EAI Endorsed Transactions
on Security and Safety

Research Article EAI.EU

# Probabilistic Inference of the Stealthy Bridges between Enterprise Networks in Cloud

Xiaoyan Sun[1,*], Jun Dai[1], Anoop Singhal[2], Peng Liu[3]

[1]California State University, Sacramento, CA 95819, USA
[2]National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899, USA
[3]The Pennsylvania State University, University Park, PA 16802, USA

## Abstract

Cloud computing, with the paradigm of computing as a utility, has the potential to significantly tranform the IT industry. Attracted by the high efficiency, low cost, and great flexibility of cloud, enterprises began to migrate large parts of their networks into cloud. The cloud becomes a public space where multiple "tenants" reside. Except for some public services, the enterprise networks in cloud should be absolutely isolated from each other. However, some "stealthy bridges" could be established 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. Cloud-level attack graphs are firstly built to capture the potential attacks enabled by stealthy bridges and reveal hidden possible attack paths. Cross-layer Bayesian networks are then constructed to infer the probability of stealthy bridge existence. The experiment results show that the cross-layer Bayesian networks are capable of inferring the existence of stealthy bridges given supporting evidence from other intrusion steps in a multi-step attack.

Received on 25 December 2017; accepted on 26 December 2017; published on 4 January 2018

**Keywords:** cloud, stealthy bridge, Bayesian network, attack graph



doi:10.4108/XX.X.X.XX

## 1. Introduction

To gain various benefits in terms of efficiency, cost and flexibility, enterprise networks are now moving some of their parts (such as web server, database server, etc.) from traditional infrastructure into cloud 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 provide highly available and scalable services by acquiring computing resources easily and instantly. Specifically, in the service model of Infrastructure-as-a-Service (IaaS), cloud providers provide resources such as storage, network and platforms in the forms of virtual machines. Cloud customers, such as the enterprises, can readily construct the entire virtual network infrastructure by renting a number of virtual machines. As the demand for computing resources changes according to the business volume, the network size can grow or shrink through simply adjusting the number of rented machines.

Although attractive in many aspects, moving into cloud also introduces security issues that are yet to be solved. One major threat is posed due to multiple tenants "living" in the same public space of cloud. Generally speaking, a public cloud can provide virtual infrastructures to many enterprises. Except for providing some public services such as web services, an enterprise network is normally expected to be like an isolated island: connections from the outside network to the protected internal network are prohibited. However, in the cloud environment, virtual machines rented by different enterprises may reside on the same cloud, and even on the same host machine. Consequently, some "stealthy bridges" can be created by attackers between the isolated enterprise network islands. As shown in

