Security Audits of Multi-tier Virtual Infrastructures in Public Infrastructure Clouds

Sören Bleikertz
IBM Research - Zurich
sbl@zurich.ibm.com

Matthias Schunter
IBM Research - Zurich
mts@zurich.ibm.com

Christian W. Probst
Technical University of Denmark
probst@imm.dtu.dk

Dimitrios Pendarakis
IBM T.J. Watson Research Center
dimitris@us.ibm.com

Konrad Eriksson
InfraSight Labs
konrad.eriksson@infrasightlabs.com

ABSTRACT

Cloud computing has gained remarkable popularity in the recent years by a wide spectrum of consumers, ranging from small start-ups to governments. However, its benefits in terms of flexibility, scalability, and low upfront investments, are shadowed by security challenges which inhibit its adoption. Managed through a web-services interface, users can configure highly flexible but complex cloud computing environments. Furthermore, users misconfiguring such cloud services poses a severe security risk that can lead to security incidents, e.g., erroneous exposure of services due to faulty network security configurations.

In this article we present a novel approach in the security assessment of the end-user configuration of multi-tier architectures deployed on infrastructure clouds such as Amazon EC2. In order to perform this assessment for the currently deployed configuration, we automated the process of extracting the configuration using the Amazon API. In the assessment we focused on the reachability and vulnerability of services in the virtual infrastructure, and presented a way for the visualization and automated analysis based on reachability and attack graphs. We proposed a query and policy language for the analysis which can be used to obtain insights into the configuration and to specify desired and undesired configurations. We have implemented the security assessment in a prototype and evaluated it for practical scenarios. Our approach effectively allows to remediate today's security concerns through validation of configurations of complex cloud infrastructures.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

CCSW'10, October 8, 2010, Chicago, Illinois, USA.

General Terms

Security

Keywords

Amazon EC2, cloud computing, reachability, attack graphs

1. INTRODUCTION

Cloud computing aims at providing standardized resources over a network that are perceived to provide unlimited scalability while being paid per use with limited up-front cost. These general principles of cloud computing can be implemented on different abstraction levels. While Infrastructure as a Service such as Amazon EC2 [3] provides virtual machines, storage, and networks, higher abstractions include Platform as a Service as well as Software as a Service that provide the actual web-based applications to end-users.

While the benefits are clear and end-users demand such services, security is a major inhibitor of cloud computing adoption on all levels of abstraction [16]. Today, public clouds such as Amazon's Elastic Compute Cloud (EC2) is used to host multi-tier infrastructures. Such infrastructures, e.g., comprise interconnected web, application, and database servers that may then be synchronized with databases in the enterprise. While this approach provides scalability, it may expose private personal or critical company data to attacks.

In order to mitigate such risks, security concepts similar to today's well-known security zones have been introduced. The so-called security groups of Amazon allow users to group machines while restricting communication through firewall-like rules. Nevertheless, the resulting configurations can be complex and security error prone. An indication of this fact is the complexity of correct firewall set-ups: When analyzing firewall configurations for 12 common mistakes [34], more than half of 37 cases exposed 9 of these 12 problems.

While security zones provide isolation and add robustness to the overall set-up, there are multiple sources of potential vulnerabilities in such a multi-tier cloud setting. In this paper we focus on end-user booting vulnerable images or creating erroneous and insecure configurations. Other risks are cloud providers incorrectly configuring or implementing their infrastructures as well as security incidents through insiders or component failures.

1.1 Our Contributions

In this paper we demonstrate how to audit the correct configuration of complex cloud infrastructures from an end-user perspective. Our approach allows to validate the correct set-up of security policies such as Amazon's security groups. This result allows to validate whether all servers in each tier can only be reached from the desired originating systems and, e.g., ensure that back-end database servers can indeed only be reached from the corresponding application servers.

We then show how to further assess the security of such an infrastructure. We automatically assess the vulnerabilities of each VM in an infrastructure and, by using attack graphs, compose these findings into an overall vulnerability assessment of the given multi-tier infrastructure.

Both results together then allow us to correct potential misconfigurations and to refine our multi-tier set-up to minimize the actual security risk as modeled by the range of exploitable vulnerabilities. Overall, this guarantees secure configuration as well as reduced vulnerabilities of our infrastructure deployed in the cloud.

Note that unlike earlier work (e.g., [23, 7]) we do not focus on how to securely implement a cloud. Given the Amazon cloud implementation we focus on how to securely use such an infrastructure without misconfiguration or creating additional vulnerabilities.

2. BACKGROUND

2.1 Scenario Illustrating an Multi-tier Infrastructure

Throughout this paper we will use the same scenario to illustrate the different audit and analysis methods. We consider an example configuration of a multi-tier web application widely used in real-world deployments consisting of web, application, and database servers. The web servers are reachable on the two common web server ports 80 (http) and 443 (https) over TCP from any source. The application servers are only reachable on an application specific port, e.g., 8080 TCP, from the web servers. Furthermore, the database servers are only reachable from the application servers on port 3306 (mysql) TCP. For maintenance purposes, all servers allow ssh access (22 TCP) from the corporate network, e.g., 1.2.3.4/24, and the servers accept ICMP packets from any source.

2.2 An Overview on Amazon Elastic Compute Cloud (EC2)

In this section we will explain relevant aspects of the Amazon architecture necessary for understanding the remaining parts of the paper.

Amazon's Elastic Compute Cloud (EC2).

EC2 is Amazon's service infrastructure cloud which allows customers to deploy and run virtual machines on Amazon's infrastructure. Virtual machines, also called *instances* in EC2, are provisioned from a machine template called *AMI* (Amazon Machine Image). A machine image contains an installation of an operating system and services required by the customer. Typically, virtual machines are directly connected to the Internet and protected by a firewall-like concept called *Security Groups*.

Amazon Security Groups.

Security Groups represent a set of inbound firewall rules associated with a name [4]. Outbound traffic is not restricted by a security group and always allowed. A virtual machine can be a member of one or multiple security groups, i.e., the traffic for that particular VM is allowed based on the union of rules specified in the associated security groups. Members of the same security group can only communicate with each other if explicitly allowed in the rules set.

