Monitoring the Update Time of Virtual Firewalls in the Cloud

Hyunwook Baek, Eric Eide, Robert Ricci and Jacobus Van der Merwe

UUCS-18-005

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

May 16, 2018

Abstract

Since cloud users do not have direct visibility into the cloud provider's infrastructure, cloud users generally depend on the information provided by the cloud providers when they need to know about states of their virtual resources. In this document, we introduce an approach to monitor the update time of infrastructure level virtual firewalls (called Security Group) from the cloud user's side and technical details for practical deployment of this approach in an OpenStack environment.

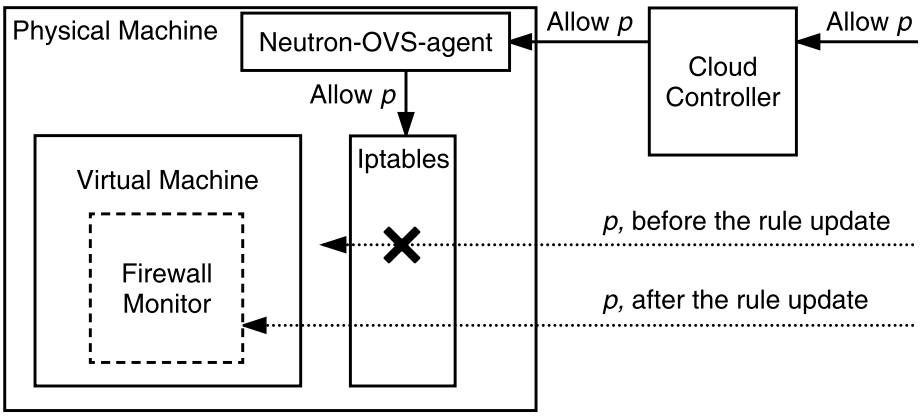

Figure 1: Monitoring update times of a virtual firewall

1 Introduction

In a cloud environment, cloud users can simply operate their virtualized data centers on the neat abstract layer; all complicated infrastructure level details are handled by the provider under the hood. However, such 'too good' abstraction also limits the visibility of the cloud users into their own resources. For instance, even if a cloud user needs to know about detailed states of his resources, there would be simply no way for him to obtain such information if the cloud provider does not explicitly provide it. In addition, for security and privacy, it could be undesirable for the cloud providers to reveal such additional infrastructure level information to the users.

In this context, we introduce an approach for cloud users to monitor their virtual resources. Especially, this document aims to provide technical details about a mechanism to monitor *the update time of infrastructure level virtual firewalls* (called Security Group) from a cloud user's virtual machine (VM). We describe the approach based on an OpenStack environment where virtual firewalls are implemented using Linux Iptables and Connection Tracking System [4]. Similar approaches can be applied to other IaaS cloud platforms.

2 Mechanism

Figure 1 illustrates a basic architecture for monitoring update times of virtual firewalls. When a cloud controller receives a request to add a firewall rule allowing a probing packet p, it forwards the request to the local agent, the agent lets the iptables add the rule correspondingly,

and the rule is finally added to the iptables (at time *t*). Meanwhile, a series of probing packets *p* is being sent to the monitoring VM and, naturally, only the probing packets arrived at the iptables after *t* can be successfully sent to the VM. Therefore, the VM can estimate the update time of the iptables from the arrival times of probe packets.

In Section 3, we describe different strategies to deploy this technique under various environmental conditions. In Section 4, we describe practical network protocols and firewall rules that can be utilized for this technique.

3 Deployment Environment

3.1 With helper nodes

The simplest way to probe changes in iptables is to have another VM (say a helper node) send (or receive) the probe packets. For example, to probe an *egress* rule x_i, the VM sends probe packets to the helper node which replies once it receives any probe packet for x_i. The problem with this approach is it requires one more VM. Though this might be a trivial problem in cases where the attacker has its own dedicated resources, it can be a challenging problem in some other cases where the attacker uses a compromised node in an isolated environment.