*Corresponding author. Email: xiaoyan.sun@csus.edu

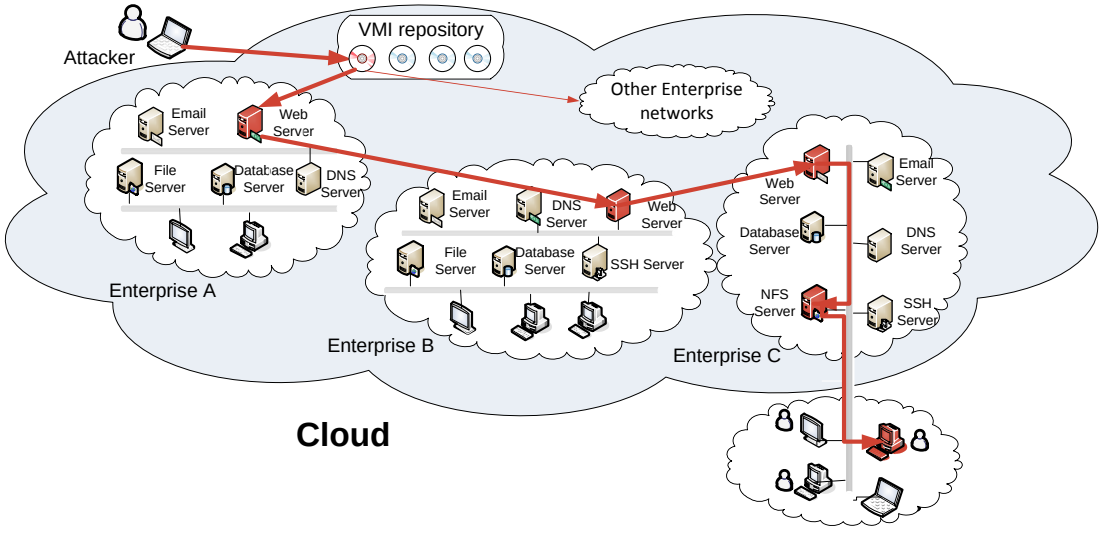



**Figure 1.** The Attack Scenario

Figure 1, attackers are able to establish stealthy bridges connecting several enterprise networks. Once the isolation among enterprise networks is penetrated, information confidentiality could be violated. Moreover, with the stealthy bridges, the attack path confined inside an enterprise network is able to traverse to another enterprise network in cloud, or even extend to the traditional infrastructures in a hybrid network. In this situation, the compromised virtual machines involved in a stealthy bridge become stepping stones for a multi-step cross-network attack. Therefore, stealthy bridges pose non-trivial threats to the security of enterprise networks residing in the cloud.

Two unique features of the public cloud enable the creation of stealthy bridges. 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 (if any) from the parent image, such as the security configurations and vulnerabilities. The countermeasure is to fix these problems through vulnerability patching, or re-configuration, etc, as soon as the VMIs are instantiated. In actual fact, however, these problems are usually ignored by normal users, which make the instance virtual machines remain vulnerable. Therefore, if an attacker creates and shares a malicious VMI that contains security holes, and the malicious VMI is later instantiated by an innocent user in an enterprise network, then it's like moving the attacker's machine directly into the internal network, without triggering the security sensors such as Intrusion Detection Systems (IDSs) or the firewall. In this case, a "stealthy bridge" can be created via security holes that bypass the hypervisor (e.g., 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. Since virtual machines on the same host transparently share the physical resources, the co-residency relationship can enable security problems such as information leakage, performance interference [4], or even co-resident virtual machine crashing. To detect and confirm the co-residency relationship, a number of techniques such as side-channel analysis or traffic analysis can be employed. 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 via a number of techniques, such as the 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]. Therefore, our strategy is to leverage the evidence (e.g. abnormal system activities, alerts, etc.) captured in other intrusion steps to infer the existence of stealthy bridges. 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 work are as follows:

First, a cloud-level attack graph is built to 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.

The preliminary version of this paper has appeared in [9]. In this paper, we extend the previous version and add new experiments to further evaluate our approach. The remainder of this paper is organized as follows. In Section 2, we review the research work related to this paper. In Section 3, we explain the existing attack graph models and introduce the cloud-level attack graph model. In Section 4, the cross-layer Bayesian network is presented with four types of identified uncertainties. Section 5 describes the implementation details for cloud-level attack graph generation and Bayesian network construction. In Section 6, we demonstrate the experiment results. More experiments are added in this section compared to [9]. Section 7 concludes the paper.

## 2. Related Work

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

**VMI sharing**. [42] 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. [12] 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]. [11] 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

[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.

to eliminate the hypervisor attack surface through new system design.

**Attack Graphs.** The attack graph is the basis graph model in our paper to capture the combined vulnerability exploits. There are mainly two representation types of attack graph [13, 14]: state enumeration attack graph (also called network-state attack graph [23]) and dependency attack graph. At the early development stage, state enumeration attack graph is the main stream. [15–21] are research works regarding state enumeration attack graph. Because of the serious complexity problem of state enumeration attack graph, researchers began to develop the new dependency attack graph. Representative works in dependency attack graph are [13, 22–27].

**Bayesian Networks.** BNs have been applied to intrusion detection [43] and cyber security analysis in traditional networks [30]. [30] 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 [10] for traditional networks.

## 3. Cloud-level Attack Graph Model

To reflect the potential attacks that could be enabled by stealthy bridges, a cloud-level attack graph should be built first as the basis for subsequent Bayesian network construction.

### 3.1. Attack Graph

Attack graph is a widely accepted solution for network vulnerability analysis. Although a number of vulnerability scanners are available for dealing with known vulnerabilities, such as Nessus[31] and OVAL interpreter [32], these scanners view vulnerabilities as isolated from each other. This is problematic considering the fact that individual vulnerabilities can be combined together for penetrating a network. Keep patching and perfectly secure one single host is useless for secure the entire network. Unfortunately, few of the security tools provide information about how attackers could combine vulnerabilities to achieve an attack goal. Even for experienced security experts or network administrators, it is quite difficult to construct an attack scenario solely by reading the vulnerability scan reports. The situation becomes even worse when it comes to big enterprise networks with hundreds to thousands of hosts.