The rules of a security group are applied in the management layer of the host, i.e., the firewalling is done outside the VM. Security groups can be used to simulate security perimeters like a DMZ or internal servers when using security groups as sources. The default firewall policy is *Deny*, therefore all rules in a security group are *Accept* rules. Rules can allow traffic based on protocol (TCP, UDP, ICMP), port range, and source (IP range or another security group).

3. REACHABILITY AUDIT OF AMAZON SECURITY GROUPS

In this section we will discuss the audit of security group configurations with regard to reachability, i.e., analyzing the information flow allowed by the configuration. Visualization of the allowed information flow and a reachability query language are presented which can support the administrator in developing new configurations and in discovering potential mistakes in the current configuration. A policy language and automated analysis are shown for the periodic verification of the configuration correctness.

3.1 Constructing a Reachability Graph

The current configuration of the security groups and related information, e.g., the security group membership of VMs, are obtained from Amazon using their API.

The visualization and the later proposed automated analysis are based on a directed multi-graph constructed from the configuration information obtained through the Amazon API. The vertices of the graph represent the set of sources and security groups defined in the configuration. The edges denote the allowed information flow specified in the rules of a security group between the sources and that particular group. For example, security group *web* allows the Internet, i.e., *0.0.0.0/0*, to access on port 80 TCP. The graph would consist of two vertices for the security group and Internet source IP with an directed edge between them labeled *80/tcp*.

3.2 Visualizing the Reachability Graph

Visualizing the reachability graph is useful for manual inspection of the correctness of the current security group configuration. According to [10], visualization takes advantage of vision, the highest bandwidth input device, and of the human perceptual abilities to make anomalies obvious to the user.

Listing 1 shows the output of the `ec2-describe-group` command, which displays the current security group configuration and is part of the command-line management suite of Amazon EC2. Even in such a simple example, it is difficult to assess the correctness of the given configuration. Amazon allows up to 100 defined security groups with multiple filter rules per group, which would result in a highly complex configuration that is even more difficult to evaluate.

In contrast Figure 1 illustrates the visualization of the

```
GROUP 1234  app application server
PERMISSION  1234  app ALLOWS  tcp 8080  8080  ↩
    FROM  USER  1234  GRPNAME web

GROUP 1234  db  database server
PERMISSION  1234  db  ALLOWS  tcp 3306  3306  ↩
    FROM  USER  1234  GRPNAME app

GROUP 1234  web web server
PERMISSION  1234  web ALLOWS  tcp 80  80  FROM  ↩
    CIDR  0.0.0.0/0
PERMISSION  1234  web ALLOWS  tcp 443 443 FROM  ↩
    CIDR  0.0.0.0/0

GROUP 1234  default default group
PERMISSION  1234  default ALLOWS  tcp 22  22  ↩
    FROM  CIDR  1.2.3.4/24
PERMISSION  1234  default ALLOWS  icmp  -1  -1  ↩
    FROM  CIDR  0.0.0.0/0
```

Listing 1: `ec2-describe-group` Command Output

reachability graph of the same configuration. In our opinion, the visualization can be more intuitively understood and judged for correctness. The multi-tier structure of the security groups is immediately obvious to the auditor and the external sources, i.e., IP ranges, are clearly pointed out.

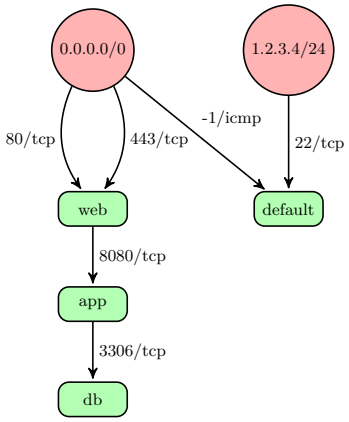

Figure 1: Visualization of Security Groups Reachability

Potential misconfigurations can easily be spotted in the visualization. For example, if the database security group would allow any source to access the database, rather than just the application group, it can easily be detected during the inspection due to an edge between the vertices *0.0.0.0/0* and *db*.

3.3 Understanding and Specifying Reachability

The visualization of the security group reachability is only useful for a manual inspection of the correctness of an initial security groups configuration. Afterwards, a periodic verification of the configuration against a policy specification is desired, in order to retain the correctness of the configuration, because changes in the configuration over its lifetime might cause violations with regard to its original intentions. Furthermore, queries can be used to answer questions about the reachability of the current configuration, e.g., for resolving reachability problems.

A Language for Reachability Queries.

Reachability queries can be specified in the following form: `from` s `to` d `port` p `proto` p'. s is either an IP address (specified as a single address or IP range), a security group, or *any* for matching all sources. d is either a security group or *any*. p can be one specific port or a port range $p_1 - p_2$. Both are transformed to a tuple (p_1, p_2), where in the former case $p_1 = p_2$. Valid values for p' are $tcp, udp, icmp$. p and p' can be set to *any* in case one is not interested in the specific port or protocol.

A Policy Language for Expressing the Expected Reachability.

For the policy language we will consider two cases. A *never* policy specifies a reachability which should never be established between a source and destination. An *only* policy allows only a specific reachability, i.e., the given information flow is allowed but any other flow is considered a violation of the policy. *never* policies are specified similarly to queries: `never from` s `to` d `port` p `proto` p'.

For an *only* policy we have to extend this syntax slightly to allow the specification of multiple port and protocol pairs per source and destination: `only from` s `to` d `port` p_1 `proto` p'_1 `and ... and port` p_n `proto` p'_n. Consider as an example a web server group which should only be reachable to port 80/tcp and port 443/tcp: `only from 0.0.0.0/0 to web port 80 proto tcp and port 443 proto tcp`.

3.4 Auditing a Configuration against a given Reachability Policy

The processing and verification of the reachability queries and policies is realized using two algorithms which are explained in the following.

Reachability Analysis.

