An Approach for the Automated Analysis of Network Access Controls in Cloud Computing Infrastructures

Thibaut Probst, Eric Alata, Mohamed Kaâniche, Vincent Nicomette

An Automated Approach for the Analysis of Network Access Controls in Cloud Computing Infrastructures

T. Probst[1], E. Alata[1], M. Kaâniche[1], V. Nicomette[1]

[1]CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
{probst,ealata,kaaniche,nicomett}@laas.fr

Abstract. This paper describes an approach for automated security analysis of network access controls in operational Infrastructure as a Service (IaaS) cloud computing environments. Our objective is to provide automated and experimental methods to analyze firewall access control mechanisms aiming at protecting cloud architectures. In order to determine the accessibilities in virtual infrastructure networks and detect unforeseen misconfigurations, we present an approach combining static and dynamic analyses, along with the analysis of discrepancies in the compared results. Our approach is sustained by experiments carried out on a VMware-based cloud platform.

Keywords: security, accessibility analysis, cloud computing, firewall, network.

1 Introduction

Cloud computing is an emerging paradigm which allows the easy hosting and management of infrastructures, platforms and applications, while reducing deployment and operation costs. Providers propose to clients different kinds of resources as services. To satisfy these needs, various technologies like virtualization, new networking concepts, Web services, are mixed in complex architectures. This complexity, along with the presence of many different actors (service providers and consumers, developers, vendors, brokers, etc.) that cannot trust each other, make such environments vulnerable to many security threats [1–3] and raise security concerns for the clients and the providers. To cope with these threats, various security mechanisms can be deployed in the cloud, including firewalls or Identity Access Management (IAM) tools, and Intrusion Detection and Prevention Systems (IDS/IPS). The first category aims to implement network access controls, while the second one aims to detect (and possibly block) attacks in a network or on a host. Cloud environments constantly evolve over time as clients can add or remove instances and users or modify configurations, which could have impacts on the cloud security. Therefore, it is important for the client and the provider to monitor and analyze at a regular basis the security level of cloud infrastructures, in order to adapt and improve the configuration

of the security tools. In this paper, we focus on analyzing cloud computing firewalls access controls, because they have a direct impact on network accessibility (or reachability) in virtual infrastructures. An accessibility is the first support of attack vectors, hence finding accessibilities is the first step in building attack scenarios and assessing the efficiency of security mechanisms. Various types of firewalls can be deployed in the cloud either in the client's network topology (and thus configured by the client), or at the hypervisor level (and thus configured by the cloud administrator). As a consequence, controls on cloud firewalls are often balanced between the clients and the provider.

The configuration of such firewalls is generally tedious and error prone due to the increasing complexity of virtualized infrastructures. Therefore, efficient methods are needed to analyze network accessibilities in an operational context and identify potential discrepancies with those defined and desired by the clients. Two methods can be used for this purpose: *1*) static analysis of devices configuration; *2*) dynamic analysis by sending traffic over the network.. In traditional network environments, static analysis is generally preferred because dynamic analysis is more intrusive and hardly doable on production environments. However, actual static analysis tools are often not designed to support the analysis of end-to-end accessibilities. Also, such tools may fail to reveal all possible network accessibilities, in particular when hidden and implicit access control rules are enforced at different layers of the virtualized infrastructures. We argue that such access control rules could be revealed by dynamic analysis approaches that could complement static accessibility analysis techniques. Therefore, our research is aimed at providing automated ways (both static and dynamic) to determine network accessibilities in a client's infrastructure and look for potential discrepancies in the results. We provide the following contributions:

- A static analysis method of cloud components configurations.
- A dynamic analysis method of cloud components.

As an outcome, network accessibilities are provided for both methods, along with an analysis of the discrepancies in firewall access controls by comparing results along with the client's network security policy that we consider provided by the client[1]. Our approach is aimed at taking into account cloud security constraints by protecting the virtual infrastructures during the audits. This is achieved by running the static and dynamic accessibility analysis on a clone of the cloud infrastructure. Moreover, we leverage cloud advantages to perform quicker and deeper audits and we make the process fully automated.

The rest of the paper is organised as follows. Section 2 introduces our approach. Section 3 describes the main phase of our approach about analysis of network access controls. Section 4 presents a VMware-based testbed environment to validate our approach, along with the experimental results. Section 5 discusses related work. Finally, conclusion and future work are provided.

[1] The definition and retrieval of a cloud security policy, along with its implementation as filtering rules, are out of the scope of our contributions.

2 Overview of the Approach: Assumptions and Principles

In this section, we recap the main assumptions we consider in our approach, and then we explain its principles.