The 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 leveraging these vulnerabilities. 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 using the acquired privilege.

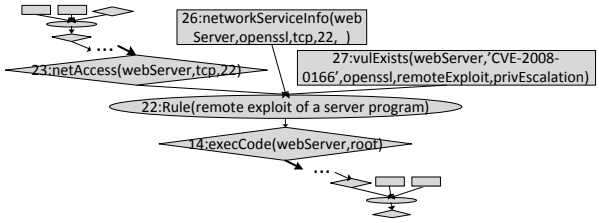



**Figure 2.** A Portion of an Example Logical Attack Graph

A variety of attack graphs have been developed for vulnerability analysis, mainly including state enumeration attack graphs [17, 18, 21] and dependency attack graphs [23–25]. In both types, the attack graph is represented with a directed graph G(V,E), where V is the set of vertex and E is the set of directed edges. The difference between state enumeration and dependency attack graphs lies in the semantic meaning of vertices and edges.

In state enumeration attack graph, a vertex represents one state of the entire network and the edges represent the transition between vertices. The network state will transit from one to another due to attacker actions. For example, $s_1 \rightarrow s_2 \rightarrow s_3$ is an attack path meaning that the network state transits from state $s_1$ to $s_2$, and then to state $s_3$. The transition happens when some specific conditions are enabled, but the transition edges do not directly show when these conditions are first enabled. The order of exploits is also considered in state enumeration attack graph.

In the dependency attack graph, a vertex represents a system condition rather than the entire network state. The edges between vertices represent the causality relationship. The dependency attack graph clearly shows what are the required preconditions for a post-condition to become true, and how the causality relationship takes effect and enables the attack to step forward. A single independent exploit appears only once in the graph and the order of exploits is not considered. This makes the dependency attack graph more succinct and easy to understand.

Our paper employs logical attack graph, which is a type of dependency attack graph. Figure 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

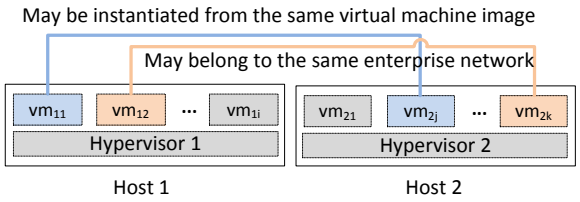



**Figure 3.** Features of the Public Cloud Structure

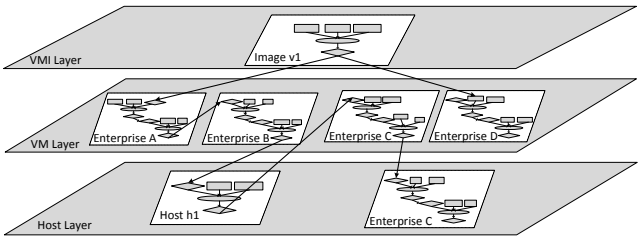



**Figure 4.** An Example Cloud-level Attack Graph Model

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 [27] that yields the derived fact.

For example, in Figure 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.

## 3.2. Cloud-level Attack Graph

When applying attack graphs to cloud, each enterprise network on cloud can scan its own virtual machines for existing vulnerabilities and generate an individual 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 entire cloud-level attack graph may not be complete due to unawareness of existing stealthy bridges. 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 Figure 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. In actual fact, an attack path indeed exists. 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 another word, 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 Figure 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, most servers of an enterprise network could be implemented in the cloud, while the personal computers and workstations could be in the traditional network infrastructure.

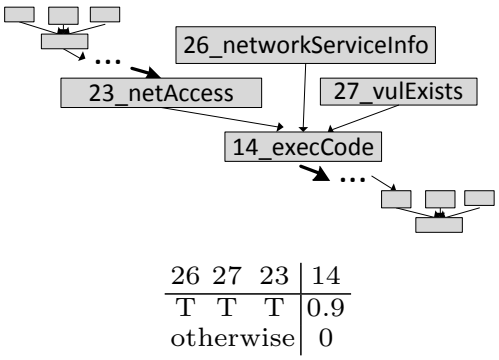
Therefore, taking the above characteristics of cloud structure into account, our cloud-level attack graph is modeled as follows.