The algorithm to process the reachability queries is given in Algorithm 1. We consider a sample input query: `from` s `to` d `port` $p_1 - p_2$ `proto` p', where the values $(s, d, (p_1, p_2), p')$ are passed as input parameters to the algorithm. The algorithm returns $True$ if the reachability specified in the query is established for a given reachability graph $G_R = (V, E)$.

Algorithm 1: Process a Reachability Query

Input: Reachability Graph G_R, Query $(s, d, (p_1, p_2), p')$
Output: $True$ or $False$ if query matches G_R

$E' \leftarrow \emptyset$
foreach $e \in E$ **do**
  if $(s = any$ ***or*** $s \subseteq e.source)$ ***and*** $(d = any$ ***or*** $d = e.destination)$ **then**
    $E' \leftarrow E' \cup \{e\}$
foreach $e \in E'$ **do**
  $c \leftarrow e.constraint$
  if $(p' = any$ ***or*** $p' = c.proto)$ ***and*** $(p_1 = any$ ***or*** $(c.p_1 \leq p_1$ ***and*** $p_2 \leq c.p_2))$ **then**
    return $True$
return $False$

In the first step, we construct a set of edges E' containing all edges between the given source and destination vertices. Since a source can be specified as an IP range, we have to check if the given source s is a subset of the source of the

edge, e.g., $10.0.0.0/24 \subset 0.0.0.0/0$. In case the source of the edge and s are security groups, $\subseteq$ acts as an ordinary equality operator. Otherwise, when either the source or destination is specified as *any*, we will include the edge regardless of the source or destination respectively.

In the second step, we compare the constraints of the edges in E' (indicated by *e.constraint*) with the constraints specified in the query. *any* for the ports and protocol short circuits the check, i.e., the query is only interested in an edge between s and d. In the other case, the protocols must be equal and the port range in the query enclosed in the port range of the edge constraint.

Policy Verification.

The process of the verification of reachability policies leverages the previously described query processing algorithm. For *never* policies in the form `never from` s `to` d `port` $p_1 - p_2$ `proto` p', we simply convert them to a query $(s, d, (p_1, p_2), p')$ which has to evaluate to $False$. Otherwise the given policy is not satisfied.

In case of an *only* policy like `only from` s `to` d `port` $p_{1,1} - p_{1,2}$ `proto` p'_1 `and ... and port` $p_{n,1} - p_{n,2}$ `proto` p'_n, we have to do a two step process for the verification. We convert the policy into n queries of the form $(s, d, (p_{i,1}, p_{i,2}), p'_i)$ which all have to evaluate to $True$. Otherwise the reachability specified in the policy is not satisfied. Furthermore, we have to verify that other information flows are not possible in the reachability graph for the particular source and destination, i.e., queries of the complement $(s, d, \bar{p}, \bar{p}')$ have to evaluate to $False$. This exclusiveness of the reachability specified in the *only* policy is checked with Algorithm 2.

Algorithm 2: Verify Exclusiveness of an *only* Policy

Input: Reachability Graph G_R, Policy $(s, d, [((p_{1,1}, p_{1,2}), p'_1), \ldots, ((p_{n,1}, p_{n,2}), p'_n)])$
Output: $True$ or $False$ if the policy is satisfied

[Construction of E' equal to Algorithm 1]

foreach $e \in E'$ **do**
  $B \leftarrow [True, True, \ldots]$
  $c \leftarrow e.constraint$
  for $i = 1$ ***to*** n **do**
    if *(*$p'_i \neq any$ ***and*** $p'_i \neq c.proto$*)* ***or*** *(*$p_{i,1} \neq any$ ***and*** *(*$c.p_1 < p_{i,1}$ ***or*** $p_{i,2} < c.p_2$*))* **then**
      $B_i \leftarrow False$
  if $True \notin B$ **then**
    return $False$
return $True$

The construction of the set of relevant edges E' is equal to the part in Algorithm 1. For each edge in this set, we initialize an array B, with a size equal to the number of port-protocol pairs specified in the policy, with $True$ values. For each such pair, we check the edge constraint if it allows further information flow than already allowed by the pair. For example the policy requires only port (p_1, p_2) but the edge constraint allows $(p_1 - 1, p_2 + 1)$, therefore allows further information flow with two more ports and thereby violates the *only* policy.

If the policy specifies *any* for the port or protocol then we can skip the constraint check. Otherwise, if the protocols of the policy and edge constraint are different, or the port range of the edge constraint is larger than the one specified in the policy, we mark a violation in the array B by setting the array slot corresponding to the port-protocol pair to $False$. After checking all pairs for a particular edge constraint, we test if B does not contain any $True$ value. If this is the case, then the policy is violated, because none port-protocol pair matched the edge constraint. In case of a non-violation, at least one pair would have matched and resulted in a $True$ value in B.

The periodic verification uses a policy specification consisting of a set of policies, which all have to evaluate to $True$, in order that the verification is successful and the configuration does not contain any violations.

4. ASSESSING THE VULNERABILITY OF AN AMAZON EC2 CONFIGURATION

While the technique presented in the previous section targeted the connectivity through single edges, we now extend this approach to inspect the overall attack vulnerability of the whole graph. Since pure reachability is a rather weak security measure, we extend the edges with a weight of how likely it is that they will be vulnerable to an attack. This results in a kind of attack graph [31, 32, 33]. The audit of security configurations using these attack graphs is concerned about the impact of security group rules with regard to services security, and is based on the previously presented reachability analysis.

4.1 Attack Graph representing the Vulnerability of EC2 Configurations

An attack graph for security groups consists of vertices based on IP ranges and AMIs (Amazon Machine Images), where the information flow between the vertices is given by the rules of the security groups the VMs — provisioned from the AMIs — are members of. Furthermore, the edges are labeled with a severity rating for the service running in the AMI and allowed by a security group rule.

For example, assume that *VM 1* is a member of security group *web* and provisioned from *AMI 1*. There exists a rule in *web* that allows *0.0.0.0/0* to access on port *443/tcp*. Suppose the web server running in *AMI 1* has a known vulnerability, then there would be an edge between *0.0.0.0/0* and *AMI 1* in the attack graph with a label *443/tcp medium*.