Main assumptions Our approach focuses on the virtual infrastructure level, that is why the considered cloud service model is IaaS. A virtual infrastructure is defined as a set of virtual datacenters (vDC), where a vDC includes virtual machines (VMs), networks, firewalls and storage. We assume thate the firewalls apply stateful packet inspection, and we consider two different types of virtual firewall commonly found in the cloud:

- Edge firewall: gateway for client's networks that routes, filters and translates inbound or outbound traffic. It is generally controlled by the client.
- Hypervisor-based firewall: introspects the traffic sent and received by VMs disregarding the topology. It can act on different scopes, which allows the creation of rules inside each client's network and applied to traffic to and from VMs of this network. There is a set of rules for each scope. This kind of firewall is generally managed by the provider, though some cloud portals can give the client access to certain scopes.

The security analysis should not disturb the client's business. It is intended to be fully automated (it does not need human intervention). The security analysis can be performed on demand (audits can be run on the client's will). Provided results correspond to the state of the system at the time when the analysis is run. These reports should be relevant to the actors (clients, provider) that control the analyzed tools. The accessibility analysis operations are run on behalf of the provider, and therefore using administrator rights on the cloud components.

Principles Our two-phase approach, illustrated in Figure 1, allows the automated analysis of network access controls in a client's virtual infrastructure deployed in a cloud. The first phase prepares the infrastructure to be analyzed by retrieving essential information from the infrastructure and cloning it, so the following steps can be done properly. It can be summarized as follows:

1. Fetch the configuration from the client's vDCs: users' privileges, IP addressing, network connections, etc.

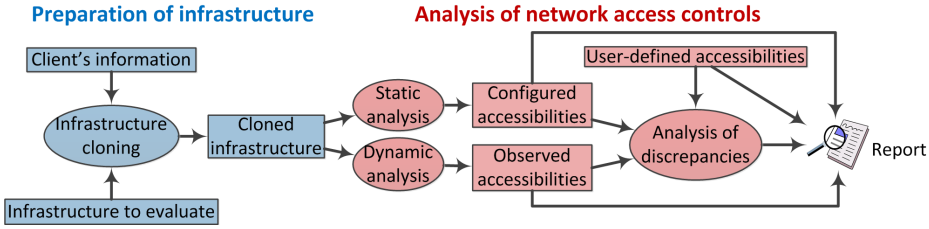

Fig. 1. Overall process

2. Create new vDCs from the configuration information.
3. Copy VMs, networks, firewalls, to the newly created datacenters.
4. Reset cloned VMs administrator password by using single-user mode. This avoids to ask the client any password for further actions on the cloned VMs.

The second phase analyzes access controls statically and dynamically to generate a security analysis report containing the accessibilities found by both methods. These are also compared to the user-defined accessibilities to perform an analysis of discrepancies in the results. This second phase is the core of our contributions and is developed in the next sections. During the audit process, we do not need to know any supplementary information, beyond what is supplied in a traditional cloud subscription process. All the other information we need are automatically retrieved using APIs provided by the cloud infrastructure and without human intervention. We take advantage of cloud computing embedded technologies to run audit operations described later on (cloning infrastructures, deploying and executing programs...).

3 Analysis of Firewall Access Controls

This section presents how we analyze network access controls in a virtual infrastructure managed by virtual firewalls, in order to provide accessibilities statically and dynamically (in addition to the user-defined one) and look for potential discrepancies in those results. An accessibility is defined as an authorized service from a source to a destination. The set of accessibilities is modeled in an accessibility matrix. The 1^{st} dimension of the matrix references the source VMs (rather than an IP address, because when sending traffic, one does not necessarily know what would be the source address if the machine has several interfaces) or a machine external to the client's networks, and the 2^{nd} dimension references the destination IP addresses. Table 1 gives an example of such a matrix. As mentioned previously, there are two main ways to conduct an accessibility analysis: statically or dynamically. We do it both ways: by extracting and analyzing information from the cloned cloud components configuration (this method falls into the static analysis category); by performing experiments: sending traffic such as network sweeps in order to verify the effective possible communications (this method falls into the dynamic analysis category). We define a discrepancy as an accessibility that has not been noticed in the matrix of configured accessibilities (generated from static analysis), in the matrix of observed accessibilities (generated from dynamic analysis), and in the user-defined accessibility matrix.