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 Figure 4. With these layers, the attack graphs are not only about the individual virtual machines, but also include potential attacks at the VMI level and the host level.

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 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. Without the VMI layer and the host layer, such individual attack graphs can be scattered. 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 Figure 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. Bugiel et al. explores a number of attacks that take advantage of the VMI sharing in Amazon EC2 [42]. Evidence from [28] 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 parent VMI is identified with a specific vulnerability, it will affect all its children virtual machine instances, and thus all the relevant enterprise networks. As a result, the attack graphs involving this VMI will be affected at the VMI layer. Besides, the attack graphs for the children virtual machines are influenced as well: a precondition node could be activated, or a new interaction rule should be constructed in the 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.




| 26 | 27 | 23 | 14 |
|---|---|---|---|
| T | T | T | 0.9 |
| otherwise | | | 0 |

**Figure 5.** A Portion of Bayesian Network with associated CPT table

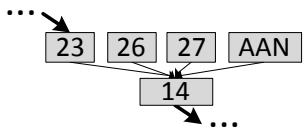
4) The host layer mainly captures the stealthy bridges caused by virtual machine co-residency and other exploits towards the physical hosts. Exploits on this layer could lead to further penetrations on the virtual machine layer. In addition, this layer actually captures all attacks that could happen 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 Figure 4, the attack graph for enterprise C extends from virtual machine layer to host layer.

## 4. Cross-layer Bayesian Networks

Since the stealthy bridges are hard to detect, our strategy is to infer the ex- istence by leveraging the evidence collected from other intrusion steps before and after the stealthy bridges. The Bayesian network is the proper tool for incorporating such evidence and performing probability inference.

### 4.1. 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




| 26 | 27 | 23 | AAN | 14 |
|---|---|---|---|---|
| T | T | T | T | 0.9 |
| otherwise | | | | 0 |

**Figure 6.** A Portion of Bayesian Network with AAN node

table (CPT). For example, Figure 5 shows a portion of a BN constructed directly from the attack graph in Figure 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.

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.

## 4.2. 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. Although the attack graph shows the potential attack paths, it cannot quantitatively analyze which path is ongoing or has been completed. The reason is that attack graphs only perform *deterministic* logic reasoning and do not consider the uncertainties associated with attacks. For example, in an attack graph, 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 [29] 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? Bayesian network is able to address such uncertainties existing in the real-time security analysis process.

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.

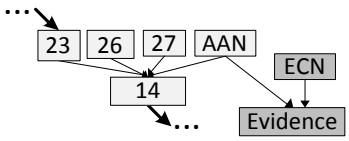
**Uncertainty of stealthy bridges existence.** The presence of known vulnerabilities is usually deterministic due to the availability of vulnerability scanners. After scanning a virtual machine or a physical host, the vulnerability scanner such as Nessus [31] is able to tell whether a known vulnerability exists or not[2]. Nonetheless, due to the unknown or hard-to-detect feature of stealthy bridges, effective scanners for them 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 [30]. 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 Figure 5 is changed to

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




| 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 |

**Figure 7.** The Evidence-Condidence Pair and Associated Exemplar CPT

Figure 6 after adding an AAN node. Correspondingly, the CPT table is modified as in Figure 6. This means "attacker taking action" is another prerequisite to be satisfied for the attack to happen.

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 and the exploitation can be easy. While for some other vulnerabilities, the exploit is still in the proof-of-concept stage and no successful exploit has been demonstrated. Exploiting such vulnerabilities from scratch should be fairly difficult. 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 Figure 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) [33] maintains a CVSS [34] scoring system for all CVE [35] 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 [30]. For example, the parameters in CPT table for Node 14 can be determined according to the corresponding AC value of the involved vulnerability.

**Uncertainty of evidence.** Evidence is the key factor for BN to function. In BN, uncertainties are indicated with probabilities 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. 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. 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 Figure 7. For another example, the alert "system log is deleted" given by Tripwire [36] 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 Figure 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 Figure 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.

## 5. Implementation

### 5.1. Cloud-level Attack Graph Generation

This paper uses *MulVAL* [27] 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.

### 5.2. 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.

**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 Figure 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* [41] 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.

