-level attack graph has the capability of revealing potential hidden attack paths. Therefore, the constructed BN also inherits the revealed hidden paths from the cloud-level attack graph. For example, the white part in Fig. 8 shows the hidden paths enabled by the stealthy bridge between enterprise network B and C. These paths will be missed by individual attack graphs if the stealthy bridge is not considered. The inputs for this BN are respectively the network deployment shown in Table 2[4] and the collected evidence is shown in Table 3. Evidence is collected against the attack steps described in our attack scenario. Not all attack steps have corresponding observed evidence.

Table 2: Network Deployment

Node	Deployed Facts
N1	IsThirdPartyImage(VMI)
N2	IsInstance(Aws, VMI)
N4	netAccess(Aws,_protocol,_port)
N17	netServiceInfo(Bws,tikiwiki,http,80,_)
N19	ResideOn(Bws,H)
N20	ResideOn(Cws,H)
N21	hacl(Cws,Cnfs,nfsProtocol,nfsPort)
N27	nfsExport(Cnfs,'/export',write,Cws)
N30	nfsMountd(CworkStation,'/mnt/share', Cnfs,'/export',read)
N32	VulExists(CworkStation,'CVE-2009-2692',kernel,localExploit,privEscalation)
N41	IsInstance(Dws,VMI)
N43	netAccess(Dws,_protocol,_port)

Table 3: Collected Evidence Corresponding to Attack Steps

Node	Step	Collected Evidence
N9	2	Wireshark shows multiple suspicious connections established
N11	2	IDS shows malicious packet detected
N13	2	Wireshark "follow tcp stream" shows a back telnet connection is instructed to open
N23	4	Cache monitor observes abnormal cache activities
N34	5	Tripwire shows several file modification toward management_tool.deb
N37	6	IDS shows Trojan horse installation
N39	6	Wireshark "follow tcp stream" find plain text in supposed encrypted-connection
N47	7	Wireshark shows a back telnet connection is instructed to open
N49	7	IDS shows malicious packet detected

We conducted four sets of simulation experiments, each with a specific purpose. For simplicity, we assume all attack steps are completed instantly with no time delay. The ground truth in our attack scenario tells that one stealthy bridge between attacker and enterprise A is established in attack step 2, and the other one between B and C is established in step 4. By taking evidence with a certain order as input, the BN will generate a corresponding sequence of probabilities for events of interest. The probabilities are compared with the ground truth to evaluate the performance of the BN.

In experiment 1, we assume all the evidence is observed in the order of the corresponding attack steps. We are interested in four events, a stealthy bridge exists in enterprise A's web server (N5), the attacker can execute arbitrary code

[4] Aws,Bws,Cws,Cnfs,Cworkstation denote A's web server, B's web server, C's web server, C's NFS server, C's workstation respectively.