		Destinations			
		IP A1	**IP B1**	**IP B2**	**IP C1**
Sources	**VM A**				Service W
	VM B	Service X			Service Y
	VM C	Service Z			

Table 1. Example of accessibility matrix

3.1 Static Analysis

To build the accessibility matrix from static analysis, we need to deduce from the cloud configuration all the authorized communications on well-known services from: 1) VMs to IP addresses; 2) VMs to an external location (out of client's infrastructure, and represented as 0.0.0.0); 3) an external location to IP addresses. Deduced accessibilities are modeled as predicates in the form of:
$accessibility(X, SPROTO, SPORT, Y, DPROTO, DPORT)$: there is an accessibility from X, on source protocol $SPROTO$ and port $SPORT$, to Y, on destination protocol $DPROTO$ and port $DPORT$.
To generate these end-to-end accessibility predicates, we need to take into account the network topology and the routing/filtering/NAT rules applied to packets. Indeed, we have to care about the interactions of rules within a firewall (for example, a rule can cancel the action of another one) and accross the topology (to memorize the actions done on packets). Furthermore, cloud computing networking and security rules use concepts like grouping (of addresses or objects), as well as service (protocol and port/subprotocol) definitions that need to be taken into account when designing static rule analysis tools. We also modeled two kinds of cloud firewalls: edge and hypervisor-based firewalls (cf. section 2).

The static analysis tool we designed is composed of two main modules: a configuration parser and a logic engine. The configuration parser extracts information from each cloud component configuration and translates them into predicates, and the logic engine uses these predicates in logic rules to generate accessibility predicates. That is why we chose the Prolog language to develop a scalable and efficient logic engine able to quickly and coherently deduce the accessibilities.

Configuration Parser The configuration parser is used to retrieve the information (in a specific format, which is mostly XML) from components configuration, so it is vendor-specific. Then it transforms this information into Prolog predicates understood by the logic engine. Here are a few examples of Prolog predicates generated by the configuration parser:

- $vm(X)$: X is a VM.
- $edge\_gateway(X)$: X is an edge firewall.
- $introspect\_network(N)$: N is a network scope of a hypervisor-based firewall.
- $network(X, N)$: X is part of network N.
- $route(X, Y, G)$: X has route to Y with G as next hop.
- $service(PROTO, PORT, S)$: S is a service composed of $PROTO$ and $PORT$.
- $snat(W, X, SPROTO, SPORT, T, TPROTO, TPORT)$: W translates X, source protocol $SPROTO$, source port $SPORT$ into $T, TPROTO, TPORT$.
- $allow/block(W, N, X, SPROTO, SPORT, Y, S)$: W has a filtering rule of order N that allows/blocks traffic from source X, on protocol $SPROTO$ and port $SPORT$ to destination Y, on service/service group S.

We developed a configuration parser for VMware vCloud and Linux-based VMs, which corresponds to our targeted environment, as detailed in section 4. We use a

provided REST API and Shell scripts to retrieve XML files from VMs, firewalls, and cloud management modules. Then, we run XSLT processing in conjunction with Python to parse and transform XML into Prolog logic predicates. We designed a XSLT sheet for each configuration we need to parse[2].

Logic Engine The logic engine runs an internal logic (set of Prolog rules using the previously presented predicates) upon the submission of an accessibility request. Accessibility requests aim to generate the accessibility predicates, as stated earlier. Let us consider the following requests:

- $accessibility(vm\text{-}372, tcp, any, \text{'172.16.2.66'}, tcp, DPORT)$.
- $accessibility(\text{'0.0.0.0'}, icmp, any, \text{'192.168.1.150'}, icmp, DPORT)$.

The logic engine is asked to return the accessibility predicates associated to all open TCP ports from vm-372 to 172.16.2.66, and then all the open ICMP subprotocols from 0.0.0.0 (external networks) to 192.168.1.150.

To process such accessibility requests, we designed an algorithm based on a set of Prolog rules, as shown on Figure 2. Starting from the source, the algorithm checks routing, NAT and filtering rules (using the initial parameters) on each edge firewall node of the route from the source to the destination. In our model, an edge firewall can also act as a simple router that allows all the traffic. Then, it checks routing from the destination to the source, but it does not verify filtering there because we consider stateful inspection firewalls. Eventually, it checks filtering at the hypervisor-based firewall, if present, for the appropriate scopes (source and destination network scopes, or the global scope by default).

The algorithm execution tune depends on the number of accessibility requests to execute, because each request may process thousands of Prolog rules. For example, to check whether a traffic is allowed on a firewall, the algorithm first looks for a rule which allows this traffic, where the source and destination can take several values: an IP address, an IP address range, a group, a network, a list of IP addresses, keywords like "any", "internal" and "external", etc. The "any" keyword can also be found for destination protocol and port values. Then, we verify that there is no potential denying rule with a smaller order (to take into account the cancellation of an allowing rule) than the previously found accepting rule. All these possible combinations lead to an exponential computational complexity (and can make accessibility requests take too long, which is not acceptable in a cloud audit process). To reduce the execution time, a maximum of rules are precompiled (once and for all) to generate all the associated predicates (for example all the allowed traffic on every firewall, i.e. traffic from/to each VM on every destination protocol and port combinations). Also, the jobs that perform the accessibility requests are multithreaded.

