TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MARKKU VAJARANTA

Security as a Service for Hybrid Clouds

Master of Science Thesis

Examiner: Professor Jarmo Harju
Examiner and topic approved in the
Computing and Electrical Engineering
Faculty meeting on 8 November 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
MARKKU VAJARANTA:Security as a Service for Hybrid Clouds
Master of Science Thesis, 47 pages, 3 Appendix pages
May 2014
Major: Communication Networks and Protocols
Examiner: Professor Jarmo Harju
Keywords: Cloud service, hybrid cloud, Openstack, cloud network, firewall, virtualization, central management, REST API

Virtualization has increased amongst the IT service providers as a method for achieving more efficient server usage. This has led to the concept of cloud services, offered by large data centers with the help of virtualization techniques. Cloud service is an abstract service, which provides IT services as self-service for the end user. For example these services can provide virtual servers as on-demand.

Virtual servers in the cloud are usually reachable from the Internet, so their protection is necessary. This master's thesis discusses firewalling virtual machines inside the cloud, together with virtual firewall distributions and their features. Cloud services are usually self-services and thus their cloud environment and firewall are managed centrally. Automated firewall provisioning and management for a cloud service is described.

Main goal for the master's thesis was to find a feasible centrally managed security system. Vyatta was used as a virtual firewall software and the test environment was built on top of the Openstack cloud. Vyatta included firewall, VPN and routing features suitable for enterprise usage.

The deployed virtual firewall performed well in the test environment with the necessary features and also the central management worked without problems. The central management system was feasible and reusable with other projects. Also the automatic deployment of Openstack cloud was a feasible choice. However, using Vyatta requires a lot of changes at least to the Openstack cloud platform, and hence the network setup and management is difficult.

There are on-going efforts to virtualize networking devices in the data center as well. This will allow the whole service platform to be centrally managed using a single interface. Thus all changes to the network and new virtual service requests can be executed as the customer demands them. Software Defined Networking (SDN) and Network Fuctions Virtualization (NFV) both drive the systems to more virtualized and centrally managed environments, thus providing an important research topic in this field.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
MARKKU VAJARANTA: Security as a Service for Hybrid Clouds
Diplomityö, 47 sivua, 3 liitesivua
Toukokuu 2014
Pääaine: Tietoliikenneverkot ja protokollat
Tarkastaja: Professori Jarmo Harju
Avainsanat: pilvipalvelu, hybridipilvi, Openstack, pilviverkko, palomuuri, virtualisointi, keskitetty hallinta, REST API

Virtualisointi on kasvattanut suosiotaan IT -palveluntarjoajien keskuudessa paremman palvelinkapasiteetin hyötysuhteen saamiseksi. Virtualisointi on myös mahdollistanut pilvipalveluiden syntymisen. Pilvipalvelut ovat abstrakteja palveluja, jotka tarjoavat asiakkaille IT -itsepalveluita esimerkiksi virtuaalipalvelmien muodossa.

Pilvessä olevat virtuaalipalvelimet on kytketty yleensä Internetiin, joten niiden palomuurauksesta tulee huolehtia. Tässä diplomityössä tutkittiin palomuurausta pilvessä sijaitseville virtuaalikoneille, virtuaalisia palomuurijakeluita ja niiden ominaisuuksia. Työssä käsitellään myös keskitettyä hallintaa, jonka avulla voidaan hallita sekä pilvipalvelua että palomuuria. Tästä syystä tutkimuskohteena oli myöskin palomuurin automaattinen provisiointi pilvipalveluun.

Diplomityössä rakennettiin pilvialustoille sopiva keskitetyn hallinnan palomuurijärjestelmä. Käytetty virtuaalinen palomuurijakelu oli nimeltään Vyatta ja koeympäristönä toimi Openstackin pilvialusta. Vyattasta löytyi yrityskäyttöön sopivat palomuuraus-, VPN- ja reititysominaisuudet.

Toteutettu virtuaalinen palomuuri toimi kokonaisuudessaan hyvin, sen ominaisuudet olivat riittävät ja keskitetty hallinta toimi moitteetta. Käytetyt menetelmät keskitettyä hallintaa varten, ja Openstackin pilven automaattiset ominaisuudet, olivat toimivia ratkaisuita. Vyattan käyttäminen vaatii silti paljon muutoksia Openstackin pilvialustan verkkoon, joten verkon rakentaminen ja hallinta tulevat vaikeiksi.

Tulevaisuudessa useat palvelinkeskuksen verkkolaitteet virtualisoidaan. Virtualisointi mahdollistaa koko verkon keskitetyn hallinnan käyttäen ohjelmistoja. Tällöin verkon muutokset ja laitetarpeet voidaan toteuttaa välittömästi asiakkaan toiveiden mukaan. Termit Software Defined Networking (SDN) ja Network Functions Virtualization (NFV) käsittelevät molemmat verkon virtualisointia ja keskitettyä hallintaa ja ne ovat suunnannäyttäjiä verkkotekniikan tutkimukselle.

PREFACE

This thesis was done for Cybercom Finland as a part of their Cloud project. The idea behind it was simply to implement a firewall that covers some of the features missing from the Openstack cloud's firewall. A new **-as-a-Service* began to rise its head.

Ironically, virtualizing a firewall has been in my mind already about ten years ago. After some experiments this did look like a practical solution already at that time. Now my thesis covers the same topic. Openstack, however, turned to be full of surprises especially in the field of networking so without assistance this project would have been exhausting.

I would like to thank MSc Riku Kovalainen and MSc Rolf Koski for finding such an intresting topic and supervising this work for Cybercom. I would also like to thank all the employees in the Cybercom Data Center for intriguing conversations and support with this thesis. This journey to the clouds was delightful to do with people like you.

I want to thank MSc Aleksi Suhonen for evaluating the thesis. My special gratitude goes to my professor Jarmo Harju for guiding me during this work. Such a comprehensive academic viewpoint was very welcome. And last, I thank Leena for proofreading this thesis and encouraging me to carry on with the work. Mixing a cloud, a central management service and a firewall together may sound extreme - and to be honest, it is.

Tampere, April 22th 2014

Markku Vajaranta

TABLE OF CONTENTS

ABBREVIATIONS AND TERMS

ACL	Access control list
AH	Authentication Header
API	Application Programming Interface
AWS	Amazon Web Services
BGP	Border Gateway Protocol
CDN	Content Delivery Network
CLI	Command Line Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DMTF CIMI	Distributed Management Task Force Cloud Infrastructure Management Interface
DMZ	Demilitarized zone
DoS	Denial-of-Service
DPI	Deep Packet Inspection
EC2	Elastic Computing Cloud, AWS
ESP	Encapsulating Security Payload
FWaaS	Firewall-as-a-Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure-as-a-Service
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
KVM	Kernel Virtual Machine
LAN	Local Area Network
LBaaS	Load-Balancer-as-a-Service
NAT	Network Address Translation
NAT-T	Network Address Translation Traversal
NFV	Network Functions Virtualization
NGFW	Next-Generation Firewall
NTP	Network Time Protocol
OS	Operating System
OSI	Open Systems Interconnection

OSPF	Open Shortest Path First
PaaS	Platform-as-a-Service
PHP	Server-side HTML embedded scripting language
PPTP	Point-to-Point Tunneling Protocol
REST(ful) API	Representational State Transfer API
RIP	Routing Information Protocol
RPF	(Unicast) Reverse Path Forwarding
SaaS	Software-as-a-Service
SDDC	Software Defined Data Center
SDN	Software Defined Network
SECaaS	Security-as-a-Service
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
vCloud	VMware cloud solution
VCPU	Virtual CPU
VETH	Virtual Ethernet
VM	Virtual machine
VPN	Virtual Private Network
vRouter	Virtual Router
vSwitch	Virtual Switch
Vyatta	Brocade Software Virtual Router
WWW	World Wide Web

1 INTRODUCTION

The evolution of data center services is again taking a leap forward. The previous one occured when the virtualization techniques emerged and several systems were consolidated into one hardware. Virtual systems have provoked the building of computing environments from where companies and end users can buy resources on-demand. These environments are called as cloud services. Cloud is an abstract service which is based on highly virtualized systems and can be described as a self service computing system.

Networking and communications often rise a question concerning the overall security of a specific system. A firewall is the basic building block for safe public network connectivity. Firewalls are thus used locally in the operating systems or on the border of internal networks to protect the whole network simultaneously. Further, as the IPv4 addressing is depleting, the Network Address Translation (NAT) system is often used to provide private address range for an internal network area. Since cloud networks are often using private IP addressing, they require a device to do the network address translation. It is also common to implement a firewall and a NAT into the same device.

One of the most common cloud services is the Infrastructure-as-a-Service (IaaS) cloud. In IaaS cloud, the user requests for virtual machines that can be connected to the Internet and thus require firewalling. The firewalling of a cloud virtual machine is traditionally done using the cloud platform provider tools. However, for corporate users the platform does not contain enough features. In addition, the cloud may be used by a private user or a company, so separate cloud environments are commonly built. Different environments can be further built using separate cloud platform providers depending on the needs of the cloud capacity.