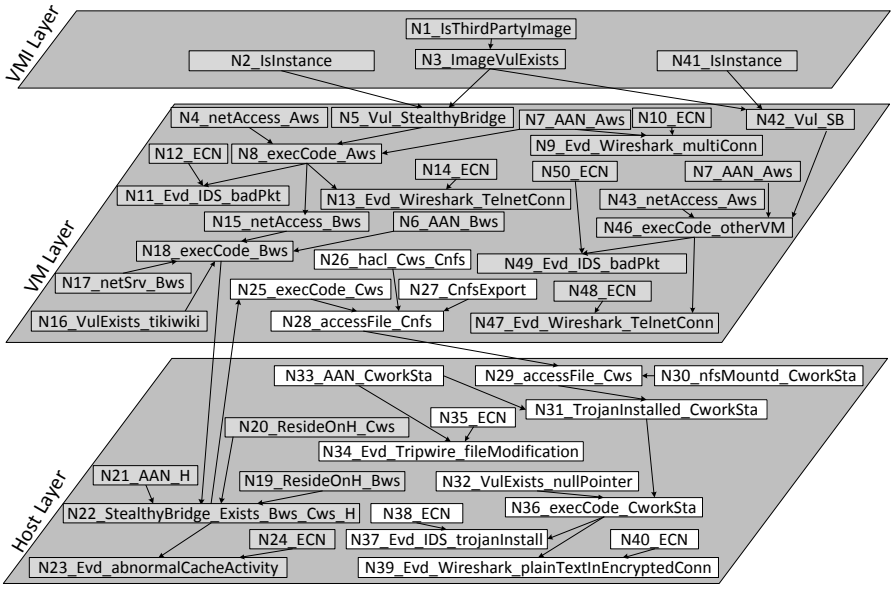


Fig. 8: The Cross-Layer Bayesian Network Constructed for the Attack Scenario

on A's web server (N8), a stealthy bridge exists in the host that B's web server reside (N22), and the attacker can execute arbitrary code on C's web server (N25). N8 and N25 respectively imply that the stealthy bridges in N5 and N22 are successfully established. Table 4 shows the results of experiment 1 given supporting evidence with corresponding confidence values. The results indicate that the probability of stealthy bridge existence is initially very low, and increases as more evidence is collected. For example, $Pr(N5 = True)$ increases from 34% with no evidence observed to 88.95% given all evidence presented. This means that a stealthy bridge is very likely to exist on enterprise A's web server after enough evidence is collected.

The first stealthy bridge in our attack scenario is established in attack step 2, and the corresponding pieces of evidence are N9, N11, and N13. $Pr(N8 = True)$ is 95.77% after all the evidence from step 2 is observed, but $Pr(N5 = True)$ is only 74.64%. This means that although the BN is almost sure that A's web server has been compromised, it doesn't have the same confidence of attributing the exploitation to the stealthy bridge, which is caused by the unknown vulnerability inherited from a VMI. $Pr(N5 = True)$ increases to 88.95% only after evidence N47 and N49 from other enterprise networks is observed for attack step 7. This means that if the same alerts appear in other instances of the same VMI, the VMI is very likely to contain the related unknown vulnerability.

The second stealthy bridge is established in step 4, and the corresponding evidence is N23. $Pr(N22 = True)$ is 57.45% after evidence N9 to N23 is collected. The number seems to be low. However, considering the unusual difficulty of leveraging a co-residency relationship, this low probability still should be treated with great attention. After all evidence is observed, the increase of $Pr(N22 =$

$True$) from 13.91% to 73.29% may require security experts to carefully scrutinize the virtual machine isolation status on the related host.

Table 4: Results of Experiment 1

Events	No evidence	N9 Medium	N11 High	N13 High	N23 High	N34 VeryHigh	N37 High	N39 VeryHigh	N47 VeryHigh	N49 VeryHigh
N5=True	34%	34%	51.54%	74.64%	75.22%	75.22%	75.41%	75.5%	86.07%	88.95%
N8=True	20.25%	22.96%	54.38%	95.77%	96.81%	96.81%	97.14%	97.31%	98.14%	98.37%
N22=True	13.91%	14.32%	19.03%	25.23%	57.45%	57.45%	67.67%	73.04%	73.24%	73.29%
N25=True	17.52%	17.89%	22.13%	27.71%	56.7%	56.7%	68.11%	74.1%	74.27%	74.32%

Experiment 2 tests the influence of false alerts to BN. In this experiment, we assume evidence N11 is a false alert generated by IDS. We perform the same analysis as in experiment 1 and compare results with it. Table 5 shows that when only 3 pieces of evidence (N9, N11, and N13) are observed, the probability of the related event is greatly affected by the false alert. For instance, $Pr(N5 = True)$ is 74.64% when N11 is correct, and is 53.9% when N11 is a false alert. But $Pr(N8 = True)$ is not greatly influenced by N11 because it's not closely related to the false alert. When all evidence is input into the BN, the influence of false alerts to related events is reduced to an acceptable level. This shows that a BN can provide relatively correct answer by combining the overall evidence set.