3.2 Dynamic Analysis

In dynamic analysis, real network packets are sent to determine the open ports on some targets. Generally, a port scanner is used from a remote machine controlled

[2] We chose not to provide details on XSLT sheets as they are implementation details.

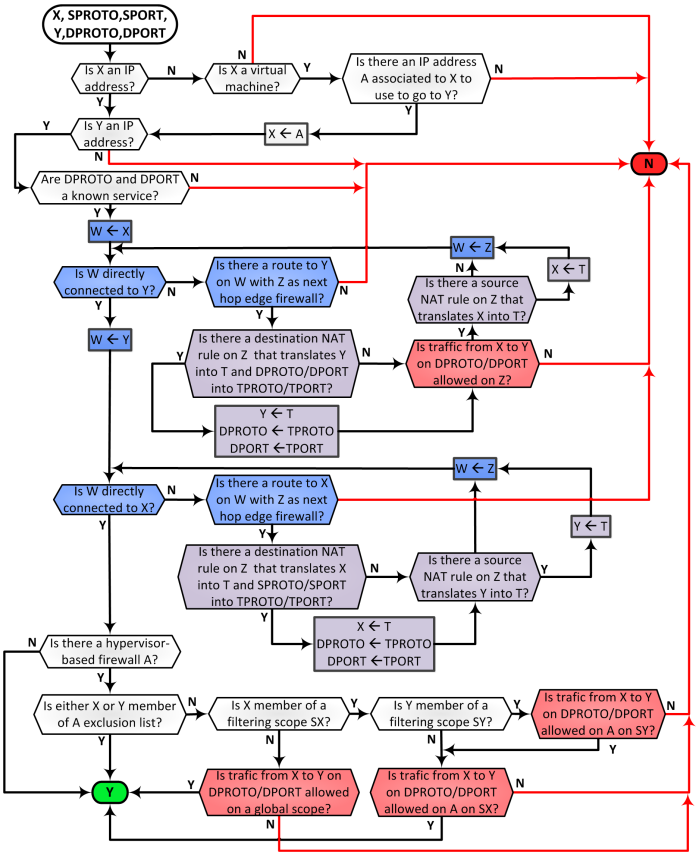

Fig. 2. Accessibility request processing algorithm

by the auditor. Common concerns of this method include the completeness of the results: the remote machine has to be set in all the possible network segments to determine all the possible accessibilities. Furthermore, this one-sided method relies on the ability to interpret network responses in order to determine accessibilities, and this is hard to do in connectionless traffic like UDP.

We are able to address these concerns, because we can easily control all the machines part of the analysis and monitor the traffic effectively received by the destinations. The proposed algorithm (Algorithm 1), is based on the elaboration of pair-to-pair sessions, testing all the possible combinations amongst all the VMs, and from VMs to the external location and vice-versa. A session comprises a server and a client. The client sends TCP, UDP and ICMP packets to the server IP address (if it has several IP addresses, then other sessions will be used) on well-known ports or subprotocols with a specific 4-octet payload we set in the packets. The server listens for all incoming packets and applies two filters: one to detect the specific payload; one to only capture packets addressed to it. Indeed, a server machine could be client of another session. However, it cannot

be server twice at a time, which allows at most a total number of concurrent running servers equal to the number of VMs. Note that we set up a 2-second sleeping time between the launch of the server and the client, to make sure that the server is ready to receive packets. The client and server programs were developed using Python sockets and Scapy API. They are automatically deployed on all the VMs prior to the execution of the algorithm. We use a three-dimension session array (1^{st} dimension: clients; 2^{nd} dimension: servers; 3^{rd} dimension: servers IP addresses) to report the done and undone sessions throughout the iterations of the main algorithm. All the values of this array are initialized to false, except when the server and the client are the same (we do not perform local sessions). At the end of each iteration of the algorithm, the clients are monitored to know whether their sending of packets is done so the servers can be interrupted and the results retrieved from them. The algorithm starts by running the VMs to VMs sessions, then the external location to VMs sessions (multithreaded, because the external location can be client of all VMs at the same time), and finally the VMs to the external location sessions sequentially. Let us note:

- NV as the total number of VMs.
- NIP_i as the total number of IP addresses of VM i.
- $NS = \sum_{0<i<NV} (1 + \sum_{\substack{0<j<NV \\ i \neq j}} NIP_j) + \sum_{0<i<NV} NIP_i$ as the total number of sessions. It is composed of the sessions from VMs to other VMs and the external location, and sessions from the external location to VMs.
- $NS_i = NIP_i \times (NV - 1)$ the number of inter-VM sessions for a server i.
- d as the delay between packet sendings.
- NP as the number of packets to be sent in each session.