## 6. Experiment

### 6.1. Attack Scenario

Figure 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* [37], existing in *Samba 3.0.0* [38], 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. [28] provides three possible ways for machine fingerprinting: ssh matching,

**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')).
```

service matching, and web matching. Through ssh key matching, Mallory finds the right instance in A and completes the exploitation towards *CVE-2007-2446* [37].

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* [39] from application *tikiwiki 1.9.8* [40], 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.

## 6.2. Experiment Results

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 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 Figure 1, the cross-layer BN is constructed as in Figure 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 Figure 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) |
| N16 | VulExists(Bws, 'CVE-2007-5423', tikiwiki, remoteExploit, privEscalation) |
| N17 | netServiceInfo(Bws, tikiwiki, http, 80, _) |
| N19 | ResideOn(Bws, H) |
| N20 | ResideOn(Cws, H) |
| N26 | 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 |

The preliminary version of this paper contains four sets of simulation experiments, each with a specific purpose [9]. In this paper, we add two more set of experiments to test how the evidence confidence value can mitigate the impact of false alerts, and to analyze the scalability of this approach. 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.

[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.

[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.

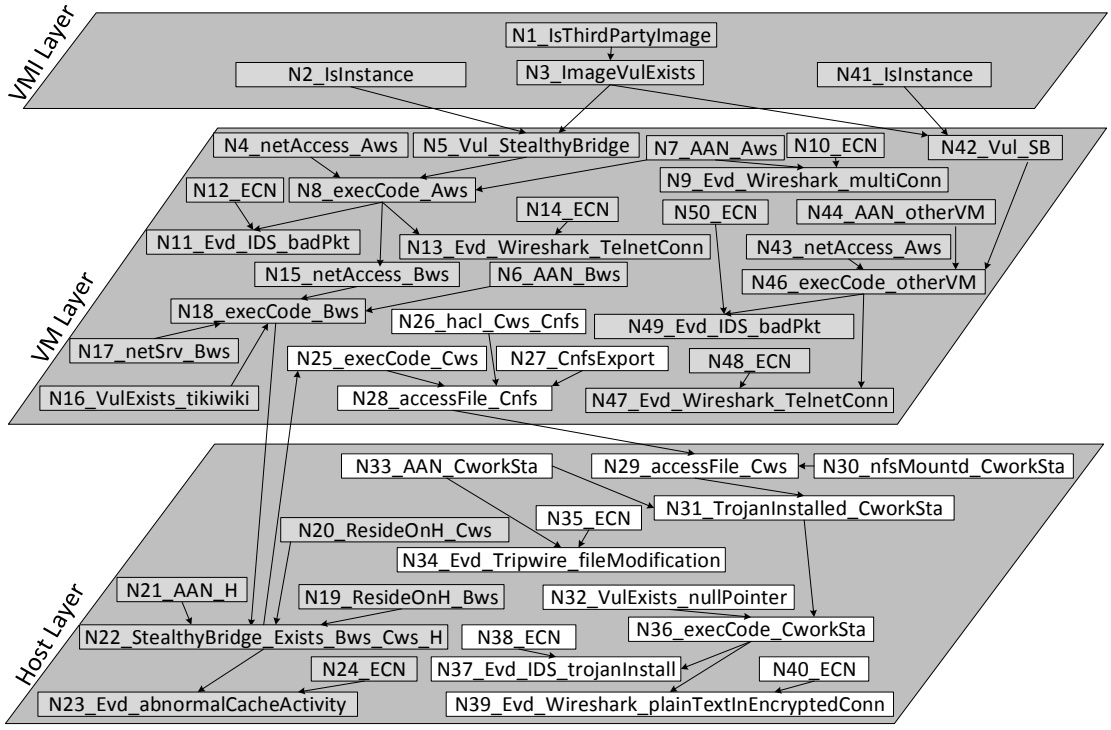



**Figure 8.** The Cross-Layer Bayesian Network Constructed for the Attack Scenario

By taking evidence with a certain order as input, the BN will generate a corresponding sequence of marginal probabilities for events of interest. The probabilities are compared with the ground truth to evaluate the performance of the BN.

**Experiment 1: Probability Inferring.**

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 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, marginal probability $Pr(N5 = True)$ 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.

**Experiment 2: Impact of False Alerts.**

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,

**Table 4.** Results of Experiment 1

| Events | No evidence | N9 Medium | N11 High | N13 VeryHigh | 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% |

**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% |

**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% |

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 the ratio of false alerts in the overall evidence set is an important factor determining the impact of false alerts. When the ratio of false alerts is low, a BN can provide relatively correct answer by combining the overall evidence set.

**Experiment 3: Impact of Evidence Confidence Value.**

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.

**Experiment 4: Impact of Evidence Input Order.**

In experiment 4, we test the affect of evidence input order to the BN analysis result (we assume the evidence is fed into BN immediately after it is collected). We bring forward the evidence N47 and N49 from step 7 and insert them before N23 and N37 respectively. The results in Table 7 show that when all the evidence from N9 to N39 is fed into BN, the final calculated probabilities are the same. This means, given the same set of evidence, BN will generate the same result regardless of the input order of evidence. However, this doesn't imply that the input order