3.2 Without helper nodes

In case there exist no available helper nodes, the VM should be able to send probe packets which can go through the firewall and come back to itself. We term packets with such property *'boomerang packets'*. To be more precise, we define a boomerang packet for a firewall rule x_i is a uniquely identifiable packet that (i) leaves from the source node and (ii) makes itself to come back to the source (iii) without bypassing[1] the firewall rule x_i. There are several mechanisms by which this can be accomplished, and different ones work in different network environments. In the following paragraphs, we discuss different environmental

[1] In a cloud environment, sometimes, a packet may legitimately pass through a virtual firewall by an implicit rule. Since this implicit rule is not visible to cloud users, the packet may seemingly bypass a virtual firewall from the cloud user's perspective. Because we describe this attack from the cloud user's perspective, we also say the packet *bypasses* the firewall

setups (from the simplest to the most complicated) and suitable mechanisms under each setup.

Multiple interfaces – Layer-2 boomerang: The simplest (and least common) environment is the case in which the VM has multiple virtual interfaces connected to the same virtual network (i.e., the same virtual switch). In this environment, the VM can generate a boomerang packet by simply setting the source layer-2 (MAC) address to one interface's MAC and the destination to the MAC address of the other interface. The switch will simply forward packets from one interface to the other.

Multiple interfaces – Layer-3 boomerang: If the two virtual interfaces are connected to different networks, but there is routing between those networks, the VM can apply a similar approach; instead of setting the MAC addresses, it can set the source and destination layer-3 (IP) addresses of the boomerang packet.

Single interface – Layer-2 boomerang: If the previously described conditions are not met, the attacker may try similar techniques with a single interface if the network environment satisfies some other conditions. For example, the layer-2 address manipulation technique may work if the connected virtual switch does not drop packets with the same source and destination MAC addresses. However, we have found this to be unlikely in most production clouds, since it is one of the most primitive features of layer-2 switch devices to maintain the MAC address table and forward/drop packets based on the table. According to our test results, this layer-2 boomerang packet is silently dropped by virtual switches in OpenStack Icehouse (released in 2014) and Mitaka (released in 2016).

Single interface – Layer-3 boomerang: As with the case of the layer-2 boomerang, if the gateway router is allowed to forward packets through the interface that the packets came from, the layer-3 address manipulation technique works with a single interface. To be specific, if the attacker VM send its gateway router a fabricated packet whose source and destination IP addresses are its own but with the destination MAC address of the gateway's, the gateway router will naturally forward the packet back to the destination, the VM.

In contrast to layer-2 switch devices, this 'U-turn' forwarding is commonplace for routers – in a network, even if the source node does not know the correct route to a destination, its gateway router is in charge of forwarding its packet through a proper route; therefore, most commercial routers are configured to forward a packet back through its ingress port in case its routing table is indicating the port. Though some routers may also send an ICMP redirect packet to the source node to 'recommend' it to use the better route, they still forward the original packets back to its destination.

There can of course be commercial routers which are manually configured to drop packets with the same source and destination IP addresses. However, we argue that it is unlikely to be found in our target environment because (1) in a current cloud environment, the virtual routers are typically less feature-rich than commercial routers, and (2) it is not a common practice for operators to block this type of boomerang packets if they are not generated aggressively to be suspected as DoS attacks. We have tested the behavior of virtual routers in two different versions of OpenStack, Icehouse (released in 2014) and Mitaka (released in 2016). In both versions, the virtual routers were not prevented from forwarding packets destined to the source, and they were also configured to send ICMP redirect packets. In addition, in both versions, there were no configurable options related to this issue not only for cloud users but also for cloud providers. In the following Section 4, we explain details about protocols and rules for the single-interface layer-3 boomerang scenario.

4 Protocols and Rules

When we make a firewall rule to allow a certain type of connections, it is natural to allow both request packets in one direction and its counterpart responses in the opposite direction. For instance, if you make a rule to allow SSH connections from your external terminal node (say A) to your VM in a cloud (say B), you must allow not only TCP traffic from A to B's port 22, but also TCP traffic from B's port 22 to A's port used for the connection. Then, should we always make a pair of (or more) rules to allow a type of connection? Fortunately, the answer is *No* since security group rules in the most of the cloud platforms are *stateful* [2, 1, 3].