An iteration is defined as the time needed to execute a session, which is $NP \times d + 2$ (2 is the 2-second sleeping time between the launch of a server and a client). Remember that several sessions can be executed in a single iteration. When VMs have one IP address, $NS_i = (NV - 1)$, and this is also the number of iterations for inter-VM exchanges (one session as a server per iteration for each VM). However, VMs can have multiple IP addresses (known as multihoming), and a VM cannot be server twice during the same iteration. The number of extra inter-VM sessions related to a server i due to multihoming is noted as $NA_i = NS_i - (NV - 1)$. Multihoming entails $\max\{NA_i\}$ additional iterations (where the extra sessions are executed in parallel according to the algorithm). Therefore, we can deduce the number of iterations for inter-VM exchanges as $NV - 1 + \max\{NA_i\}$. Sessions from the external location to VMs IP address are run in parallel ($\max\{NIP_i\}$ iterations), and sessions from VMs to an external location are run sequentially (NV iterations). Adding the 2-second sleeping time present in the *get_access*() function (to ensure the results of a session are generated on a server before retrieving them), we get a global estimation time of:
$T = (2NV - 1 + \max\{NA_i\} + \max\{NIP_i\}) \times (NP \times d + 2) + NS \times 2$
In the formula, we do not take into account the time of code instructions and the time needed to retrieve the accessibilities and write them on disk. Our algorithm

Algorithm 1 Dynamic analysis algorithm

Require: V: set of VMs and their IP addresses; M: session matrix;
this: external audit machine; *this_ip*: external audit machine IP address
Ensure: AM: accessibility matrix

```
1: still_sessions ← True
2: while still_sessions do
3:    still_sessions ← False
4:    for n ∈ 1...Card(V) do
5:       session_found ← False; S ← ∅; vm1 ← 1
6:       while vm1 ≤ Card(V) and !session_found do
7:          vm2 ← 1
8:          while vm2 ≤ Card(V) and !session_found do
9:             ip ← 1
10:            while M[vm1][vm2][ip] do ip++ end while
11:            if !M[vm1][vm2][ip] and S ∩ {V[vm1],V[vm2][ip]} == ∅ then
12:               still_sessions ← True
13:               run_server(V[vm2]);sleep(2); run_client(V[vm1])
14:               S ← S ∪ {V[vm1],V[vm2],V[vm2][ip]}
15:               M[vm1][vm2][ip] ← True; session_found ← True
16:            end if
17:            vm2++
18:         end while
19:         vm1++
20:      end while
21:   end for
22:   for s ∈ S do AM ← get_access(s) end for
23: end while
24: S ← ∅;
25: for vm ∈ 1...Card(V) do
26:    for ip ∈ 1...Card(vm[ip]) do
27:       run_server(V[vm]); run_client(this); S ← S ∪ {this,V[vm],V[vm][ip]}
28:    end for
29: end for
30: for s ∈ S do AM ← get_access(s) end for
31: for vm ∈ 1...Card(V) do
32:    run_server(this); run_client(V[vm]); AM ← get_access({V[vm],this,this_ip})
33: end for
```

has a linear execution time which depends on the number of VMs and IPs per VM. A simple algorithm that would execute each session sequentially would have an exponential execution time depending on the number of sessions (of course the sleeping times we put would be needed and to be considered too).

4 Testbed and Experimental Results

Testbed Environment To validate the feasibility and efficiency of our algorithms, we run our experiments on a cloud platform based on VMware which