4.2 Constructing an Attack Graph through Discovery and Vulnerability Scanning

The previously described attack graph can be automatically constructed in three steps.

The first step is to establish the relationship between AMIs and security groups. Using the Amazon API, we can obtain the currently running VMs, including information on the AMIs used for provisioning them and the security groups they are members of. Typically, the number of different AMIs per security group should be rather small, because the role of the security group is reflected by a specific AMI, e.g., multiple instances of a web server AMI would be member of the *web* security group. In our example we assume the existence of four different AMIs; *AMI 1* is a member of security group *web*, *AMI 2* of *app*, *AMI 3* and *AMI 4* of *db*. Furthermore, all AMIs are member of *default*. Figure 2 illustrates the relationships between security groups and AMIs for our scenario.

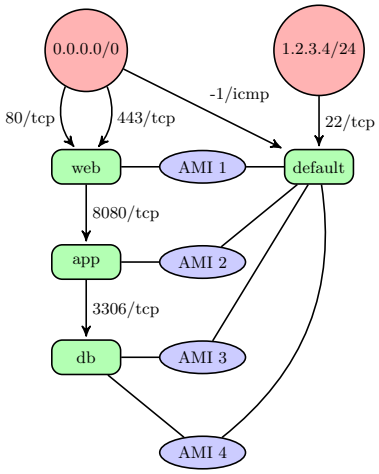

Figure 2: Relationship between Security Groups and AMIs

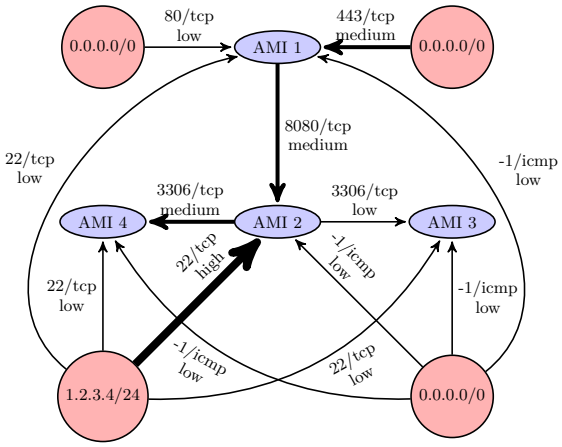

Figure 3: Attack Graph of the Multi-Tier Application

The second step in the construction is to obtain a severity rating for the services running in the AMIs. Using Amazon EC2 we can start each AMI in a separate VM for analysis purposes, thereby not affecting the production systems. The AMI's security from an external point of view can be determined in the VM using vulnerability scanners like Nessus [30]. Further information such as patch levels and version numbers can be obtained from an internal point of view by logging into the AMI instance. For each port of an AMI we thus obtain a severity rating from a domain of vulnerability ratings. Here we use the range of *Low*, *Medium*, or *High*. For typical applications this level of granularity seems to be sufficient, but a more fine-grained rating can be achieved using Common Vulnerability Scoring System (CVSS) [17].

The final step in the construction is to combine the previously obtained information with the reachability graph of security groups. A security group is replaced by all the AMIs related to that particular group, and the edges between the sources and the AMIs are additionally labeled with the severity rating for the port and protocol associated with each edge. Figure 3 illustrates the attack graph for the example scenario where thicker edges represent higher vulnerability ratings.

In the current configuration, an attacker in the corporate network could compromise a VM of the application server group, which are provisioned using *AMI 2*, through a vulnerability in the ssh service. Afterwards, further attacks can be launched against the medium rated mysql service running on instances of *AMI 4*, potentially compromising the database.

4.3 Understanding and Specifying acceptable Risk

When analyzing attack graphs constructed from cloud configurations, one is particularly interested in the *weakest* path from one vertex to another. The weakest path is the shortest path with the highest vulnerability rating, i.e., the most likely path an attacker would take to compromise a specific resource, because less vulnerabilities with high severity ratings are more likely to be exploited successfully.

Query Language to Understand Vulnerabilities.

Reachability queries are specified in the following form: `from` s `to` d `vuln` v. We drop protocols and ports in the query, since we are mostly interested in the vulnerability, not how an attack is performed. s is either an IP address (specified as a single address or IP range), an AMI, or *any* for matching all sources, and d is either an AMI or *any*. The vulnerability value v can be any value from the domain of vulnerability ratings, or the special value *any*. The result of such a query is a set of shortest paths P from s to d. If *any* is specified for v, then all paths are considered, otherwise only paths with a minimum vulnerability higher than or equal to v are considered.

Policy Language for Specifying Acceptable Vulnerabilities.

For the policy language we consider the same cases as in Subsection 3.3. As before, a *never* policy specifies an unwanted connection between a source and a destination. They are specified similarly to queries: `never from` s `to` d `vuln` v, where s, d, and v can have the same values as for queries. The interpretation of a *never* policy is that there may never be a connection with a vulnerability rating *higher* than or equal to v.

Unlike before, *only* policies are very similar to *never* policies when dealing with attack graphs. This is because we only want to restrict the vulnerability ratings allowed on connections between nodes. These policies have the form `only from` s `to` d `vuln` v, that is the only difference is the initial keyword. Again, s, d, and v can have the same values as for queries. The interpretation of an *only* policy is that there may only be connections with vulnerability ratings *lower* than or equal to v.

4.4 Vulnerability Audit using Weakest Path Algorithm

The analysis of queries on the attack graph, and the testing of attack graphs against policies, is performed by means of Dijkstra's shortest path algorithm on a weighted graph. The weight of the edges is based on the vulnerability rating where the weight relation is the following: $High < Medium < Low$. Since Dijkstra's algorithm will determine the shortest path

with the lowest weight, we will obtain the path of least resistance an attacker might take. In case the query or policy contains a vulnerability parameter unequal to *any*, annotated by $\overline{any}$, we have to transform the attack graph before performing the Dijkstra's algorithm by removing all edges with a vulnerability rating lower than the one specified.