In OpenStack, the statefulness of security group rules is enabled by Connection Tracking System (conntrack) [4]. Briefly speaking, conntrack is a module that estimates current states of network connections by inspecting the header of each packet. This connection state information can help iptables to dynamically filter the packets related to existing connections. In the previous SSH example, if we set a firewall rule for A to B's port 22 and if we send a SSH request from A's port 56789 to B's port 22, the conntrack will tell iptables that packets from B's port 22 to A's port 56789 is related to an existing connection, and then the iptables will implicitly let the egress packets pass through the firewall.

For monitoring infrastructure level activities through firewalls, a problem with conntrack is that it *may* or *may not* let our boomerang packets work: probe packets may be silently dropped at the firewall or *bypass* the firewall rules, depending on protocols, rules and system versions. Therefore, the attacker must be careful when they choose protocols and rules for the side channel. In this section, we introduce several representative protocols and rules for

Table 1: Protocols and Rules for probing

EB: the probe may bypass the egress-firewall even after the allowing rule is deleted, **IB:** the probe bypasses the ingress-firewall, **D:** the probe is silently dropped by firewall. For the case of **EB**, the attacker need to consider the connection time-out duration. For the case of **IB** and **D**, the probing mechanism will not work. In OpenStack Juno, a connection can be reused for probing a rule if the previous connection is timed out. In OpenStack Mitaka, a connection is terminated when the corresponding rule is deleted, so a connection can be reused regardless of the state of the previous connection.

Mechanism	Rule Direction	Juno	Mitaka	Note
ICMP Echo Rquest-Reply	Egress	EB	OK	Default connection time-out: 30 sec. No ingress rule is needed.
	Ingress	IB	IB	
Request-type ICMP Boomerang	Egress	EB	OK	Default connection time-out: 30 sec.
	Ingress	EB	OK	
Non-repqeust-type ICMP Boomerang	Egress	D	OK	Does not make any connection entry.
	Ingress	D	OK	
TCP SYN or ACK Boomerang	Egress	EB	OK	Default connection time-out: 120 sec (SYN), 300 sec (ACK).
	Ingress	EB	OK	
Other TCP Boomerang	Egress	D	OK	Does not make any connection entry.
	Ingress	D	OK	
UDP Boomerang (sport $\neq$ dport)	Egress	EB	OK	Default connection time-out: 30 sec.
	Ingress	EB	OK	
UDP Boomerang (sport = dport)	Egress	EB	OK	Default connection time-out: 120 sec. No ingress rule is needed.
	Ingress	IB	IB	
Boomerang using Other Protocols	Egress	EB	OK	Default connection time-out: 600 sec. No ingress rule is needed.
	Ingress	IB	IB	

generating boomerang packets.

4.1 ICMP Ping

Before we talk about the details of boomerang packets and conntrack, we first start with the simpler protocol – ICMP ping.

Assume an attacker's VM (say A) has a reachable node (say B) in the network, and the node B replies to ICMP echo requests. Under this setup, the attacker VM A can use ICMP echo request/reply as probe packets for a firewall rule:

```
Allow Egress ICMP type:8 code:0 dst:B_IP
```

To probe if this firewall rule is established, the attacker VM can start to ping the node B with the probe packets as follows:

```
srcIP:A_IP dstIP:B_IP proto:ICMP
type:8 code:0 id:123 seq:0-65535
```

If a probe packet successfully pass through the firewall rule, it will arrive at the node B, and the node will send back a corresponding echo reply packet with the same id and sequence number. This echo reply packet can also successfully pass through the firewall because the conntrack makes a special ingress rule for it when it first sees the counterpart echo request packet[2]. Therefore, the attacker node can estimate the time when the firewall rule is established by checking the departure time of the echo request packet which corresponds to the firstly arrived echo reply packet.