These come together to form a system described in Figure 1.1. A service hosting operator offers cloud services for both private users and companies. The public cloud meant for private use is built using an open source platform and the private cloud using a commercial cloud platform provider. The management of the cloud system is centralized to a *cloud dashboard* server which is responsible for operating with both cloud platforms. If the inbuilt firewalling is insufficient, other options such as virtual firewall with Virtual Private Networks (VPN) need to be considered. Because the virtual firewall operates in the cloud, a firewall management is convenient to add to the centralized cloud dashboard.

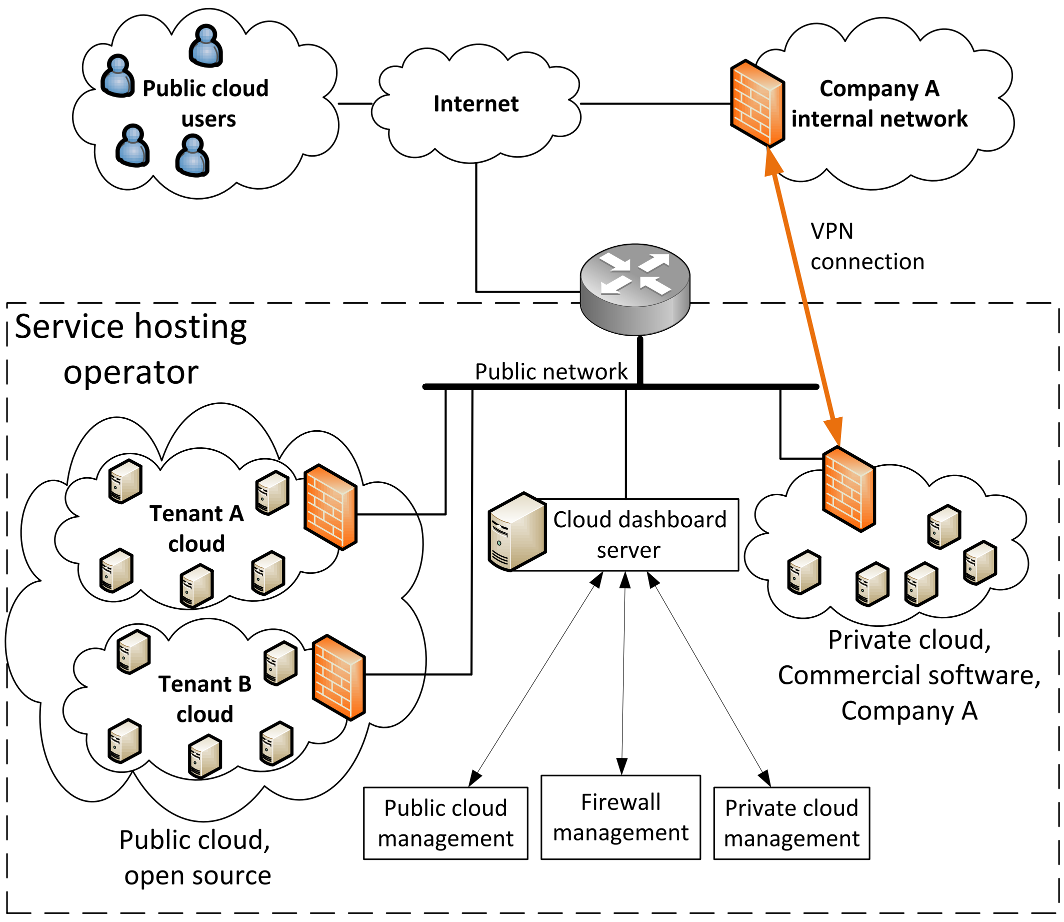

Figure 1.1: An example of a cloud and central management system.

The overall system therefore relies on the central management and its capabilities to operate all the necessary parts. However, since current cloud platforms do not provide sufficient solutions to the firewall management problem, a prototype implementation is built in this thesis.

The thesis includes theory concerning the firewall in the chapter 2. As the system relies heavily on virtualization and cloud computing, the cloud is discussed in the chapter 3. The chapter 4 describes the virtualized firewall, its environment and the chosen system. Implementation of the system is discussed in the chapter 5.

2 NETWORK FIREWALL SYSTEMS

Firewall is often considered as the primary security element. However, it is only a basic part of network security. A firewall is best described as a connection limitation element, which denies undesirable access to the destination system. This chapter describes what a firewall is, what it can protect and how it does that. Some example implementations and best practices are also provided.

2.1 Fundamentals of Firewalling

A firewall in general is a limiting device or a piece of software running inside an operating system to protect one or more computers or network devices from unwanted packets travelling in the Internet. The most important job of a firewall is to decide which traffic or connection is to be allowed through. The administrator specifies policies that control a number of criteria which in turn make decisions about the received and transmitted packets.

First of the validation criteria is the source and destination IP address. A firewall can make a decision whether it lets traffic through, based on an IP address only. Filtering data according to an IP address is very convenient for example in company internal networks. If data from a specific network address area is considered valid, a rule based on that can be created. IP address spoofing may occur and it is described in Section 2.3.

More specific criteria for data filtering are the protocol field and the port number field. Most common protocols are TCP, UDP and ICMP. For example WWW traffic typically uses TCP port 80 and therefore an easy way to deny only WWW traffic is to deny TCP protocol port 80 connections. Port numbers from 1-1023 are so called well-known ports, that are meant for some special service or software. Ports from 1024-49151 are registered to be used with vendor specific applications. The remaining ports, from 49152 to 65535, are dynamic and they can be used by any application. [1, p. 91]

Software firewalls protecting a single computer may sometimes be able to filter traffic based on the target software in the system. In these cases the firewall ruleset may look for addresses, ports or protocols, but it also verifies the target software in the system. This way the system may block any undesired software from reaching the Internet.

Before implementing a firewall, the following questions should be considered: What really needs protection? Who are we protecting ourselves from? What must be allowed? Asking these basic questions will lead to the best compromise between ultimate security and usability. Some of the best practices in firewalling are overseen in Section 2.4. [1]

2.2 Firewall Operation Modes

Firewalls are divided in two categories, gateway firewalls and application proxy firewalls [1, p. 98]. The gateway firewalls are further divided into stateful inspection firewalls and packet filters, also called as stateless firewalls. Despite the nomenclature, the different firewalls have their own place in the networking security. These are discussed next.

2.2.1 Stateless Firewall

Packet filters are the simplest way to limit network connectivity. Filters are usually lightweight and thus cause little extra load to the device running it. Therefore packet filters are usually implemented in network devices such as routers that are designed for other purposes, but can handle these filters.

Packet filters usually operate on the OSI level 3 [2, p. 6]. That causes the filtering to be based on the information in the IP haeder which in turn gives rise to the access control lists (ACL), the foundation of packet filters. A firewall checks whether an ACL contains a match clause to the packet travelling through the filter. The ACL is usually read from top to bottom and in case a match clause is found, an ACL-directed action takes place. In mismatch cases, a deny action usually is executed. An example of an ACL is shown in the Appendix A in Cisco specific notation. The ACL permits some ICMP protocol messages, DNS related traffic on top of UDP and TCP, port 80 (WWW) directed traffic and DHCP traffic (bootps). At the end of the ACL is the implicit deny action to filter out all other messages.

Though stateless packet filters are fast and commonly used, they have several limitations. Due to the low OSI level placement, the filter cannot prevent an attack on the application layer. The statelessness also means incapability to recognize TCP/IP spoofing attacks. In general, all upper layer functions are out of reach.

Normal network activity sometimes requires a firewall to open a certain port to establish a connection. However a stateless firewall is unable to keep track of the destination and source of the connection which compromises security. This problem gave rise to the stateful firewall system. [2, p. 6]

2.2.2 Stateful Firewall

A stateful firewall is in essence a stateless firewall which operates on the OSI level 4 and thus processes more information [2]. It provides session verification for connection oriented protocol, TCP. For each TCP connection, an own session is made and a port is opened on the local computer to receive traffic from a remote system. The same port must be opened in the firewall to maintain a connection between these two.

This kind of a connection places the local computer under the risk of intrusion. The stateful firewall addresses the issue by keeping a state-table containing all outbound connections made by the stateful (TCP) protocol. The inbound connections are accepted based on the source IP address, protocol and port. In addition, the stateful inspection verifies the incoming TCP packets to belong to an established connection. For other protocols, the stateful firewall operates as a packet filter.