Based on the different combinations of query/policy source and destination parameters, that is, $(\overline{any}, \overline{any})$, $(\overline{any}, any)$, $(any, \overline{any})$, and (any, any), we have to perform a different variation of the Dijkstra's algorithm. For the case $(\overline{any}, \overline{any})$, i.e., a specific source and destination, the Dijkstra's algorithm can be terminated upon finding the shortest path for the destination. In the other case where the destination is *any*, the regular algorithm will be performed which determines the shortest path between a single source and all other vertices. For $(any, \overline{any})$, i.e., the source is *any*, we reverse the attack graph and start the single source Dijkstra's algorithm from the destination. Due to the reversal of the attack graph, we also have to reverse the paths obtained by the Dijkstra's algorithm. In case of (any, any), all-pairs shortest paths is determined using Dijkstra's algorithm starting from every vertex. Alternatively, the Floyd-Warshall algorithm could be used to determine the all-pairs shortest paths.

The policy processing can be based on the approach of finding the weakest path. A *never* policy basically states the fact that no weakest path with the properties specified in the policy should exist. In case of *only* policies, they state that never should exist a path with a vulnerability *higher* than the one specified. For example, the policy specifies a medium vulnerability, then no path with a high vulnerability should exist.

4.5 Reducing Vulnerabilities Through Transformation

The purpose of the transformation process is to reduce the complexity of the configuration by removing unnecessary rules and extracting common ones, and to improve the security by splitting existing security groups.

Splitting of Groups.

Based on the attack graph and a specification of the desired services dependency, we can propose a new security group configuration by splitting existing security groups. The splitting process has to balance between increasing configuration complexity and security. One extreme case would be to place each AMI in a separate security group, therefore minimizing the affect of a vulnerability in one AMI to other AMIs. Generally, high or medium rated AMIs are isolated in separate groups, while AMIs with a low or no severity rating can remain in the same group.

Closing Unnecessary Open Ports.

Using the AMI analysis of open ports and the rules of the corresponding security group the instances of that AMI are a member of, we can identify unnecessary rules in the configuration if ports are opened in the configuration, but no service is listening on that particular port in the AMI instances. The difference of the set of ports opened by the security group and the set of ports used by the AMIs represents the ports which are unnecessarily opened. These ports can be automatically removed from the security group rules set during a transformation phase.

Extracting Common Ports.

A VM can be a member of several security groups and we can leverage this fact to reduce the complexity of the configuration by identifying common ports in the rules of the existing groups and extract them into a separate group. For example, if all groups would allow ssh access from a corporate network and allow ICMP packets, we could automatically extract these rules into a separate group. All instances would be additionally a member of that group. The principle is similar to *Refactoring* found in software engineering, where common functionality is extracted.

5. IMPLEMENTATION & EVALUATION

In this section we will shortly present our implementation of the security assessment. We will provide time measurements and a practical evaluation of the tool.

5.1 Prototype Implementation

We implemented the previously described construction of reachability and attack graphs, as well as the processing of queries and policies for such graphs in Python. The implementation, called *SAVEly*, was straight forward given the detailed algorithms presented earlier. We are using the *boto* [6] library for obtaining the configuration from Amazon EC2, and the *NetworkX* [18] library for the graph handling and shortest-path algorithms.

The attack graph construction is using the OpenVAS vulnerability scanner [20]. For each discovered AMI we spawn a new instance in a specialized *savely_scan* security group which allows all traffic from the scanner machine's IP address, instead of scanning the production servers directly. This approach of scanning will not affect the production servers due to aggressive scanning probes.

5.2 Performance Measurements

We will give time measurements for the analysis on sample deployments on Amazon using the SAVEly tool. The accuracy of the time measurements is not very important, because we mainly want to convey a general overview of the time consumptions rather than using the measurements for performance improvements and profiling. The measurements were obtained on a regular laptop machine with a 2.13GHz Pentium M processor, 2 GB of RAM, and running Gentoo Linux.

Reachability Graph Analysis.

The deployed configuration results in a reachability graph with *257 vertices* (resulting from the security groups) and *505 edges* (based on the security groups rules). We obtained the following measurements for the individual parts of the analysis:

- Obtain SGs from Amazon: $1.7s$
- Build reachability graph: $0.04s$
- Perform reachability queries (7): $0.025s$
- Perform policy check (2 never, 1 only): $0.01s$

Clearly obtaining the configuration from Amazon is the slowest part in the process, because a cryptographically signed transaction over the Internet to Amazon's API server has to be performed.

Attack Graph Analysis.

For the construction of the attack graph we use the OpenVAS [20] vulnerability scanner with 585 plugins enabled. Furthermore, we have to obtain the security groups and running instances from Amazon. We obtain the following measurements with one running instance found and scanned:

- Get Security Groups: $1.6s$
- Get Instances: $0.15s$
- Build Attack Graph: $2min52s$

The construction of the attack graph is by far the most time consuming part in the analysis process, because it involves starting a new instance at Amazon EC2 and performing a vulnerability scan, where the vulnerability scan is the dominating factor.

Since the constructed attack graph is rather simple, because it only involves one AMI, we are performing the analysis on an attack graph similar to the one presented in Figure 3. However, the ICMP edges were removed. We obtain the following time measurements for performing queries and policy checks on the attack graph:

- Queries (5): $0.01s$
- Policies (1 never, 1 only): $0.0015s$

Evidently for such simple attack graphs the query and policy checks can be performed in an instant.

The most time consuming part in the attack graph analysis is the vulnerability scan. A possible improvement for this step could be to parallelize it, i.e., start all instances at the same time and perform the vulnerability scan from an equal number of scanning instances. However the scanning instances can not be run on Amazon EC2, because their policy prohibits the port scan of other instances.