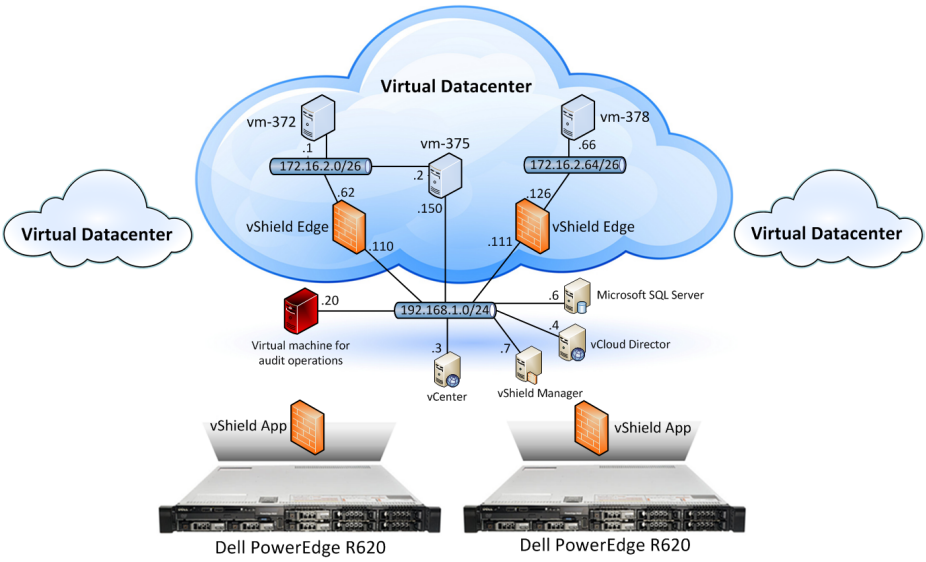

Fig. 3. VMware-based experimental platform: scenario 1

is widely used for the deployment of IaaS clouds. This platform includes two physical rack servers (Dell PowerEdge R620 with two Intel Xeon E5-2660 and 64Go of RAM) connected on a network switch (HP 5120-24G EI). The servers run VMware vCloud Suite (VMware top IaaS solution). It includes a hypervisor (VMware ESXi), cloud management software (vCenter, vCloud Director, Microsoft SQL Server), and cloud security management sofware (vShield Manager). We deployed two scenarios: one is a small size virtual infrastructure composed of one vDC to illustrate our approach; another one is a large size infrastructure composed of three vDCs to explore the scalability of our approach. Figure 3 illustrates the experimental platform with scenario 1 and its layer 3 topology. VMs run the Debian operating system. VMs and edge firewalls are connected using distributed virtual switches. The audit operations are executed from a VM equipped with 4 CPU cores and 4Go of RAM and running a Debian-based system with the necessary tools and libraries. It is placed in the cloud management network which is external to clients' networks. In scenario 1, there are 3 VMs (one IP address for two of them, two IP addresses for one of them). There are 8 possible inter-VM sessions. Adding the sessions related to the external location (the VM for audit operations), we have a total of 15 sessions. We also implemented a total of 30 routing, filtering and NAT rules on the firewalls in order to implement the accessibility matrix shown in Table 2. In scenario 2, there are 23 VMs (one IP address for 22 of them, two IP addresses for one of them). There are 528 inter-VM possible sessions. Adding the sessions related to the external location, we have a total of 575 sessions. We also implemented a total of 120 routing, filtering and NAT rules on the firewalls. Table 3 summarizes the two scenarios.

		Destinations				
		172.16.2.1	**172.16.2.2**	**192.168.1.150**	**172.16.2.66**	**0.0.0.0**
Sources	**vm-372**				echo-request	
	vm-375	MySQL			SSH	
	vm-378	echo-reply				
	0.0.0.0			any		

Table 2. User-defined accessibility matrix implemented in scenario 1

# of	vDCs	VMs	Sessions	Networks	Edge firewalls	Hypervisor-based firewall scopes	Routing, NAT, Filtering Rules
Scenario 1	1	3	15	3	2	3	30
Scenario 2	3	23	575	7	5	7	120

Table 3. Examples scenarios

Prior to the execution of the static and dynamic analysis, we used a set of embedded tools provided by VMware to perform the automated cloning phase of the infrastructure. Note that in both scenarios, we consider a total of 587 known TCP, UDP and ICMP services[3].

Static analysis In scenario 1, there are $587 \times 15 = 8805$ accessibility requests executed to build the accessibility matrix. Figure 4 is the accessibility graph associated to the generated matrix. It took 22.61s to process the accessibility requests. In scenario 2, there are $587 \times 575 = 337525$ accessibility requests, and it took 4mn24s to run the algorithm. This shows that our logic engine is scalable, where nearly 38 times more requests are only 12 times slower to execute.