The stateful inspection requires higher performance from the hardware than the packet filtering due to the state table it maintains. It is also the reason why the stateful firewalls are generally considered more secure than the packet filters. The stateful firewalls are, however, inconvenient to use as the first traffic filtering elements in high bandwidth networks. Despite all, the stateful firewall is considered secure enough and performs well in most cases as a single firewall device for a company internal network. [2, p. 11]

2.2.3 Application Proxy Firewall

A proxy is a system, which receives messages and re-sends them on behalf of an original source. An application proxy firewall, sometimes called as a reverse proxy, does the same kind of forwarding but it also examines the messages and makes forwarding decisions based on a ruleset. The ruleset may for instance contain information about undesired patterns or messages specified by the network administrator. Based on the packet examination, the proxy firewall verifies the validity of the message. The valid messages are rewritten and sent to the destination, otherwise the firewall rejects the message. An example ruleset of the SMTP application proxy denying spam or abuse cases is shown in the Appendix B. [1, p. 99]

The application proxy firewall offers a lot of security. The greatest advance is that incoming messages are first sent to the proxy and then forwarded to the intended destination [1, p. 99]. This way the backend server is protected from any direct connections. Since the application proxy operates on the application level of the OSI model, all levels can be inspected. Moreover, the message inspection provides extensive logging capabilities that may reveal hacking attempts [2].

There are drawbacks to the application proxy firewall as well. One of the greatest is the need for processing time. Because the message is carefully examined, the firewall spends a lot of time to complete each packet. Thus the application proxy is not the system of choice for the high bandwidth or real-time systems. The setup and maintaining of this kind of a firewall is highly challenging which discourages the building of an application proxy for each server. [2, p. 14]

2.2.4 Implementations in the Networks

The different firewalls have their own purposes in networking. An example network setup containing all the different firewall techniques and their necessary features is shown in Figure 2.1. Regardless of the network structure, more than one firewall can be used and it is even recommended especially if the network is extensive. Multiple layers of firewalls defend the network efficiently and thus provide a safer environment.

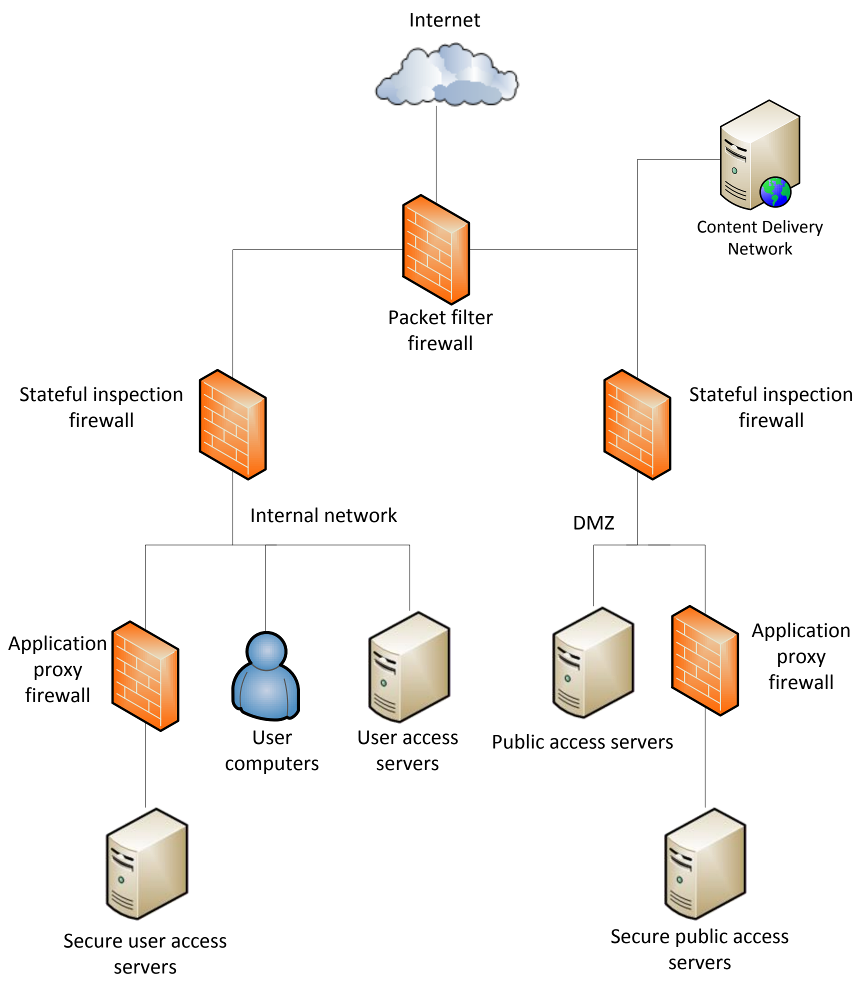
Internet
Content Delivery Network
Packet filter firewall
Stateful inspection firewall
Stateful inspection firewall
Internal network
DMZ
Application proxy firewall
User computers
User access servers
Public access servers
Application proxy firewall
Secure user access servers
Secure public access servers

Figure 2.1: Example firewalling schema.

The network presented in Figure 2.1 is connected to the Internet using a packet filter firewall. The packet filter is the only firewall limiting the network connectivity to the content delivery network (CDN). A CDN requires a lot of bandwidth and hence the packet filter is the firewall of choice. The internal network is further secured by a stateful firewall. Thus it contains all the user computers and servers, as well as application proxy firewalls to enhance the security of some vital servers like database or mail system. The demilitarized zone (DMZ) is likewise placed behind a stateful inspection firewall and, further, contain the application proxy.

The network defense described above includes firewalling from the lightest to the heaviest solution and also from the most generic to the most specific. The packet filter firewall is best placed on the border of the network and the Internet. As the packet filter feature implemented in a router requires little capacity and can easily handle large amounts of data, it won't reduce performance of the router. In the example shown in Figure 2.1 the stateful firewalls are then placed behind the packet filter to secure the organization's internal network and DMZ. This way the stateful firewall maintains the session tables and protects against attacks let through by the packet filter. As some traffic is blocked by the stateless firewall, it lightens the load on the stateful firewall. The application proxies should be placed as the last firewalls in front of the servers requiring highest security. A more common implementation, however, is to use only a stateful firewall to filter all traffic on the edge of the network. This firewall hence provides stateful inspection for TCP and packet filtering for other protocols.

2.3 Preventing Attacks

Manufacturers or developers may have built special features on firewall systems to prevent some specific kinds of network attacks. These attack recognition systems deliver specific policies to a firewall which then operates as a limiting element. Below are described some of the most common systems that operate to support firewall.

Intrusion detection systems (IDS) are an effective means for identifying network attacks or hacking attempts. There are two different main types of recognition systems, a signature-based and an anomaly-based. The signature-based system seeks signatures for commonly known attacks whereas the anomaly-based system creates statistics to represent the normal network activity and acts when an alteration is found. However, the IDS only gathers the information into a log and it is up to the administrator to take action. [3]

Intrusion prevention systems (IPS) include all the IDS features in addition to an automatic action for incidents. Some common procedures for preventing attacks are terminating the connection or session, blocking access to the target due to a specific attack attribute such as source IP, and blocking access to the destination resource.

[4]

Deep packet inspection (DPI) is a method, that integrates a stateful firewall with intrusion detection and prevention systems. The stateful firewall examines the received packet and the intrusion detection system will inspect the message carried. The DPI provides significant benefit when compared to a traditional stateful firewall system because the message is also inspected for fraud patterns. The disadvantage of the DPI is that it requires a lot of computating power due to the heavy modifying of the packet as well as the need for the application state information. [5]

The methods presented above detect attacks below application layer. However, nowdays more and more traffic uses common service ports and protocols such as HTTP. This has caused the protocol and port based firewalling to be less effective. Gartner (2009) has presented the Next-Generation Firewall (NGFW) to be a system detecting and preventing these attacks in both inbound and outbound directions. It is also declared that the IPS and DPI methods are ineffective against these attacks [6]. Also, the Spire research (2004) claims: "The deterministic intrusion prevention is the next generation firewall with deep packet inspection." [7]. The NGFW is a system that evaluates the protocol stack all the way to the application level and protects systems within the network from malicious traffic that may be hidden inside common traffic.

2.4 Best Practices

Firewall solutions as a research topic have been going on for years. While the development of the security systems progresses, also the intrusions and attacks become more sophisticated. Security issues are widely considered resulting in some best practices in network security.

Building a firewalling system begins by planning a firewalling design. The design should include information such as what needs to be protected, from whom, what methods and devices are in use and how to protect the destination. Also possible bottlenecks and performance issues need consideration.

The out directed filtering is often forgotten. Companies tend to maintain the inbound data filtering – whether stateful or stateless – well, but no outbound limitation rules are used. The outbound limitation rules are very delicate to plan but are worth having, because by filtering out directed traffic some infected computers can be prevented from sending data into the Internet. Compromised systems in the internal network are unfortunately a reality. For example, the Internet Relay Chat (IRC) protocol is often used by malicious software to communicate with hackers [1, p. 117]. Thus by denying connection from typical IRC ports, it may be possible to prevent computers from ending up in a botnet or the likes.

Layered security architecture, also called as the defense in depth, enhance the security of the firewalling. The main concern is to efficiently prevent breaking into systems while still providing a fully functional network. The defense in depth also provides security when a server has been compromised, since the network is protected from itself also. An example network built with defense in depth is shown in Figure 2.2.