5.3 Effectiveness of Attack-graph-based Security Audits

We limit the evaluation of the tool's functionality to attack graphs due to space constraints. The attack graph analysis is more interesting from a security perspective than the reachability one. We consider the more complex attack graph shown in Figure 3 to demonstrate the analysis. In the first part we demonstrate the query language which supports an administrator to get an insight into the vulnerability of services in the current configuration. The second part deals with the policy language for specifying undesired properties of the configuration in terms of service vulnerabilities and attack paths.

A sample query file for the attack graph analysis is given in Listing 2. The queries test the service vulnerability exposed to the Internet and the corporate network, and the general vulnerability within the whole deployment.

```
from 0.0.0.0/0 to any vuln medium
from 0.0.0.0/0 to any vuln high
from 1.2.3.4/24 to any vuln medium
from any to any vuln high
from any to AMI2 vuln medium
```

Listing 2: Attack Query File

The output of the query analysis is given in Listing 3. For each query the tool also provides the most likely attack paths which fulfill the query. For example the first query, which tests which resources could be compromised by any attackers exploiting medium rated vulnerabilities, provides us with three potential paths resulting in the compromise of AMI1, AMI2, and AMI4.

```
from 0.0.0.0/0 to any vuln medium
=> [['0.0.0.0/0', 'AMI1', 'AMI2', 'AMI4'], ↩
    ['0.0.0.0/0', 'AMI1'], ['0.0.0.0/0', 'AMI1',↩
     'AMI2']]
True
from 0.0.0.0/0 to any vuln high
=> []
False
from 1.2.3.4/24 to any vuln medium
=> [['1.2.3.4/24', 'AMI2', 'AMI4'], ↩
    ['1.2.3.4/24', 'AMI2'], ['0.0.0.0/0', 'AMI1↩
    ', 'AMI2', 'AMI4'], ['0.0.0.0/0', 'AMI1'], ↩
    ['0.0.0.0/0', 'AMI1', 'AMI2']]
True
from any to any vuln high
=> [['1.2.3.4/24', 'AMI2']]
True
from any to AMI2 vuln medium
=> [['1.2.3.4/24', 'AMI2'], ['0.0.0.0/0', 'AMI1↩
    ', 'AMI2'], ['AMI1', 'AMI2']]
True
```

Listing 3: Attack Query Output

Now we demonstrate the ability of the tool to verify policies, i.e., to check for undesired behavior. A simple policy file is shown in Listing 4 which implements the idea that an administrator wants to be sure that no high rated vulnerabilities exists in the deployed multi-tier application and that at maximum low rated vulnerabilities are exposed to any attacker.

```
never from any to any vuln high
only from 0.0.0.0/0 to any vuln low
```

Listing 4: Attack Policy File

The output of the policy analysis is given in Listing 5. Reconsidering the query output previously shown, it is not surprising that both policies are violated by the current configuration. The second last query returned a potential attack paths using a high rated vulnerability between `1.2.3.4/24` and `AMI2`, which can be manually verified using the illustration of the attack graph in Figure 3. The second policy is violated because several attack paths for any attacker exist using medium rated vulnerabilities as shown using the first query.

```
Attack Policy valid:
 policy ('any', 'any', 'high') violation: ↩
     [['1.2.3.4/24', 'AMI2']]
 policy ('0.0.0.0/0', 'any', 'medium') violation↩
     : [['0.0.0.0/0', 'AMI1', 'AMI2', 'AMI4'], ↩
     ['0.0.0.0/0', 'AMI1'], ['0.0.0.0/0', 'AMI1↩
     ', 'AMI2']]
False
```

Listing 5: Attack Policy Output

The policy violation detection with given counter-examples will be very useful for an administrator to fix discovered vulnerabilities in the deployed complex multi-tier application.

6. RELATED WORK

6.1 Virtual Machine and Cloud Security

Information Leakage.

The multi-tenancy of virtualization and in particular cloud computing in form of infrastructure as a service, faces the scenario that an attacker will share the same physical resources as other tenants. This sharing of resources could lead to information leakage due to known or unknown covert channels.

A very interesting approach was presented in [23] which consists of a method for predicting the placements of VMs in the Amazon cloud and discussing potential side-channels and their implications. The placement is in particular interesting for attackers who target a specific victim and want to place a VM on the same physical server. Placing a VM on the same physical server, i.e., establishing co-residency, will allow further attacks using side-channel vulnerabilities to extract potentially sensitive information about the other VM. Work on side-channel vulnerabilities leading to the exposure of cryptographic keys were presented in [21] (Intel Hyper-Threading), [2] (CPU branch prediction), and [1] (I-Cache exploiting).

Another problem which could lead to information leakage is a result of the rollback functionality of VMs. Due to the rollback, cryptographic protocols could be affected either due to reuse of keys or reusing the same random numbers generated in an earlier run of the protocol. In [24] the issue of randomness problems in cryptographic protocols is discussed and it is shown how such a problem can be exploited in the case of TLS. In order to mitigate the problem related to randomness in virtual machines, they propose a framework for securing existing protocols by the means of hedged cryptography, which means that cryptographic protocols will provide a weaker security notion in the presence of a bad randomness source [5].

Remote Attestation.

Remote attestation tackles the problem of assuring that a remote platform consists of a trusted set of hardware and software resources. This is typically done in open distributed systems to ensure that the other peers are not running malicious software [15].

The *Terra* [9] architecture is based on a trusted VMM (TVMM) which provides two different execution contexts for VMs: open box and closed box. The first one is equivalent to a regular general-purpose hardware platform and the second one resembles a special-purpose platform which is typically found in closed systems like mobile phones and game consoles. The closed box environment provides, among other capabilities, remote attestation for assuring remote parties about the integrity of the hardware and software stack of a VM.

The usage of attestation in the area of cloud computing, i.e., trusted cloud computing was presented in [13] and [25]. The problem is that for cloud consumers the IaaS providers operate a black box and the consumer does not have any insights about the underlying security of the architecture. The architectures presented in the two papers use attestation to assure that a known and trusted software stack is in use and provides adequate security for the VMs. For example, [25] makes use of a trusted VMM as presented earlier.