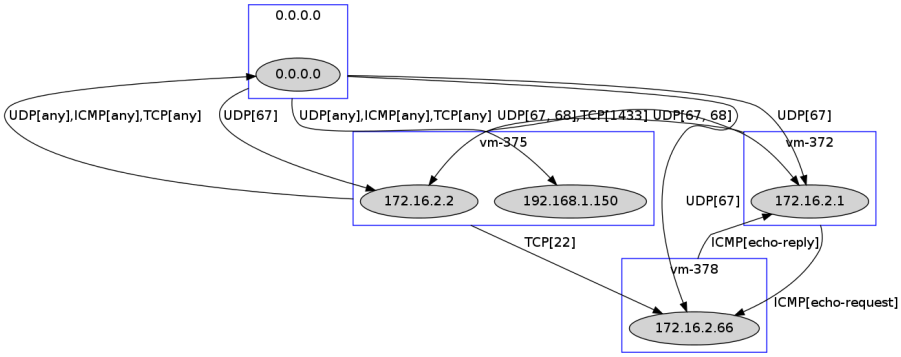

Fig. 4. Accessibility graph generated from static analysis in scenario 1

Dynamic analysis In scenario 1, using 2-second sleeping times and a 10ms inter-packet delay, the theoretical execution time of our alorithm would be (cf. section 3.2) $T = (6-1+2+2) \times (587 \times 0.01 + 2) + (15 \times 2) = 1\text{mn}41\text{s}$. Running the dynamic analysis on our testbed gave a real execution time of 2mn58s. The

[3] We used the services used in the DARPA Internet network, commonly found under the /etc/services file in Linux-based distributions.

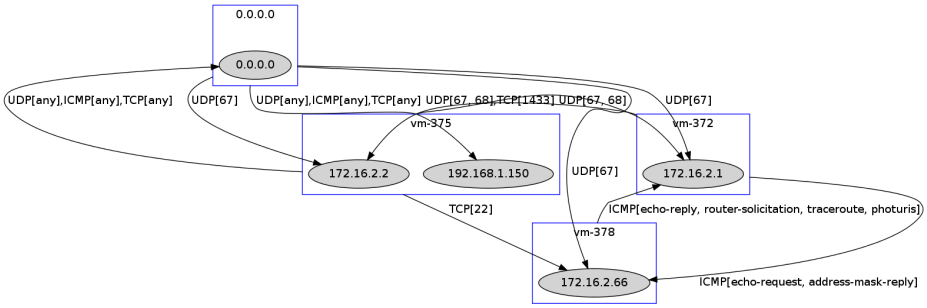

Fig. 5. Accessibility graph generated from dynamic analysis in scenario 1

overhead is explained by code instructions. Figure 5 is the accessibility graph associated to the generated matrix. In scenario 2, the duration was 47mn46s with a 1ms inter-packet delay and 0.5-second sleeping times, which is an acceptable result given the size of the infrastructure in this case, and the total number of packets sent (337 525 packets). Note that the inter-packet delay and the sleeping times parameters have to be adjusted according to the capacity of the physical hardware and the size of the infrastructure to analyze. It took 1h16mn08s using the parameters from scenario 1 (2-second sleeping times and a 10ms inter-packet delay), and 56mn47s with a 5ms inter-packet delay and 1-second sleeping times.

Discrepancy analysis Here are the discrepancies reported from scenario 1:

- 0.0.0.0 → 172.16.2.1, 172.16.2.2 (UDP 67): not defined but configured.
- 172.16.2.1 → 172.16.2.2 (UDP 67,68): not defined but configured.
- 172.16.2.2 → 172.16.2.1 (UDP 67,68): not defined but configured.
- 0.0.0.0 → 172.16.2.1, 172.16.2.2 (UDP 67): not defined but observed.
- 172.16.2.1 → 172.16.2.2 (UDP 67,68): not defined but observed.
- 172.16.2.2 → 172.16.2.1 (UDP 67,68): not defined but observed.
- vm-378 → 172.16.2.1 (router-solicitation, traceroute, photuris): not configured but observed.
- vm-372 → 172.16.2.66 (addr-mask-reply): not configured but observed.

These discrepancies can be explained because VMware firewalls let pass more traffic than expected in some rules (here some ICMP management subprotocols when allowing "echo-request" or "echo-reply" traffic) and implicitly configure some rules (here UDP port 67 and 68 traffic when activating DHCP features).

5 Related Work

Network accessibilities are built by discovering the hosts and analyzing connectivity between them. They could be derived either statically, based on network equipments configuration (analyzing routing, filtering and NAT rules); or dynamically, by running port scans. In [4], the authors rigorously formulate the

problem of reachability in networks. While this is useful as grounding work to understand well this problem, they do not provide methods to collect network information and their model does not handle complex NAT. Furthermore, they do not provide a practical algorithm or experimental results showing the scalability of their approach. The approach presented in [5], about computing accessibility matrices and expressing accessibility queries, is thorough and very relevant. Their data structures, algortihms and query language were a basis to build our static analysis approach, though we kept it simpler and more adapted to cloud networks. We also believe that using an imperative language (Prolog) rather than a declarative one (C++) is more adapted to compute complex accessibility queries. [6, 7] are good examples of analysis of filtering configurations but are restricted to a device scope. They do not compute end-to-end accessibilities which are needed in our context.