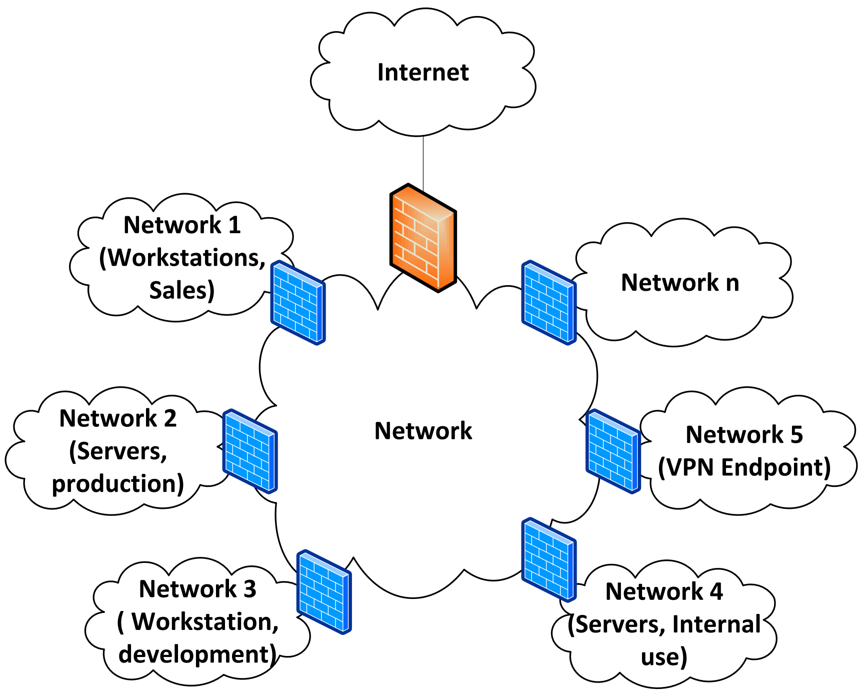
Internet
Network 1 (Workstations, Sales)
Network n
Network 2 (Servers, production)
Network
Network 5 (VPN Endpoint)
Network 3 (Workstation, development)
Network 4 (Servers, Internal use)

Figure 2.2: Defense in depth example network.

This network contains two different types of firewalls: the stateless and the stateful. The firewall between the "Internet" and the "Network" is stateless and the others are stateful. The stateless firewall should be replaced by a stateful one if the device can handle traffic flow. The purpose of multiple firewalls is to have the networks separated by their function and vulnerability. The separation can be based on the role of the servers and workstations in the entity. This creates an environment where one network area can be protected from the others. Even if one network is compromised, the others remain safe.

Figure 2.1 demonstrates a basic defense in depth firewalling by containing a number of separated zones. Each network is an zone of its an own and the most important computers are placed behind several firewalls from the Internet. The figure also illustrates the use of an application proxy firewall as the protector of a single server, which is the recommendation.

When using zones, in a firewall or router device, an ethernet interface is applied in a zone. Policies and routing decisions take place between zones. Figure 2.3 shows an example of a traditional zone setup. The router contains four different interfaces and four different zones. One of these is a *local zone*. In the zone based firewalling all traffic must travel from one zone to the other. A local zone has been invented to allow traffic to the router itself. However this is not compulsory.

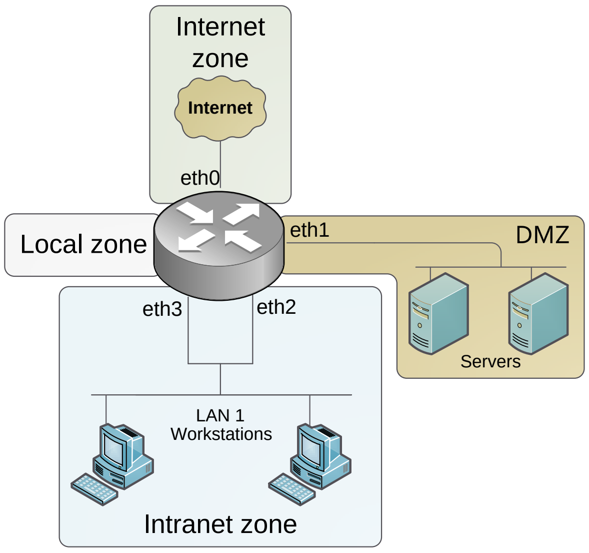

Figure 2.3: Router example zone assignment

Other common zones include Internet, internal network and a DMZ zone. As shown in Figure 2.1, the main purpose of the DMZ is to contain all public servers. The packet filter and the stateful firewall preceding the DMZ contain open ports to allow traffic to services running in the public access servers while the internal network remains closed. If more security is desired for a server within the DMZ, it can be placed behind an application proxy firewall, for instance, or in another DMZ where the firewall policies are stricter.

In the example zone assignment (Figure 2.3), the Internet is connected to the router only via eth0 interface. The router is further connected to the DMZ by the eth1 interface whereas the internal network zone connects to it by both eth2 and eth3 interfaces. The latter interfaces are bonded together to share an IP address and they are therefore considered as a one logical security area containing the same security policies.

The deny action should always be used as the default setting in the firewall. Exceptions can then be added to allow required traffic. This way the risk for an accidental harmful firewall policy is minimized.

Each network, or zone, should have specific firewall rules in both directions. The firewalling policies should exist between all zones, including internal network and the DMZ. The servers in the DMZ are naturally more vulnerable to attacks, so some of them may become compromised, yet the internal network remains safe thanks to firewall policies. Additional security to the internal network of a company against attacks from within is brought by the different zones and the defense in depth. In general, any user having access to the local internal network should be considered as a potential threat [2, p. 21]. A direct access to the network enables traffic sniffing or attacks against the servers that can be reached from the internal zones.

These guidelines about organization firewalling are methods to enhance network security. Unfortunately there is no silver bullet for making a secure network. Some essential elements are network monitoring devices and an established plan for discovered attacks. However, true security comes only from the persistent development of the security architecture.

3 CLOUD SYSTEMS

The cloud systems are computational systems providing an environment for running large numbers of program code or operating systems (OS). Especially servers hosted in a cloud system require virtualization by the cloud platform. The virtualization is thus a key feature in the cloud system.

This chapter will discuss virtualization in general, how cloud relates to it, what a cloud environment is and capacity viewpoints. As the cloud environment grows larger, an universal and easy method for communicating with different parts is required. This chapter also covers the management of a cloud environment, its networking and firewalling.

3.1 Virtualization Trend

In the computing context, virtualization can be considered as an abstraction of a service, storage, device or network. Implementations include computers, devices and software just to name a few. Virtualization lowers the costs of hardware remarkably and reduces idling of hardware by providing software acting like hardware.

Virtualization is not limited only to virtual machines. Almost anything can be virtualized as denoted by Ameen and Hamo in: "Survey of server virtualization" [8]. An entire computer can be virtualized, or only the desktop, for instance. In desktop virtualization, the desktop is located on a server and displayed elsewhere. Storage virtualization reserves logical blocks from a physical storage device. These are only a few examples of virtualization. Virtualization consolidates resources used by multiple parties into one hardware system.

Virtualization has grown rapidly over last few years especially among the hosting operators. With virtualization, hosting companies may have more than one operating system running in hardware. A piece of software running multiple operating systems simultaneously is called a *hypervisor*, and the system running a hypervisor software is referred to as a *host machine*. A hypervisor provides a platform for virtual machines (VM) that are also known as *guest machines*. The hypervisor can further be separated in to two main types. The hypervisors of type 1 run directly on hardware, whereas the type 2 hypervisors run on a host operating system. Type 1 is also known as a bare metal, or native, and type 2 as a hosted hypervisor. Both of these are built on top of an operating system. The difference is that the type 1 host operating system is for hosting VMs only and the type 2 host operating system is a regular desktop OS such as Windows 7 with a VirtualPC hypervisor for running the VMs. [8]

The power of server virtualization comes from running several operating systems parallel in a piece of hardware. Virtual machines can change their host machines fluently on the go in large virtual environments. Furthermore, snapshots can be created of the VM to transfer the VM to another environment. Thus a server operating system can accurately be set up in the hypervisor. [9]

The easy deployment of the VM is the key to popularity. In traditional virtualization the deployment process can only be done by the system administrator. However, the need for rapid server delivery for the end user has been rising among the VM hosting operators. The answer is a cloud environment.

3.2 The Cloud in a Nutshell

The Cloud is an abstract service usually based on highly virtualized environments that can be activated as an on-demand self-service and that gets billed by the usage. Cloud VMs are always available in the Internet and are easily deployable. The cloud should not be limited only to virtual servers as the following common levels of services already exist: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Their relationship is displayed in Figure 3.1.