An example for the usage of attestation in a real-world application is Enomaly's Elastic Computing Platform, which is a virtualized systems management software, in the *High Assurance Edition* [8].

6.2 Attack Graphs

Attack graphs are a way to model network risks in a network using a graph-based approach, where nodes represent a possible attack state, e.g., user privilege escalation, and the edges represent a way of changing states, e.g., using an exploit with certain pre- and post-conditions [29, 22]. An elaborate review of literature on attack graphs from the year 2005 can be found in [14].

Attack graphs can be constructed by security experts, however this manual approach becomes very difficult due to the scalability of the problem. Therefore, an automated approach for the construction and analysis of attack graphs is desired. In [28] an early tool for the construction and analysis of attack graphs was presented. It uses manually obtained information about attacker and exploit capabilities, and automatically gathered information about the machine configurations and vulnerabilities, in order to construct an attack graph. A shortest-path for the graph is computed which represents the most likely path an attack will take. A similar analysis using shortest-path was also discussed in [29, 22]. An approach based on symbolic model checking for automatic and efficient graph construction was presented in [26]. The authors of [27] demonstrate a toolkit for the analysis and construction of attack graphs.

Visualization of reachability, attack graphs, and attack paths is very important for administrators in order to effectively leverage these tools and apply the obtained results for improving the security of the network. Sample work in the visualization of attack graphs can be found in [33, 32].

6.3 Topological Vulnerability Analysis

One of the most comprehensive approaches in attack graph construction and analysis is *Topological Vulnerability Analysis* (TVA) presented in multiple publications [11, 12]. The main idea is to analyze dependencies of attack exploits, in order to find paths of an attacker to compromise specific targets in the network. Information from vulnerability scanners, which provide detailed information of isolated vulnerabilities, are gathered and combined. Based on the attack graphs they can propose changes in the configuration to increase the network security, and an interactive graph visualization tool can help administrators in finding network problems.

Only relying on a vulnerability scanner can limit the insights gained into the possible vulnerable services of a host. For example, client-side vulnerabilities are not covered by vulnerability scanners. Therefore, a new approach was presented in [19], which correlates information of a asset management database with a vulnerability database. For instance, a host is running a specific version of a web browser, which is noted in the asset database, and the vulnerability database contains information about that particular version of the web browser.

Several limitations of TVA were pointed out in [14]:

- Exploit information, i.e., pre- and post-conditions, entered by hand
- Firewall and router rules not automatically imported
- Poor scaling to large networks
- Requires low-level attack details

Furthermore, a general problem of using vulnerability scanners on production systems is that the probes of the scanner can cause problems and damages on the systems, e.g., due to aggressive behavior of the probes.

7. OPEN QUESTIONS

Reducing Vulnerabilities.

Besides splitting security groups, it would also be interesting to split AMIs. In case an AMI contains multiple services with different severity ratings, the high severity services should be isolated from the other ones. This splitting of an AMI is more difficult to automate, since we have to understand the configuration of the service we want to isolate, in order to move it to a separate AMI. The open question is how to safely extract services from an operating system image without breaking the functionality of the service and potential dependencies of services hosted on the same image.

Security Audits for Private Clouds.

Our approach validates the configuration of Amazon's EC2. To actually guarantee security, it assumes that a specified configuration is correctly implemented by Amazon. A resulting question is how the analysis techniques presented for the Amazon cloud can be transferred to private clouds where information about the configuration are harder to extract. For example, can we use the concept of security groups also in the private cloud scenario, i.e., a set of VMs hosted on a physical server which is protected by a firewall can be in a "security group" depending on the firewall configuration. Understanding the relationship between the firewall rules and VMs is crucial for extracting security groups from a private cloud.

Furthermore, spawning instances of production VMs for security analysis purposes, in order that the original VM is not affected by the analysis, can be more difficult in private cloud environments where the management infrastructure could be less automated, not ready for programmatic use, and more heterogeneous.

AMI Security & Multi-tenancy.

Besides analyzing the AMIs regarding vulnerable services, we could also perform an analysis to check for cloud security best practices. For example to check that images do not contain hard-coded host keys, e.g., for ssh. Two instances of an AMI should respond with different host keys.

Furthermore, multi-tenancy and side-channel attacks are problematic in cloud computing environments. We could introduce a policy and validation method for allowing or disallowing co-residency on physical machines for VMs or security groups in general. For example members of a high-priority security group should not be located on the physical host as members of a low-priority one.

8. CONCLUSIONS AND OUTLOOK

In this article we presented a novel approach of assessing the security of multi-tier applications deployed in infrastructure clouds using Amazon EC2 as an example case. The security assessment consists of an analysis step of the system with regard to two properties: reachability and services vulnerabilities. For both cases, a query language and processor allow administrators to get an insight into the reachability and vulnerability of the services, e.g., in order to detect misconfigurations which expose the wrong services or to find vulnerable services. Furthermore, a policy language allows the specification of the desired state of the system which can be periodically verified.

We implemented the security assessment in Python and evaluated it against a sample multi-tier application on Amazon EC2. Violations in the vulnerability policies were successfully detected, and possible attack paths were presented to the administrator in order to enable him to secure the involved services.

Finally, we discussed open questions and provided an outlook of future research directions. For example, we pointed out that it would be interesting to extend the process of splitting security groups to AMIs in order to reduce vulnerability risks. Furthermore, we discussed to extend the security assessment to cover public infrastructure clouds, and to handle multi-tenancy and AMI security issues in public cloud environments.

In conclusion, we can say that our security assessment will be a valuable tool for administrators of multi-tier applications deployed in infrastructure clouds like Amazon EC2. Troubleshooting using the query languages can be done and the policy language allows a specification of the desired state, in order that the deployment remains secure. Our practical evaluation demonstrates the feasibility of this approach.

Acknowledgment

This project was partially supported by the MASTER research project funded by the European Commission's FP7 programme.

9. REFERENCES

[1] Aciiçmez, O. Yet another microarchitectural attack: exploiting i-cache. In *CSAW '07: Proceedings of the 2007 ACM workshop on Computer security architecture* (New York, NY, USA, 2007), ACM, pp. 11–18.