Table 5: Results of Experiment 2

Events	with 3 pieces of evidence		with all evidence	
	N11=True	N11=False	N11=True	N11=False
N5	74.64%	53.9%	88.95%	79.59%
N8	95.77%	58.6%	98.37%	79.07%
N22	25.23%	19.66%	73.29%	68.62%
N25	27.71%	22.7%	74.32%	70.24%

Since security experts may change their confidence value towards evidence based on their new knowledge and observation, experiment 3 tests the influence of evidence confidence value to the BN. This experiment generates similar results as in experiment 2, as shown in Table 6. When evidence is rare, the confidence value changes from VeryHigh to Low has larger influence to related events than when evidence is sufficient.

Table 6: Results of Experiment 3

Events	with 3 pieces of evidence		with all evidence	
	N14=VeryHigh	N14=Low	N14=VeryHigh	N14=Low
N5	74.64%	54.29%	88.95%	79.82%
N8	95.77%	59.30%	98.37%	79.54%
N22	25.23%	19.77%	73.29%	68.73%
N25	27.71%	22.79%	74.32%	70.34%

In experiment 4, we test the affect of evidence input order to the BN analysis result. We bring forward the evidence N47 and N49 from step 7 and insert them before N23 and N37 respectively. The analysis shows that a BN can still produce reliable results in the presence of changing evidence order.

6 Related Work

We explore the literature for the following topics that are related to our paper.

VMI sharing. [34] explores a variety of attacks that leverage the virtual machine image sharing in Amazon EC2. Researchers were able to extract highly sensitive information from publicly available VMIs. The analysis revealed that 30% of the 1100 analyzed AMIs (Amazon Machine Images) at the time of the

analysis contained public keys that are backdoors for the AMI Publishers. The backdoor problem is not limited to AMIs created by individuals, but also affects those from well-known open-source projects and companies.

Co-Residency. The security issues caused by virtual machine co-residency have attracted researchers' attention recently. [11] pointed out that the shared resource environment of cloud will introduce security issues that are fundamentally new and unique to cloud. [5] shows how attackers can identify on which host a target virtual machine is likely to reside in Amazon EC2, and then place the malicious virtual machine onto the same host through a number of instantiating attemps. Such co-residency can be used for further malicious activities, such as launching side-channel attack to extract information from a target virtual machine [6]. [10] takes an opposite perspective and proposes to detect co-residency via side-channel analysis. [4] demonstrates a new class of attacks called resource-freeing attacks (RFAs), which leverage the performance interference of co-resident virtual machine. [8] presents a traffic analysis attack that can initiate a covert channel and confirm co-residency with a target virtual machine instance. [7] also considers attacks towards hypervisor and propose to eliminate the hypervisor attack surface through new system design.

Bayesian Networks. BNs have been applied to intrusion detection [35] and cyber security analysis in traditional networks [23]. [23] analyzes which hosts are likely to be compromised based on known vulnerabilities and observed alerts. Our work lands on a different cloud environment and takes a reverse strategy by using BN to infer the stealthy bridges, which are unknown in nature. In the future, the inference of stealthy bridges can be further extended to identify the zero-day attack paths in cloud, as in [9] for traditional networks.

7 Conclusion and Discussion

This paper identifies the problem of stealthy bridges between isolated enterprise networks in the public cloud. To infer the existence of stealthy bridges, we propose a two-step approach. A cloud-level attack graph is first built to capture the potential attacks enabled by stealthy bridges. Based on the attack graph, a cross-layer Bayesian network is constructed by identifying uncertainty types existing in attacks exploiting stealthy bridges. The experiments show that the cross-layer Bayesian network is able to infer the existence of stealthy bridges given supporting evidence from other intrusion steps. However, one challenge posed by cloud environments needs further effort. Since the structure of cloud is very dynamic, generating the cloud-level attack graph from scratch whenever a change happens is expensive and time-consuming. Therefore, an incremental algorithm needs to be developed to address such frequent changes such as virtual machine turning on and off, configuration changes, etc.