However, this approach may not work if the attacker node reuses the same rule. Once the conntrack observes a ping request packet, it creates a new connection entry for the ping based on the following five tuples of the packet – source IP, destination IP, ICMP type, ICMP code and ICMP ID. From this point, any packets that have the same tuples bypass the firewall until the connection entry expires. This means, in the previous example, except for the very first ping request packet, every ping request packets generated by the ping process will bypass the firewall, *even after the rule is deleted.*

For this reason, the attacker cannot simply use the vanilla ping program, which does not support an option to change ICMP code and ID values. Alternatively, if the attacker can directly generate ICMP echo request packets, the attacker can use ICMP rules and probe packets as follows:

```
<Rule x>
Allow Egress ICMP type:8 code:k
```

```
<Probes for the rule x>
srcIP:A_IP dstIP:B_IP proto:ICMP
type:8 code:k id:0-65535 seq:0-65535
```

[2]To be more precise, the iptables will check the echo reply packet against a special rule, and the rule will query the conntrack to determine the packet is related or belongs to any existing connections.

where $k \in \{0, \ldots, 255\}$. Here, there exist 256 of different ICMP echo request rules (differentiated by the code). Also, each rule has 65,536 of different matching connection entries (differentiated by the ID), each of which has 65,536 different uniquely identifiable packets (differentiated by the sequence number). This means, the attacker can reuse the same rule and still avoid egress-firewall bypassing by using different ID values. In practice, this number of rules is enough to continuously monitor iptables update events. For example, with default conntrack and OpenStack setup, the time-out duration of ICMP connection entry is 30 seconds[3] and OpenStack's iptables update period is 2 seconds.

The egress-firewall bypassing problem may or may not happen depending on the specific cloud platform and its configuration. For example, in earlier versions of OpenStack, we can observe the egress-firewall bypassing problem. Likewise, according to the Amazon AWS user guide [1], AWS security groups also have the same firewall-bypassing phenomenon[4]. However, in newer versions of OpenStack, the same problem does not occur because the connection entries in the conntrack are explicitly terminated when their corresponding firewall rules are deleted [3].

A limitation of this approach is that *we cannot use ingress rules for probing*; since ICMP request packets make corresponding reply packets bypass the firewall, ingress rules matching these ICMP echo reply packets can never be probed through this mechanism. This can be a serious limitation depending on the situation of the attacker VM, which we will discuss in Section 4.6.

4.2 ICMP Boomerang Packets

The aforementioned limitation of ICMP ping as a probing mechanism is fundamentally because the conntrack module expects the ingress probe packet to come when it first sees the egress probe packet. This means, if an ingress probe packet is seemingly unrelated to its egress counterpart, the pair of packets can be utilized for probing regardless of the behavior of the conntrack module.

Fortunately, we can make use of the previously introduced layer-3 boomerang mechanism to make probe packets with this property. For example, if we make a layer-3 boomerang

[3]The connection entry expires if no related packet comes for the time-out duration.

[4]Since cloud providers do not reveal their infrastructure level details, it is difficult to precisely understand internal connection tracking mechanism of each cloud provider. However, for Amazon AWS, the description about behavior of their security group's connection tracking mechanism is exactly the same as that of Linux conntrack-tool. Thus, we may guess that AWS utilizes similar iptables and connection tracking system to implement their security group system.

```
Chain neutron-openvswi-ic0795dc1-b (1 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere             state INVALID
RETURN     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
RETURN     udp  --  10.254.0.3           anywhere             udp spt:bootps dpt:bootpc
RETURN     tcp  --  anywhere             anywhere             tcp dpt:ssh
RETURN     icmp --  anywhere             anywhere
RETURN     tcp  --  anywhere             anywhere             tcp dpt:http
neutron-openvswi-sg-fallback  all  --  anywhere             anywhere
```

(a) Iptables chain of OpenStack Juno

```
Chain neutron-openvswi-i7994c564-d (1 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
RETURN     udp  --  10.254.0.2           anywhere             udp spt:bootps udp dpt:bootpc
RETURN     tcp  --  anywhere             anywhere             tcp dpt:ssh
DROP       all  --  anywhere             anywhere             state INVALID
neutron-openvswi-sg-fallback  all  --  anywhere             anywhere
```