[2] Aciiçmez, O., Koç, c. K., and Seifert, J.-P. On the power of simple branch prediction analysis. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security* (New York, NY, USA, 2007), ACM, pp. 312–320.

[3] Amazon. The Amazon Elastic Compute Cloud (EC2). Available at `http://aws.amazon.com/ec2/`, last accessed March 2010, 2010.

[4] Amazon Web Services. Amazon Web Services: Overview of Security Processes, November 2009.

[5] Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., and Yilek, S. Hedged public-key encryption: How to protect against bad randomness. In *ASIACRYPT* (2009), pp. 232–249.

[6] BOTO. boto - Python interface to Amazon Web Services. Available at `http://code.google.com/p/boto/`, last accessed June 2010, 2010.

[7] CHRISTODORESCU, M., SAILER, R., SCHALES, D. L., SGANDURRA, D., AND ZAMBONI, D. Cloud Security Is Not (Just) Virtualization Security. In *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security* (New York, NY, USA, 2009), ACM, pp. 97–102.

[8] COHEN, R. Announcing Enomaly ECP High Assurance Edition for Trusted Cloud Computing. Available at `http://www.elasticvapor.com/2010/04/announcing-enomaly-ecp-high-assurance.html`, last accessed June 2010, 2010.

[9] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: A Virtual Machine-Based Platform for Trusted Computing. *SIGOPS Oper. Syst. Rev. 37*, 5 (2003), 193–206.

[10] GOODALL, J. R. Introduction to Visualization for Computer Security. In *VizSEC* (2007), pp. 1–17.

[11] JAJODIA, S., LIU, P., SWARUP, V., AND WANG, C. *Cyber Situational Awareness: Issues and Research.* Springer, 2009, ch. Topological Vulnerability Analysis, pp. 139–154.

[12] JAJODIA, S., AND NOEL, S. *Algorithms, Architectures, and Information Systems Security.* World Scientific Press, 2007, ch. Topological Vulnerability Analysis: A Powerful New Approach for Network Attack Prevention, Detection, and Response.

[13] KRAUTHEIM, F. J. Private Virtual Infrastructure for Cloud Computing. In *HotCloud '09: Workshop on Hot Topics in Cloud Computing* (2009), USENIX.

[14] LIPPMANN, R. P., AND INGOLS, K. W. An Annotated Review of Past Papers on Attack Graphs, 2005.

[15] MAO, W., MARTIN, A., JIN, H., AND ZHANG, H. Innovations for grid security from trusted computing. In *Fourteenth International Workshop on Security Protocols* (2006), LNCS, Springer-Verlag.

[16] MELL, P., AND GRANCE, T. Effectively and Securely Using the Cloud Computing Paradigm, October 2009.

[17] MELL, P., SCARFONE, K., AND ROMANOSKY, S. A Complete Guide to the Common Vulnerability Scoring System Version 2.0. Available at `http://www.first.org/cvss/cvss-guide.html`, last accessed June 2010, June 2007.

[18] NETWORKX DEVELOPERS. NetworkX. Available at `http://networkx.lanl.gov/`, last accessed June 2010, 2010.

[19] NOEL, S., ELDER, M., JAJODIA, S., KALAPA, P., O'HARE, S., AND PROLE, K. Advances in Topological Vulnerability Analysis. In *CATCH '09: Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 124–129.

[20] OPENVAS. OpenVAS, Open Vulnerability Assessment System. Available at `http://www.openvas.org`, last accessed May 2010, 2010.

[21] PERCIVAL, C. Cache missing for fun and profit, May 2005.

[22] PHILLIPS, C., AND SWILER, L. P. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms* (New York, NY, USA, 1998), ACM, pp. 71–79.

[23] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), ACM, pp. 199–212.

[24] RISTENPART, T., AND YILEK, S. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Proceedings of Network and Distributed Security Symposium – NDSS '10* (2010).

[25] SANTOS, N., GUMMADI, K. P., AND RODRIGUES, R. Towards Trusted Cloud Computing. In *HotCloud '09: Workshop on Hot Topics in Cloud Computing* (2009), USENIX.

[26] SHEYNER, O., HAINES, J., JHA, S., LIPPMANN, R., AND WING, J. M. Automated Generation and Analysis of Attack Graphs. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2002), IEEE Computer Society, p. 273.

[27] SHEYNER, O., AND WING, J. Tools for generating and analyzing attack graphs. In *Proceedings of formal methods for components and objects* (2004), LNCS, pp. 344–371.

[28] SWILER, L., PHILLIPS, C., ELLIS, D., AND CHAKERIAN, S. Computer-attack graph generation tool. In *DARPA Information Survivability Conference Exposition II, 2001. DISCEX '01. Proceedings* (2001), vol. 2, pp. 307 –321 vol.2.

[29] SWILER, L. P., PHILLIPS, C., AND GAYLOR, T. A Graph-Based Network-Vulnerability Analysis System. In *Sandia National Laboratories, Albuquerque,New* (1997), ACM Press, pp. 97–3010.

[30] TENABLE NETWORK SECURITY. Nessus, the Network Vulnerability Scanner. Available at `http://www.nessus.org`, last accessed March 2010, 2010.

[31] TRAN, T., AL-SHAER, E., AND BOUTABA, R. PolicyVis: Firewall Security Policy Visualization and Inspection. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–16.

[32] WILLIAMS, L., LIPPMANN, R., AND INGOLS, K. An Interactive Attack Graph Cascade and Reachability Display. In *VizSEC* (2007), pp. 221–236.

[33] WILLIAMS, L., LIPPMANN, R., AND INGOLS, K. GARNET: A Graphical Attack Graph and Reachability Network Evaluation Tool. In *VizSec '08: Proceedings of the 5th international workshop on Visualization for Computer Security* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 44–59.

[34] WOOL, A. A Quantitative Study of Firewall Configuration Errors. *Computer 37*, 6 (2004), 62–67.