Considering the general topic of cloud security audits, one can mention the approaches presented in [8, 9], in addition to the publication of recommandations and guidance on security assessments by the Cloud Security Alliance (CSA) [10]. In [8], cloud infrastructures are analyzed using accessibility graphs and vulnerability discovery to build attack graphs and find shortest paths as critical attack scenarios. Although the proposed approach is judicious, their static analysis model includes only one global firewall and thus does not address rules consistency and interactions in complex network topologies. We can also cite the work in [11] on static flow analysis in virtualized infrastructures, where interaction of cloud ressources are generated as a graph model. However, they do not take into account filtering rules at the upper levels. Furthermore, both [8] and [11] do not propose a dynamic analysis method to verify network accessibilities. In [9], the authors provide an automated audit system as a service for cloud environments, along with a language to define a cloud security policy. The goal of this system is to allow automatic auditing of VMs following user-defined policies. The policy scenarios are quite thorough and tend to model the security requirements a cloud would have to meet, including network access controls. Although it can audit systems in real time, their solution requires embedding agents within each of the key points of the infrastructure, which is not feasible in proprietary clouds. Our approach is more lightweight as we preserve the original components and execute programs in the VMs only for the time of the analysis.

6 Conclusion and Perspectives

In this paper, we have described the basics of our approach to automatically analyze network access controls in cloud computing virtual infrastructures. It aims at identifying accessibilities managed by virtual firewalls, considering a combination of static and dynamic analysis methods to derive accessibilities, along with the analysis of discrepancies in the results. The proposed methodology was designed to take into account the constraints inherent to cloud computing with limited impact on the provider and client's business. Experiments have been carried out on a VMware-based cloud platform to illustrate the feasibil-

ity and scalability of our approach. The developed tools shall be integrated in an industrial secured cloud computing framework. Having designed a generic approach, we can plan to extend our VMware-based prototype to other IaaS solutions. This would result in the development of adapted configuration parsers and XSLT sheets, and customize the use of provided APIs to manipulate the ressources.

Ongoing work includes the extension of the evaluation of cloud security tools to IDS/IPS mechanisms. We are currently exploring the construction and execution of attack scenarios from the accessibilities found, in order to assess the efficiency of deployed IPS/IDS probes [12]. We intend to keep the evaluation process fully automated and without any impact on the client's business.

References

1. Jensen, M., Schwenk, J., Gruschka, N., Iacono, L.L.: On technical security issues in cloud computing. In: Cloud Computing, 2009. CLOUD'09. IEEE International Conference on, IEEE (2009) 109–116
2. Studnia, I., Alata, E., Deswarte, Y., Kaâniche, M., Nicomette, V., et al.: Survey of security problems in cloud computing virtual machines. Proceedings of Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012) (2012) 61–74
3. Oktay, Sahingoz: Attack types and intrusion detection systems in cloud computing. In: Proceedings of the 6th International Information Security & Cryptology Conference. (2013) 71–76
4. Xie, G.G., Zhan, J., Maltz, D.A., Zhang, H., Greenberg, A., Hjalmtysson, G., Rexford, J.: On static reachability analysis of ip networks. In: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE. Volume 3., IEEE (2005) 2170–2183
5. Khakpour, A., Liu, A.X.: Quarnet: A tool for quantifying static network reachability. Michigan State University, East Lansing, Michigan, Tech. Rep. MSU-CSE-09-2 (2009)
6. Marmorstein, R., Kearns, P.: A tool for automated iptables firewall analysis. In Association, U., ed.: ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference. (2005) 44–44
7. Nelson, T., Barratt, C., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: The margrave tool for firewall analysis. In: USENIX Large Installation System Administration Conference. (2010)
8. Bleikertz, S.: Automated security analysis of infrastructure clouds. Master's thesis, Norwegian University of Science and Technologys (2010)
9. Doelitzscher, F., Ruebsamen, T., Karbe, T., Knahl, M., Reich, C., Clarke, N.: Sun behind clouds-on automatic cloud security audits and a cloud audit policy language. International Journal On Advances in Networks and Services **6**(1 and 2) (2013) 1–16
10. Alliance, C.S.: Secaas implementation guidance: Security assessments. (2012)
11. Bleikertz, S., Groß, T., Schunter, M., Eriksson, K.: Automated information flow analysis of virtualized infrastructures. In: Computer Security–ESORICS 2011. Springer (2011) 392–415
12. Probst, T., Alata, E., Kaaniche, M., Nicomette, V., Deswarte, Y.: An approach for security evaluation and analysis in cloud computing. SAFECOMP 2013 FastAbstract (2013)