of evidence is not important for *real-time* security analysis. For example, in both Table 4 and Table 7, N23 is the crucial evidence for determining Pr(N22 = True). If N23 is collected at an early stage of the attack, the relatively high value of Pr(N22 = True) generated by BN may alert network defenders to check the involved virtual machines and hosts. As a result, the potential damage and loss to the victim enterprise network could possibly be mitigated or even stopped. Therefore, promptly collecting and feeding the evidence into BN is vital for real-time security analysis.

**Experiment 5: Mitigate Impact of False Alerts by Tuning Evidence Confidence Value.**

As evaluated in experiment 2, the ratio of false alerts in the overall evidence set is an important factor determining the impact of false alerts. However, in real security analysis, the ratio of false alerts is usually not a parameter that can be adjusted. In most cases, it is determined by the deployed security sensors and will not change significantly. For example, if an enterprise network deploys an IDS that suffers from high false rates, the ratio of false alerts in the overall evidence set will also be relatively high. The ratio will generally remain unchanged unless the security sensor is replaced. Hence, given such relatively stable ratio, it is important to find another way to mitigate the impact of false alerts. Tuning the evidence confidence value is one solution.

In experiment 5, we still assume evidence N11 is a false alert generated by IDS and only 3 pieces of evidence (N9, N11, and N13) are observed (so that the influence of confidence value towards impact of false alerts will be more evident). Table 8 shows the computed probabilities when the confidence value (specified in N12) for false alert N11 is "VeryHigh", "Medium", and "Low" respectively. When the confidence value is "VeryHigh", the false alert can generate great impact on the final results (e.g. Pr(N5 = True) is 76.49% when N11 is "True", and 34.00% when N11 is "False"). When the confidence value for false alert N11 is "Low", the false alert has little impact on the final result (e.g. the results for Pr(N5 = True) are very close: 69.96% when N11 is "True", and 67.09% when N11 is "False"). Therefore,

**Table 7.** Results of Experiment 4

| Events | No evidence | N9 Medium | N11 High | N13 VeryHigh | N47 VeryHigh | N23 High | N34 VeryHigh | N49 VeryHigh | N37 High | N39 VeryHigh |
|---|---|---|---|---|---|---|---|---|---|---|
| N5=True | 34% | 34% | 51.54% | 74.64% | 85.51% | 85.89% | 85.89% | 88.8% | 88.9% | 88.95% |
| N8=True | 20.25% | 22.96% | 54.38% | 95.77% | 97.07% | 97.8% | 97.8% | 98.06% | 98.27% | 98.37% |
| N22=True | 13.91% | 14.32% | 19.03% | 25.23% | 25.43% | 57.7% | 57.7% | 57.77% | 67.96% | 73.29% |
| N25=True | 17.52% | 17.89% | 22.13% | 27.71% | 27.89% | 56.93% | 56.93% | 56.99% | 68.37% | 74.32% |

**Table 8.** Results of Experiment 5

| Events | N12=VeryHigh | | N12=Medium | | N12=Low | |
|---|---|---|---|---|---|---|
| | N11=True | N11=False | N11=True | N11=False | N11=True | N11=False |
| N5 | 76.49% | 34.00% | 71.12% | 65.31% | 69.96% | 67.09% |
| N8 | 99.08% | 22.96% | 89.47% | 79.01% | 87.38% | 82.25% |
| N22 | 25.73% | 14.32% | 24.29% | 22.73% | 23.98% | 23.21% |
| N25 | 28.16% | 17.89% | 26.86% | 25.46% | 26.58% | 25.89% |

the impact of false alerts can be mitigated by tuning the corresponding confidence value for the evidence. In practical application, if a security sensor suffer from high false rates, the evidence generated by this sensor should have a relatively low confidence value. Similarly, evidence generated by security sensors with low false rates should have a relatively high confidence value. In such a way, the impact of false alerts can be mitigated in BN analysis.