(b) Iptables chain of OpenStack Mitaka

Figure 2: Snapshot of exemplary Iptables chains of different OpenStack. We can see the rule to drop INVALID packet is placed at the top of the iptables chain in OpenStack Juno, but at the bottom in OpenStack Mitaka

packet with ICMP echo request header as follows:

```
<Egress Probe Packet>
srcMAC:A_MAC dstMAC:GW_MAC
srcIP:A_IP dstIP:A_IP proto:ICMP
type:8 code:0 id:123 seq:355
```

(where `GW_MAC` refers to the MAC address of the gateway), then the gateway will forward this packet back to A after it changes MAC addresses as follows:

```
<Ingress Probe Packet>
srcMAC:GW_MAC dstMAC:A_MAC
srcIP:A_IP dstIP:A_IP proto:ICMP
type:8 code:0 id:123 seq:355
```

In contrast to the case of ICMP ping, this ingress probe packet does not bypass the ingress firewall because this does not match what the conntrack expects – conntrack waits for the corresponding ICMP echo reply. Therefore, as long as we do not send ICMP echo replies, the boomerang probe packet goes through the ingress-firewall and may or may not be filtered

depending on the status of the ingress firewall rule. This feature allows us to utilize both the ingress and egress rules for probing the iptables update time.

However, the result may vary depending on ICMP type. For request-type ICMP packets (i.e., `type` $\in \{8, 13, 15, 17\}$), conntrack initializes a new connection once it observes an egress packet and waits for the corresponding ingress packet as we described above. For other ICMP types, however, the probe packets are recognized as neither initializing a new connection nor related to any existing connection. Thus, conntrack marks these packets as INVALID. A problem here is that the measure against INVALID packets may vary depending on cloud platform. In older versions of OpenStack, every security group has an implicit rule with highest priority that drops any INVALID packets, and this rule has fundamentally prevented any INVALID packet to pass the firewall regardless of whether there exist a matching rule or not. Yet, this has been corrected in the newer version of OpenStack where other firewall rules have higher priority so that INVALID packets also pass the firewall if there exist a matching rule. Figure 2 shows snapshots of iptables in different OpenStack versions where the priority of the rule to drop INVALID packet is different. For this reason, non-request-type ICMP cannot be used for probe packets in older versions of OpenStack.

4.3 TCP Boomerang Packets

For TCP packets, since conntrack initializes a new connection only if it sees a TCP SYN or ACK packet, the attacker may generate TCP SYN and ACK boomerang packets in a similar manner to request-type ICMP boomerang packets. For example, for TCP rule:

```
<Rule x>
Allow Egress TCP dport:k dst:A_IP
```

the attacker make use of the following sets of probe packets:

```
<Probes for the rule x>
srcIP:A_IP dstIP:A_IP
proto:TCP flags:SYN or ACK
sport:0-65535 dport:k seq:0-4294967295
```

Other TCP packets (including TCP SYN/ACK) are treated as INVALID similar to non-request-type ICMP packets, so they can be used as boomerang packets for newer versions of

OpenStack.

Compared to ICMP, a benefit of the TCP boomerang mechanism is that it has a larger number of rules and connections available. Since TCP firewall rules are differentiated by the destination port number and connections are differentiated by the source and destination port numbers, there can be at most 65,536 different rules and each rule can have at most 65,536 different connections. However, in case of using a small number of rules and connections, the TCP boomerang mechanism can have a disadvantage because the time-out duration for TCP connection entries is generally longer than ICMP (120 seconds for TCP SYN and 300 seconds for TCP ACK).