The SaaS delivers the contents of the highest OSI layer to the user. In Software-as-a-Service the software publisher sells to the end user only the software access via the Internet. Software is then delivered as a service when requested by the user. [11]

The PaaS hosts a platform for an application development environment that includes all the necessary modules for software developing. With the PaaS systems the developer is able to focus on the product instead of the environment. Software developed using the PaaS tools is often run in the cloud as a SaaS system. [12]

The IaaS is on a lower in OSI level compared with SaaS or PaaS. With this service the customers, also called as tenants, can rapidly deploy servers to evaluate a system scenario or software. A tenant can be either an individual user or a company that controls a specific set of virtual servers inside the cloud. As the IaaS cloud operates on a pay-as-you-go model, the costs are minimal in short term use. Also the deployment and testing phases become easier with software licensing. However, in long term use the costs may rise above the value of hardware. [13]

The IaaS cloud can be done with hardware only, but this requires a complex management system. The IaaS thus benefits from virtualization because of the easy

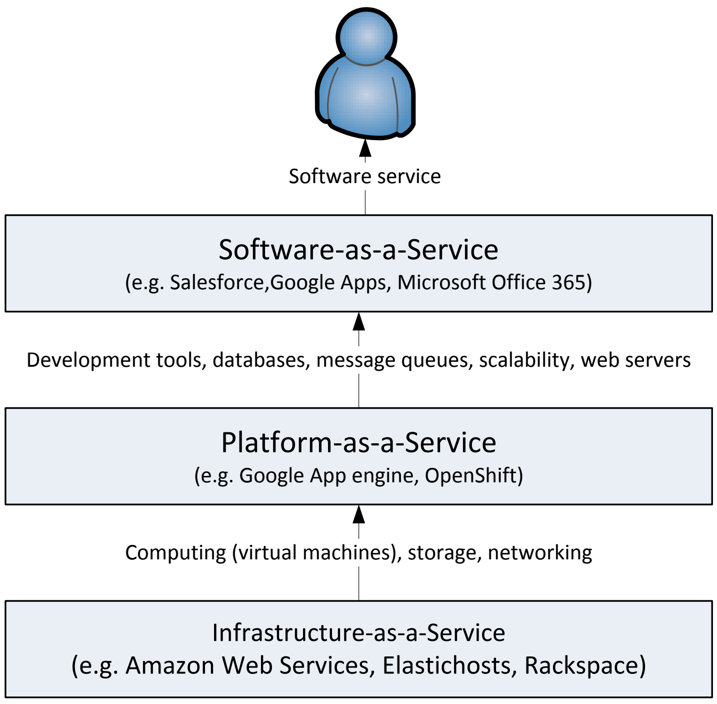
Software service
Software-as-a-Service
(e.g. Salesforce,Google Apps, Microsoft Office 365)
Development tools, databases, message queues, scalability, web servers
Platform-as-a-Service
(e.g. Google App engine, OpenShift)
Computing (virtual machines), storage, networking
Infrastructure-as-a-Service
(e.g. Amazon Web Services, Elastichosts, Rackspace)

Figure 3.1: Cloud service stack [10]

and fast setup of the virtual machines. Virtual machines are also called as *instances* when they are placed in the cloud. A speciality of the instances is that they have some highly dynamic resources available. These resources are further used as *flavors* in the instances. Each instance has its own flavor that specifies the number of virtual CPUs, the amount of memory and the hard disk assigned to it. A flavor follows its instance in the cloud when transferred from one host to another. The instances can be freely moved inside the cloud and this is usually done automatically to balance the load in the cloud platform. The moving operation is also called as *migration*.

Hardware systems hosting the cloud can vary a lot, being either a single server or a full-sized server farm. The size depends on the needs of the system. If only a proof-of-concept is intended, an IaaS cloud can be set up in a VM. On the other hand, a full-sized multi-tenant cloud environment requires a great deal of capacity. Also commodity hardware, meaning hardware no longer used for production, sometimes runs the cloud.

Cloud systems include four different deployment models: private, community, public and hybrid cloud. *Private* clouds are for one organization only. A *community* cloud is shared between co-working groups. The private and community clouds are usually managed by the owner of the cloud. *Public* clouds are available for everybody including individual users, groups and companies. They are set up and sold by organizations specialized in cloud services. A cloud infrastructure including two or more of the clouds presented above is called a *hybrid* cloud. The hybrid cloud is often customized and priced depending on the customer requirements. It also offers a good platform as the backend solutions are separated, yet data and applications can be migrated between the clouds. [14]

Cloud platform providers such as Openstack or VMWare can be used to set up a cloud with any of the presented cloud deployment models, and all the deployment models can be created using one or more cloud platforms. The cloud set up using more than one platform provider is called a *multi-platform* cloud. When a private cloud is set up using one platform and a public cloud using another, it becomes a multi-platform hybrid cloud. A common implementation is to have a public cloud deployed on top of an open-source cloud platform provider and a private cloud on top of a commercial cloud platform provider.

Openstack is one of the open-source cloud computing projects. It aims to provide comprehensive software to manage vast amounts of resources for the cloud [15]. The Openstack architecture consists of different nodes, that are responsible for specific service areas. Figure 3.2 displays these and their relationships. The different architecture parts are a network, a block storage, a compute, an image service, an object storage, an identity and a dashboard [16].

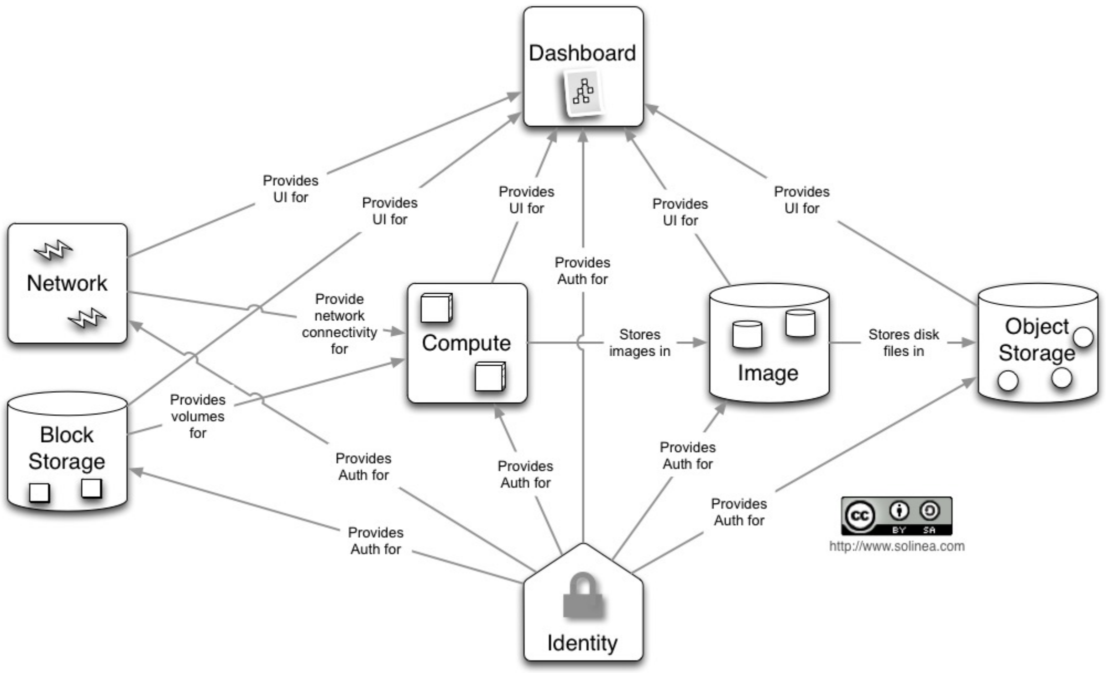

Figure 3.2: Openstack conceptual architecture [16]

The virtual machines are running in the compute node and the other parts provide supporting elements for them. The network is responsible for the network connectivity and the image service provides storage volumes for the VMs. The image service stores actual virtual disk files to the object storage. The only visible part for the user is the dashboard which provides the frontend service. The identity service authenticates the use use of all these parts. The parts also need to communicate with others in an effective way in order to have a stable and fast environment for the customer. The messaging between nodes is done using universal Application Programming Interface (API). [16]

3.3 Managing Large Scale Cloud Environment

Cloud environments set up using multiple platform provider products can be centrally managed by a set of custom tools. One of these is REST API which is discussed in this section. Also a central management tool, Deltacloud, is briefly explained.

3.3.1 REST API

A Representational State Transfer API (REST API) features the use of URI-identified resources and commands instead of traditional application state resources. The benefit is that the server is no longer required to maintain a session. Without the state information the resource can be freely used all the time, making it the possible for many clients to use the same application resource simultaneously. [17]

The REST is built on top of an HTTP. The HTTP is an access method for applications that communicate using REST URI statements. This way the application does not need any specific protocol for communication as this happens fluently with HTTP traffic. There are four main types of messages implemented in the REST system: *put*, *get*, *post* and *delete*.

The *put* message generates a new resource to the server and this is used afterwards by other messages to identify the right sessions. A *put* message returns the resource URI to the client in message exchange. A *get* message sends a message to the server application and requests for data information from the specific URI. With *post* messages it is possible to send information to the server for updating. The update can either be for an existing resource, or it can create a new resource which then returns the information like the *put* message. After a resource is no longer needed, it should be deleted by sending a *delete* message to appropriate URI. [17]

3.3.2 The Deltacloud

The Deltacloud provides an abstraction layer for managing cloud environments set up by using products of many platform providers. It offers three distinctive API layers to work with different cloud systems: Deltacloud classic API, Distributed Management Task Force Cloud Infrastructure Management Interface (DMTF CIMI) API or EC2 API [18]. The cloud set up with multiple cloud platforms can thus be managed by a central management portal. An overview of the Deltacloud is displayed in Figure 3.3. Customers connect to the deltacloud REST API through client software and the deltacloud provides the connectivity, for example to EC2, through "deltacloud-driver-ec2" driver. EC2 refers to Amazon Elastic Compute cloud [19].