**Experiment 6: Complexity.**

Since the BN is constructed on the basis of an attack graph, the size of BN mainly depends on the size of attack graph. According to Theorem 2 in [26], the logical attack graph for a network with N machines has a size at most $O(N^2)$. As we apply logical attack graph to cloud, we consider both virtual machines and physical hosts and regard them as normal hosts having special connections between each other. For a cloud with $n$ virtual machines and $m$ physical hosts, the corresponding attack graph has a size at most $O((n+m)^2)$. Considering $n >> m$ in a normal cloud, the size should be at most $O(n^2)$.

To further investigate the inference costs for BNs, we constructed 11 Bayesian networks with different size (Table 9) in SamIam. For most exact inference algorithms, the complexity of inference is mainly determined by the treewidth of the network. Nevertheless, determining the treewidth is also difficult. While we cannot explore all different tree structures and inference algorithms in this limited space, we provide the compilation costs for the BNs we constructed, as shown in Figure 9, to give readers a sense regarding the time and memory cost. The experiment was conducted in SamIam, with recursive conditioning as the inference algorithm adopted.

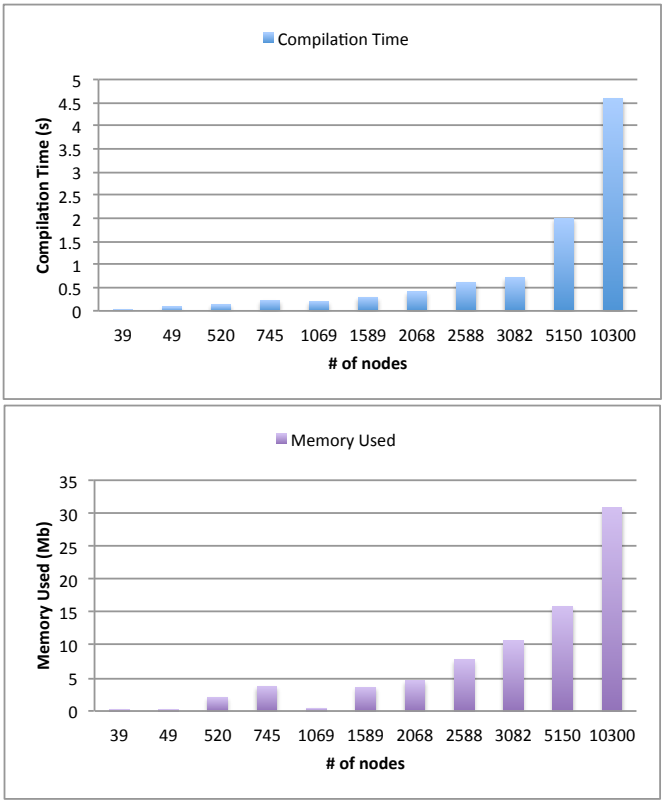



**Figure 9.** Time and Memory Used for BN Compilation

## 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. We designed and conducted six sets of experiments to evaluate our approach. The experiment results 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

**Table 9.** Size of Bayesian Networks

| BN | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # of nodes | 39 | 49 | 520 | 745 | 1069 | 1589 | 2068 | 2588 | 3082 | 5150 | 10300 |
| # of edges | 37 | 48 | 668 | 968 | 1244 | 1912 | 2545 | 3213 | 3854 | 6399 | 12798 |

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

This paper is not subject to copyright in the United States. Commercial products are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

## 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 neighbor's expense)*, in Proceedings of the 2012 ACM conference on Computer and communications security (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*, in Proceedings of the 2009 ACM conference on Computer and communications security (CCS), 2009.
[6] D. X. Song, D. Wagner, and X. Tian, *Timing Analysis of Keystrokes and Timing Attacks on SSH.*, in USENIX Security Symposium, 2001.
[7] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, *Eliminating the Hypervisor Attack Surface for a More Secure Cloud*, in Proceedings of the 2011 ACM conference on Computer and communications security (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 Proceedings of the 2012 ACM Workshop on Cloud computing security workshop (CCSW), 2012.
[9] X. Sun, J. Dai, A. Singhal, and P. Liu, *Inferring the Stealthy Bridges between Enterprise Network Islands in Cloud Using Cross-Layer Bayesian Networks*, in Proceedings of 10th International Conference on Security and Privacy in Communication Networks (SecureComm), 2014.
[10] J. Dai, X. Sun, and P. Liu, *Patrol: Revealing Zero-Day Attack Paths through Network-Wide System Object Dependencies*, in 2013 European Symposium on Research in Computer Security (ESORICS), 2013.
[11] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, *HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis*, in Proceedings of 2011 IEEE Symposium on Security and Privacy (S&P), 2011.