4.4 UDP Boomerang Packets

The behavior of conntrack for UDP packets is very similar to that for request-type ICMP packets: conntrack initiates a new connection when it sees an UDP packet and waits for reply packets. Since UDP does not have any specific form of reply, the reply packets are simply the packets with reversed IP addresses and port numbers. Therefore, for UDP firewall rule:

```
<Rule x>
Allow Egress UDP dport:k dst:A_IP
```

the attacker can probe the iptables update time using probe packets as follows:

```
<Probes for the rule x>
srcIP:A_IP dstIP:A_IP proto:UDP
sport:0-65535 dport:k seq:0-4294967295
```

Since the ingress and egress probe packet have the same port numbers, conntrack does not treat the ingress probe packets as the reply for the egress probes, and naturally the ingress probe packets do not bypass the firewall. However, there is an exception: if the source port and the destination port of a boomerang packet are the same, the ingress probe packet will be recognized as a response to the egress probe packet by the conntrack. In this case, the ingress will bypass the firewall as the case of ICMP ping.

4.5 Other Protocols

The Linux conntrack-tool supports TCP, UDP and ICMP, and takes a *default behavior* for other protocols – initiate a connection once it sees a packet, and wait for reply packets from the opposite direction. Here, a connection is differentiated only by three tuples – protocol number and source/destination IP addresses. Therefore, if we create a boomerang packet with a random protocol number, the ingress probe packet is always recognized as a reply to the egress probe. This is almost identical to the case of ICMP ping and UDP with the same port numbers. However, for these protocols, the default connection time-out duration is much longer (600 seconds) and the numbers of available rules and connections are fewer (253 rules and 1 connection per rule), so these protocols are less desirable for use as boomerang packets.

Table 1 summarizes the properties of aforementioned probing mechanisms in two different OpenStack versions.

4.6 Ingress rules or Egress rules?

When a rule is updated in iptables, the corresponding probe packets start to pass the firewall and arrive back at the attacker VM. Therefore, we can measure the arrival time of the first-arrived ingress probe packet and estimate the time when the iptables is updated. At this point, there is a difference between ingress rules and egress rules. If an ingress rule is used, the arrival time of the first passed probe packet is immediate to the iptables update time, and the attacker VM can immediately notice the event. However, for egress rules, once the iptables is updated, the first passed probe packet must make a round trip at first before it arrives back to the attacker VM. In this case, the actual iptables update time will be close to the departure time the probe packet so the VM should also search for the departure time of the matching probe packet. Naturally, there will be longer delay between the notification time and the actual event time in case of utilizing egress rules. Therefore, in cases when the attacker needs immediate notification of the iptables update event , utilizing ingress rules is preferred. In addition, since it is commonplace for cloud users to set an egress rule that allows any traffic, utilizing egress rules may not be an available option in all cases.

However, there is one important benefit of utilizing egress rules. Since egress rules drops probe packets just after the probe packets depart, most of the probe packets will not go through the cloud provider's network and the total amount of the probe traffic can be significantly decreased. This property can be especially helpful to decrease the chance to be detected from the cloud administrator's network monitoring systems.

4.7 Probing deletion of rules

So far, we have estimated the update time of iptables by measuring the time when the creation of a firewall rule is actually reflected on the iptables. This is because noticing creation of rules is more immediate and definitive than deletion. When a rule is created, a probe packet that first passes the firewall will be directly observed by the attacker VM. However, when deleted, the VM will simply observe there is a long delay in arrival of the probe packets since the last passed probe packet has arrived, so that it cannot judge whether the delay is due to the update of the iptables or due to network congestion. Therefore, it is necessary to conduct statistical analysis on the arrival and departure times of probe packets to notice iptables update times from deletion of a rule, and this can be an serious obstacle for reactive systems.

Nevertheless, using rule deletion time can be useful in case the number of available rules are very limited and the immediate notification of the iptables update event is not required.

References

[1] Amazon AWS Conntrack. `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html#security-group-connection-tracking`.

[2] Microsoft Azure Network Security Group. `https://azure.microsoft.com/en-us/blog/network-security-groups/`.

[3] OpenStack Conntrack. `https://blueprints.launchpad.net/neutron/+spec/conntrack-in-security-group`.

[4] P. Ayuso. Netfilter's connection tracking system. *LOGIN: The USENIX magazine*, 31(3), 2006.