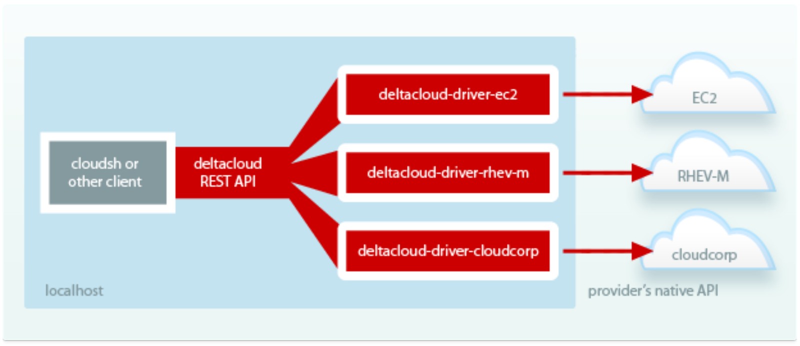

Figure 3.3: Deltacloud overview [18]

The drivers are not limited only to the commercial cloud operators such as Amazon or Rackspace. There are also a number of open-source cloud platforms (Openstack, Eucalyptus). The cloud environments can have parts from a commercial cloud platform like VMWare vCloud as well as from a free open-source environment, and yet be managed by the same portal.

An environment running more than one cloud operating system simultaneously provides environment with a very high availability. The customers will rarely end up with an entirely unaccessible cloud. When a cloud set up on one platform provider environment crashes, the other system will still be functional. Naturally the VMs in a crashed cloud are unusable, but at least the customers will be able to launch new virtual machines in another cloud operating in the Deltacloud.

3.4 Hypervisor Capacity and Load Estimation

Virtual machines running in hardware share its resources. This may be done equally or by approximating the need for resources for each VM. A hypervisor allocates a number of CPUs, physical memory and disk for VMs. The CPU, the disk and the memory are classified as private by the VM, but in reality the hypervisor shares these with all virtual machines. This causes capacity problems when many machines require plenty of a particular resource.

"VM consolidation: A real case based on OpenStack Cloud" by Corradi, Farelli and Foschini discusses VM CPU performance related to the host CPU in the Openstack Diablo cloud. The test environment was running Intel Core Duo E7600 @3.06GHz CPU and had Kernel Virtual Machine (KVM) as a hypervisor. The virtual machines contained one virtual CPU (VCPU) with fair processor scheduling by the host system. The results of the CPU load experiment are shown in Figure 3.4. [20]

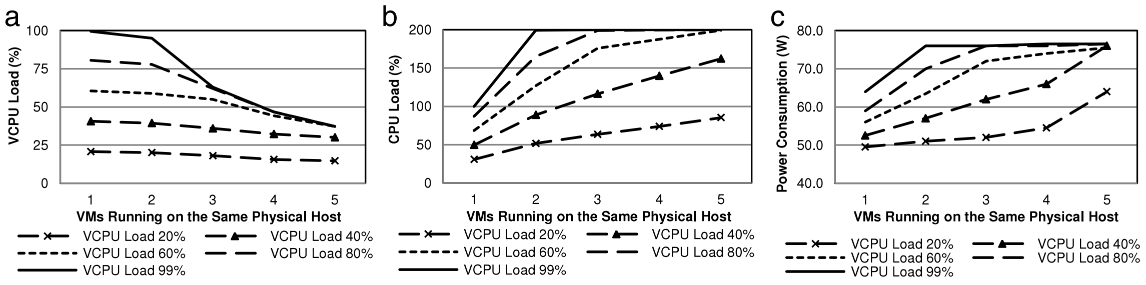

Figure 3.4: VCPU Load (A), aggregated host CPU load (B), and power consumption (C) for the raw VCPU test. [20]

The VCPU load graph (A) demonstrates the virtual CPU state when more virtual machines are added to the hypervisor. On the X axis are the number of virtual machines running, on the Y axis is the actual VCPU load per VM and the lines display the amount of required VCPU load. There are two physical CPUs, which is directly seen on the graph: the two virtual machines with a 99% CPU load requirement can both use almost as much computing power as they need. However, right after the third VM the load increases and the CPU time has to be shared between the virtual machines, which causes loss of VCPU performance on each VM. As more virtual machines with high requested CPU times are added, each VM ends up having less and less CPU time.

In Figure 3.4 the host CPU load graph (B) displays the hypervisor CPU usage with changing number of VMs and their loads. The Y axis represents the hypervisor CPU load and the maximum is 200% because the host contains two CPUs. As in the VCPU load graph (A), the knee point comes after the second VM. The host manages well two machines at the most. When the third VM requires CPU time more than approximately 80%, the host CPU is too heavily loaded to provide the CPU time requested. The knee point also correlates with the number of physical CPU cores. The VCPU loads drop when the host CPU reaches a 200% usage. This may cause problems in a hypervisor that has a constant load. The constant load occurs especially with a VM running a router, a firewall or a VPN service. A hypervisor operating with such virtual machines should be designed to answer the needs of the expected normal VM state.

Virtualization reduces the idling of hardware and the power required per VM. In Figure 3.4 the power usage graph (C) displays the power consumption of different VCPU loads with an increasing amount of virtual machines. The knee point, like before, is reached when the hypervisor becomes fully loaded. As a conclusion, an environment can handle a lot of virtual machines as long as they don't require constant high CPU usage.

3.5 Firewalling and Network Abstraction in the Openstack Cloud

Cloud virtual machines require firewalling as well as physical computers. It is common to firewall either a single cloud VM or the whole network where the VM belongs to. Because of the abstract network topology in cloud computing, firewalling is, unlike in the traditional network, done by security groups in the Openstack cloud.

A security group is a firewall ruleset that contains at least one statement including protocol, destination and source port, as well as source address statements for each group. The security group is then mapped in the Linux *iptables* that provides a stateful firewall for the client virtual machines [21]. Iptables is a Linux package that configures the Linux kernel firewall. Iptables is stateless by default but the network configuration script changes it to become stateful. The iptables firewalling has a wide understanding of the different networking protocols but security groups can only be configured for regular TCP, UDP and ICMP traffic in the Openstack [22].

A typical network and firewall setup is shown in Figure 3.5. Each tenant, or customer, controls a number of VMs located in the network area reserved for that specific customer. The IP addressing in these networks is done by using a private addressing range. The Internet connection for VMs is accomplished with the use of a NAT. The NAT translates private IP addresses to public IP addresses. In the Openstack cloud platform, these public IP addresses are called as *floating IP* addresses.

A floating IP is a public IP address that follows the client VM in the cloud. The floating IP is bound to a specific VM using 1:1 NAT. Even if a customer requires a direct public IP address to the VM, a private IP, NAT and floating IP, are used. Without this, a transparent live migration between cloud compute nodes would not be possible when using a traditional network.

The Openstack cloud networking node uses a networking *namespace* to isolate traffic inside a single cloud node. In namespace several networking layers are run parallel in Linux with each layer having its own firewall, NAT policies and routing table. Thus namespace is a totally isolated environment that has its own networking stack. It tolerates overlapping IP addressing ranges for client networks. [22]

Although the networking in the cloud platform is very abstract, it requires a lot of different virtual components such as interfaces, ports and switches to be fully functional. Further, the individual virtual machines in the cloud also need them to gain connectivity to the Internet or to other VMs. The Openstack has approached this in the Grizzly release by using a separate Networking-as-a-Service called Neutron

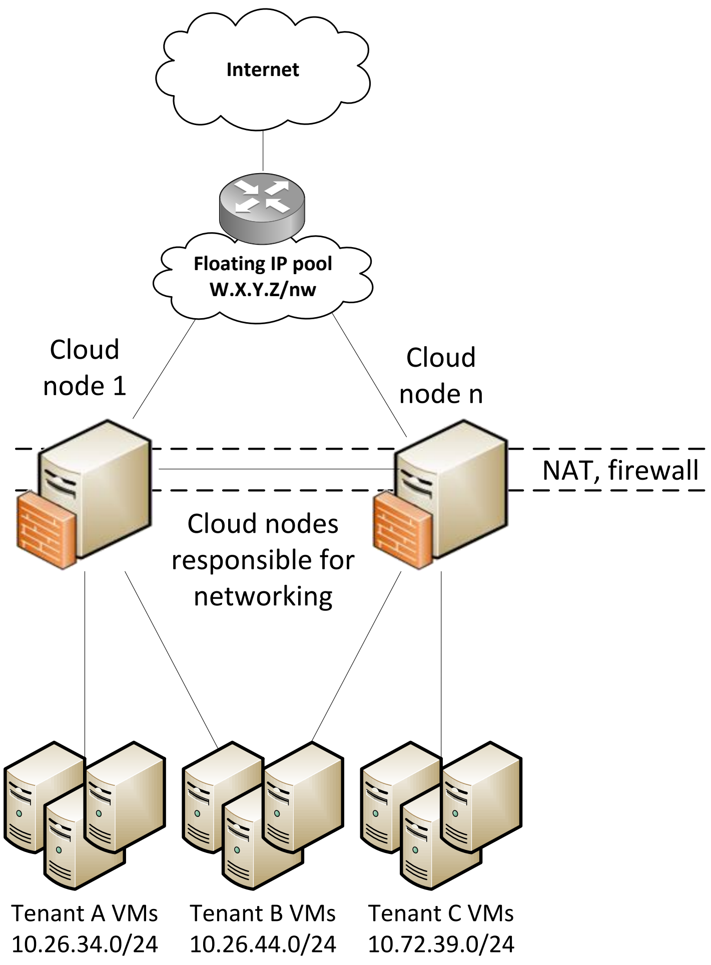

Figure 3.5: Typical cloud networking and firewall structure.