Disclaimer

Acknowledgements

This work was supported by ARO W911NF-09-1-0525 (MURI), NSF CNS-1223710, NSF CNS-1422594, ARO W911NF-13-1-0421 (MURI), and AFOSR W911NF1210055.

References

[1] Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/
[2] Rackspace. http://www.rackspace.com/
[3] Windows Azure: Microsoft's Cloud. https://www.windowsazure.com/en-us/
[4] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, Resource-freeing attacks: improve your cloud performance (at your neighbors expense), ACM CCS 2012.
[5] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, ACM CCS 2009.
[6] D. X. Song, D. Wagner, and X. Tian, Timing Analysis of Keystrokes and Timing Attacks on SSH., in USENIX Security, 2001.
[7] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, Eliminating the Hypervisor Attack Surface for a More Secure Cloud, ACM CCS 2011.
[8] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler, Detecting co-residency with active traffic analysis techniques, in CCSW 2012.
[9] J. Dai, X. Sun, and P. Liu. "Patrol: Revealing Zero-Day Attack Paths through Network-Wide System Object Dependencies," ESORICS 2013.
[10] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis, IEEE S&P 2011.
[11] Y. Chen, V. Paxson, and R. H. Katz, Whats new about cloud computing security. University of California, Berkeley Report No. UCB/EECS-2010-5 January, 2010.
[12] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, Automated generation and analysis of attack graphs, IEEE S&P 2002.
[13] C. R. Ramakrishnan, R. Sekar, and others, Model-based analysis of configuration vulnerabilities, Journal of Computer Security, vol. 10, no. 1/2, pp. 189209, 2002.
[14] C. Phillips and L. P. Swiler, A graph-based system for network-vulnerability analysis, in Proceedings of the 1998 workshop on New security paradigms, 1998.
[15] S. Jajodia, S. Noel, and B. OBerry, Topological analysis of network attack vulnerability, Managing Cyber Threats, pp. 247266, 2005.
[16] P. Ammann, D. Wijesekera, and S. Kaushik, Scalable, graph-based network vulnerability analysis, ACM CCS 2002.
[17] K. Ingols, R. Lippmann, and K. Piwowarski, Practical attack graph generation for network defense, ACSAC 2006.
[18] X. Ou, W. F. Boyer, and M. A. McQueen, A scalable approach to attack graph generation, ACM CCS 2006.
[19] X. Ou, S. Govindavajhala, and A. W. Appel, MulVAL: A logic-based network security analyzer, USENIX Security, 2005.
[20] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro, A security analysis of Amazon's elastic compute cloud service, ACM SAC, 2012.
[21] K. Lazri, S. Laniepce, and J. Ben-Othman, Reconsidering Intrusion Monitoring Requirements in Shared Cloud Platforms, ARES 2013.
[22] http://www.snort.org/.
[23] Peng Xie, Jason Li, Xinming Ou, Peng Liu, and Renato Levy. "Using Bayesian networks for cyber security analysis," DSN 2010.
[24] http://www.tenable.com/products/nessus.
[25] http://nvd.nist.gov/.
[26] http://nvd.nist.gov/cvss.cfm.
[27] http://cve.mitre.org/.
[28] http://www.tripwire.com/.
[29] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2446.
[30] https://www.samba.org.
[31] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5423.
[32] https://info.tiki.org/.
[33] http://reasoning.cs.ucla.edu/samiam/.
[34] S. Bugiel, S. Nrnberger, T. Pppelmann, A.-R. Sadeghi, and T. Schneider, AmazonIA: when elasticity snaps back, ACM CCS 2011.
[35] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. "Bayesian event classification for intrusion detection." ACSAC 2003.