[12] Y. Chen, V. Paxson, and R. H. Katz, *What's new about cloud computing security*, University of California, Berkeley Report No. UCB/EECS-2010-5 January, 2010.
[13] S. Noel and S. Jajodia, *Managing attack graph complexity through visual hierarchical aggregation*, in Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, 2004.
[14] R. Sawilla and X. Ou, *Googling attack graphs*. Defence R&D Canada-Ottawa, 2007.
[15] O. M. Sheyner, *Scenario graphs and attack graphs*, University of Wisconsin, 2004.
[16] O. Sheyner and J. Wing, *Tools for generating and analyzing attack graphs*, in Formal methods for components and objects, 2004.
[17] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, *Automated generation and analysis of attack graphs*, in Proceedings of 2002 IEEE Symposium on Security and Privacy (S&P), 2002.
[18] C. R. Ramakrishnan, R. Sekar, *Model-based analysis of configuration vulnerabilities*, Journal of Computer Security, vol. 10, no. 1/2, 2002.
[19] S. Jha, O. Sheyner, and J. Wing, *Two formal analyses of attack graphs*, in Proceedings of the 15th IEEE Computer Security Foundations Workshop, 2002.
[20] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, *Computer-attack graph generation tool*, in Proceedings of DARPA Information Survivability Conference & Exposition II, 2001.
[21] 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.
[22] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, *Efficient minimum-cost network hardening via exploit dependency graphs*, in Proceedings of 19th Annual Computer Security Applications Conference (ACSAC), 2003.
[23] S. Jajodia, S. Noel, and B. O'Berry, *Topological analysis of network attack vulnerability*, Managing Cyber Threats, 2005.
[24] P. Ammann, D. Wijesekera, and S. Kaushik, *Scalable, graph-based network vulnerability analysis*, in Proceedings of the 2002 ACM conference on Computer and communications security (CCS), 2002.
[25] K. Ingols, R. Lippmann, and K. Piwowarski, *Practical attack graph generation for network defense*, in 22nd Annual Computer Security Applications Conference (ACSAC), 2006.
[26] X. Ou, W. F. Boyer, and M. A. McQueen, *A scalable approach to attack graph generation*, in Proceedings of the 2006 ACM conference on Computer and communications security (CCS), 2006.
[27] X. Ou, S. Govindavajhala, and A. W. Appel, *MulVAL: A logic-based network security analyzer*, in USENIX Security Symposium, 2005.
[28] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro, *A security analysis of Amazon's elastic compute cloud service*, in Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC), 2012.
[29] Snort. http://www.snort.org/.
[30] Peng Xie, Jason Li, Xinming Ou, Peng Liu, and Renato Levy. *Using Bayesian networks for cyber security analysis*, in Dependable Systems and Networks (DSN), IEEE/IFIP, 2010.
[31] Nessus. *http://www.tenable.com/products/nessus.*
[32] OVAL. *https://oval.mitre.org/*
[33] NVD. *http://nvd.nist.gov/.*
[34] CVSS. *http://nvd.nist.gov/cvss.cfm.*
[35] CVE. *http://cve.mitre.org/.*
[36] Tripwire. *http://www.tripwire.com/.*
[37] CVE-2007-2446. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2446.*
[38] SAMBA. *https://www.samba.org.*
[39] CVE-2007-5423. *http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5423.*
[40] tikiwiki. *https://info.tiki.org/.*
[41] SamIam. *http://reasoning.cs.ucla.edu/samiam/.*
[42] S. Bugiel, S. Nurnberger, T. Poppelmann, A.-R. Sadeghi, and T. Schneider, *AmazonIA: when elasticity snaps back*, in Proceedings of the 2011 ACM conference on Computer and communications security (CCS), 2011.
[43] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. *Bayesian event classification for intrusion detection.* in 19th Annual Computer Security Applications Conference (ACSAC), 2003.