(formerly Quantum) [23]. The Neutron relies on different virtual plugins running in a host Linux OS. Traditionally, the Open vSwitch plugin is used together with other plugins to provide network functionality.

The Open vSwitch is an open-source L2 virtual switch especially designed to transfer traffic both between the VMs and with the outside world. The vSwitch is thus responsible for transferring traffic inside a single cloud node and between all VMs in a single hypervisor. Another plugin similar to vSwitch is a Linux bridge plugin. The Linux bridge acts like a virtual *hub*, repeating every message it receives to all other ports. The Linux bridge could even replace the Open vSwitch, but it lacks the necessary features such as VLAN.

However, the vSwitch and the Linux bridge need to be connected to other plugins inside a cloud node. This can be accomplished by using a plugin called *virtual ethernet (VETH)* pair. A VETH interface transmits all traffic it receives directly to another VETH interface and thus can be considered as a virtual patch cable.

Because the physical network contains L3 devices such as routers, the virtual network needs also their equivalent. The Open vSwitch uses *internal ports* to operate as an L3 interface. The internal ports are used for routing or Dynamic Host Configuration Protocol (DHCP) server, for example. DHCP server allocates automatically IP addresses for devices in a network. A DHCP is thus required for an automatic IP address allocation for VMs in the cloud. The Neutron uses a DHCP server called *dnsmasq* to provide this.

An interface is also an imporant part of a virtual network. It is required to connect to either a physical network or a VM. A physical interface such as *eth0* is an entrance point to a physical network. A virtual interface called *TAP* is used inside a virtual system. A TAP device is an L2 level virtual ethernet interface that is especially used in a cloud hypervisor to communicate with a VM. A TAP device is named as *vnetX* where the X is a running number of the interface.

When using the Open vSwitch plugin for a network environment in the Openstack cloud platform a network setup for both compute and network node is required. The networking node structure is shown in Figure 3.6 and the compute node network structure in Figure 3.7.

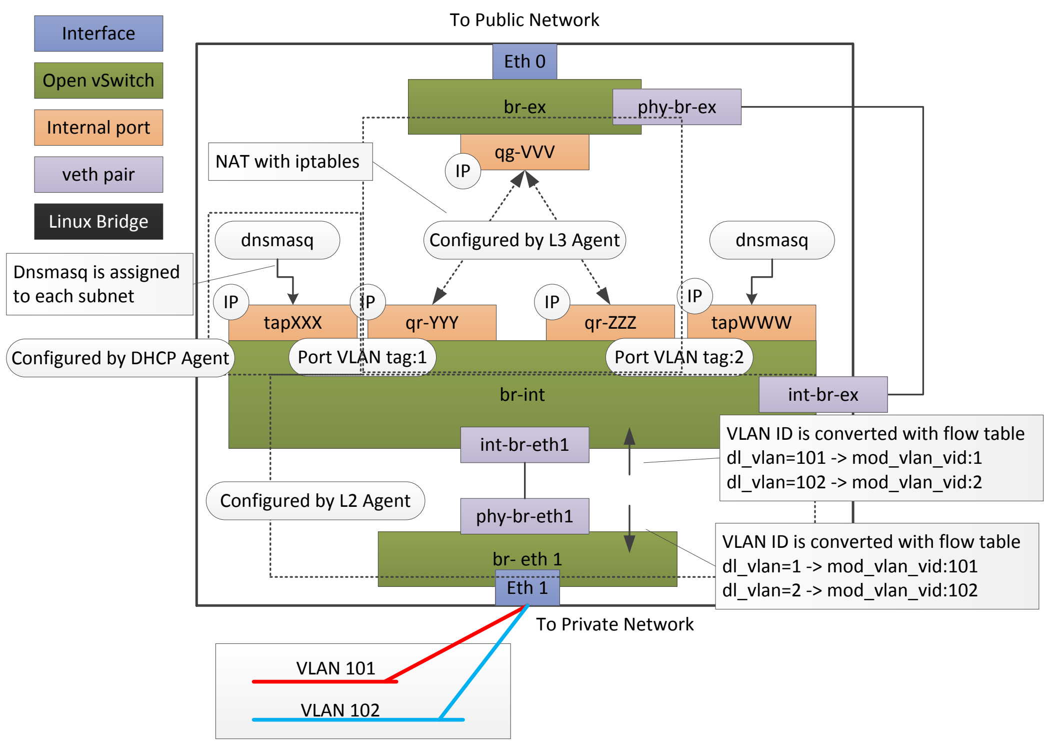

Figure 3.6: Networking node structure [22].

A public network and the networking node are connected by an interface named *eth0* in Figure 3.6. Incoming traffic is forwarded from the *eth0* to an open vSwitch *br-ex*. The internal ports *qg-VVV*, *qr-YYY* and *qr-ZZZ* form a router responsible for transferring traffic between the open vSwitches *br-ex* and *br-int*. A namespace is used to create the router. The ports *tapXXX* and *tapWWW* are connected to the *br-int* vSwitch and they are considered as DHCP server ports. Thus the virtual port *tapXXX* operates as a DHCP server and the port *qr-YYY* as a router for the network in VLAN 1.

The vSwitch *br-int* is responsible for keeping virtual network traffic isolated. The internal VLAN ID that is used for isolation is converted to a suitable global VLAN ID between the vSwitches *br-int* and *br-eth1*. A VETH pair (*int-br-eth1, phy-br-eth1*) between vSwitches makes the data flow. The vSwitch *br-eth1* is also connected to the interface *eth1* to gain access to a private network.

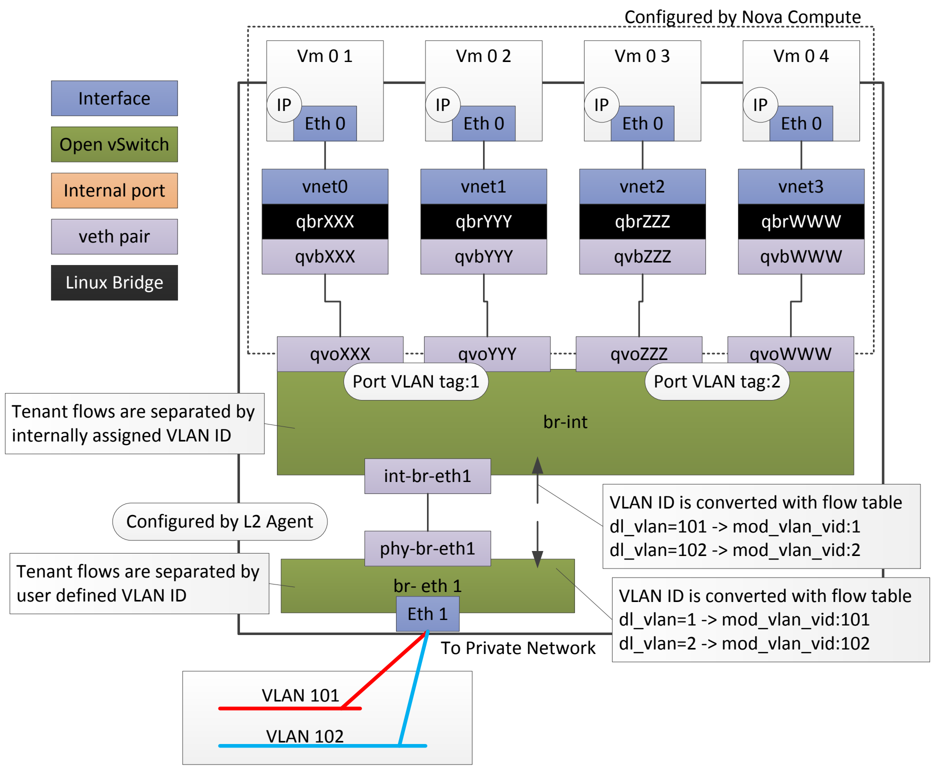
Configured by Nova Compute
Interface
Open vSwitch
Internal port
veth pair
Linux Bridge
Vm 0 1
Vm 0 2
Vm 0 3
Vm 0 4
IP
Eth 0
vnet0
vnet1
vnet2
vnet3
qbrXXX
qbrYYY
qbrZZZ
qbrWWW
qvbXXX
qvbYYY
qvbZZZ
qvbWWW
qvoXXX
qvoYYY
qvoZZZ
qvoWWW
Port VLAN tag:1
Port VLAN tag:2
br-int
Tenant flows are separated by internally assigned VLAN ID
int-br-eth1
Configured by L2 Agent
phy-br-eth1
VLAN ID is converted with flow table dl_vlan=101 -> mod_vlan_vid:1 dl_vlan=102 -> mod_vlan_vid:2
Tenant flows are separated by user defined VLAN ID
br- eth 1
Eth 1
VLAN ID is converted with flow table dl_vlan=1 -> mod_vlan_vid:101 dl_vlan=2 -> mod_vlan_vid:102
To Private Network
VLAN 101
VLAN 102

Figure 3.7: Compute node network structure [22].

The private network is responsible for transferring data between all the compute and network nodes. It can also be described as a backbone network for internal cloud traffic. The compute node virtual network structure and its connection to the internal network is shown in Figure 3.7. In this example, the internal network is connected to the *eth1* interface at the compute node. The *eth1* is then connected to a virtual switch *br-eth1* and this is further connected to another vSwitch *br-int*. Again, these two vSwitches are required to make a VLAN ID conversion between the global and internal VLAN IDs. The virtual switches *br-eth1* and *br-int* are connected by using the same VETH pair (*int-br-eth1, phy-br-eth1*) as in the networking node.

The vSwitch integration bridge (*br-int*) running in the compute node (Figure 3.7) is the central part of the virtual network structure. The virtual networks are separated by using customer specific VLAN IDs on the *br-int* vSwitch. Therefore all VMs in the same VLAN communicate through this vSwitch inside a single hypervisor.

There is one end of a VETH interface for each VM to connect them with the *br-int* vSwitch. The interfaces are named as *qvoXXX*, *qvoYYY* etc. The other end of the VETH interface pair is named *qvbXXX* and it connects to a VM specific virtual Linux Bridge device named *qbrXXX*. The *XXX* part in the name identifies the interface or bridge to belong to a single VM. A virtual TAP interface like *vnet0* is connected to each Linux Bridge and is also the VM interface seen by the guest OS. The TAP interface needs to be connected to the vSwitch via the Linux Bridge because the iptables rules are assigned to the TAP interface and the Open vSwitch is not compatible with TAP interfaces that contain iptables rules.

As a conclusion, the Openstack cloud platform provides a virtual network environment that mimics the structure of a physical network. The current configurable parts, the firewall and the NAT, are however seldom enough for a more advanced user. The complexity of the network plugins is necessitated by the existence of multiple networks as well as the NAT and firewall features. Unfortunately this also means that additional means of security are difficult or almost impossible to implement. Another solution like a virtual firewall is thus implicated.

4 THE VIRTUAL FIREWALL

Firewalling in the cloud can be done using either a cloud platform provider implementation, a hardware appliance, or a virtual machine inside the cloud itself. This chapter describes the current problems in firewall implementations, how the firewalling of a cloud network is traditionally done and future prospects. Also the centralized management is considered in this chapter.

4.1 Security-as-a-Service Concept

A traditional way to filter traffic in the cloud network is to use the cloud platform's own firewall. However, common features in the firewall appliances such as VPN, IDS, load balancing and routing are currently not available in the Openstack cloud platform for example. Even if a cloud platform provider implements more features, these may be missing from the other providers, which causes problems in a multi-platform cloud environment.

An alternative is a separate security solution featuring a stateful firewall, a VPN endpoint and also a routing daemon. The cloud environment also requires a suitable networking structure to operate with a firewall VM. Controlling a VM in the cloud is usually done via a local console or an SSH connection. A centralized management through public network is needed for the multi-platform hybrid cloud, requiring a REST API or some similar feature from the security solution. The multi-platform hybrid cloud requires a multi-purpose firewall VM that can be launched on-demand in the cloud and used as a gateway device for a specific cloud network segment. This concept is known as Security-as-a-Service (SECaaS).

The SECaaS concept may provide all the missing features described above in addition to many more which are required by the hosting operator or the customer. The SECaaS may for example be a fully sized NGFW that monitors network traffic for intrusions. Also the SECaaS offers a great visibility to a single firewall and to the whole network because all SECaaS devices are controlled centrally from the cloud management and they report their state to the control center. In addition to logging capabilities through the central management, also configurations for desired services can be distributed to a number of devices.

The cloud network can likewise be protected by an external firewall appliance. However, the deployment time for such an appliance is longer than for a virtualized one. Also the virtual firewall is usually the best choice because the cloud is meant to offer on-demand services for the user with the pay-as-you-go -model.

4.2 Virtualizing the Firewall to Cloud

A virtual firewall is usually set up on top of Linux or Unix and it runs a special set of software capable of operating as both a router and a firewall. The features of a virtualized firewall and a traditional firewall appliance are mostly alike. The major difference is the device running the firewall engine. A firewall appliance is a physical device, whereas a virtualized firewall is just a bunch of code running inside a hypervisor.

Since the virtual firewall is a VM running inside a hypervisor, it can be cloned. Therefore it is relatively easy to clone the required amount of firewall VMs to provide firewalls for multiple separate networks. The virtual firewalls can exist in the cloud which opens new deployment options. For instance, the firewall can be launched as an on-demand self-service using the pay-as-you-go billing. This means that the end user is no longer required to wait for the vendor to deliver the appliance. Instead of the traditional physical appliances, the firewalling and network setup is done using only software.

The virtual firewall functionality is based on a VM containing at least two interfaces between which the firewalling is done. The cloud network traditionally operates by using private IP addressing and thus the virtual firewall is required to use a NAT between the interfaces. The virtual firewall transfers the NAT and the firewall operations from the cloud platform solution, such as Openstack Neutron, to a virtual machine running in the same cloud. This makes the load caused by the firewall and NAT to the Neutron module to be distributed in the computation nodes of the cloud. The conventional packet flow in the Openstack network using the Open vSwitch plugin from the end user to the VM is shown in Figure 4.1.

Figure 4.1 contains the network structures that were described in depth in Section 3.5. There are two main parts: the cloud platform network and the compute nodes. The network nodes (Neutron) are connected to the cloud provider's edge router using the *eth0* interface. Traffic exits and enters the Internet here. The *eth1* interface in the Neutron nodes operates as the connection point to cloud's private network where data is exchanged between all the necessary nodes. The compute nodes containing the VMs are connected to this private network using the *eth0* interface.

A more simple setup is displayed in Figure 4.2. There is now no separate network node, and thus the public network is directly connected to the compute nodes. The NAT and the firewalling are done by a firewall VM instead of a Neutron network node running *iptables* and the NAT. Traffic between the VMs in the same compute

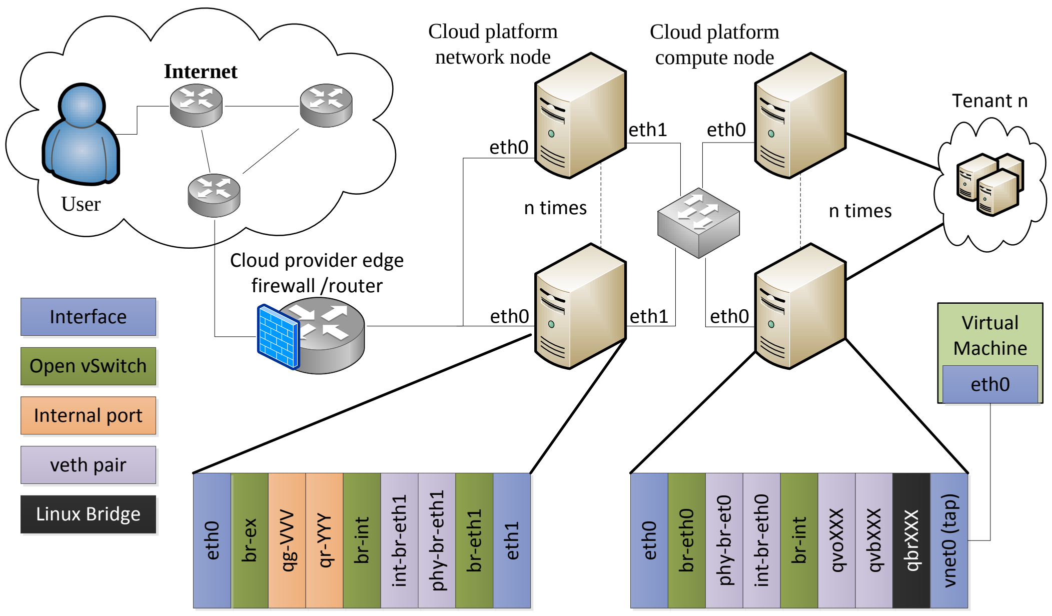

Figure 4.1: Openstack cloud conventional setup using vSwitch module.

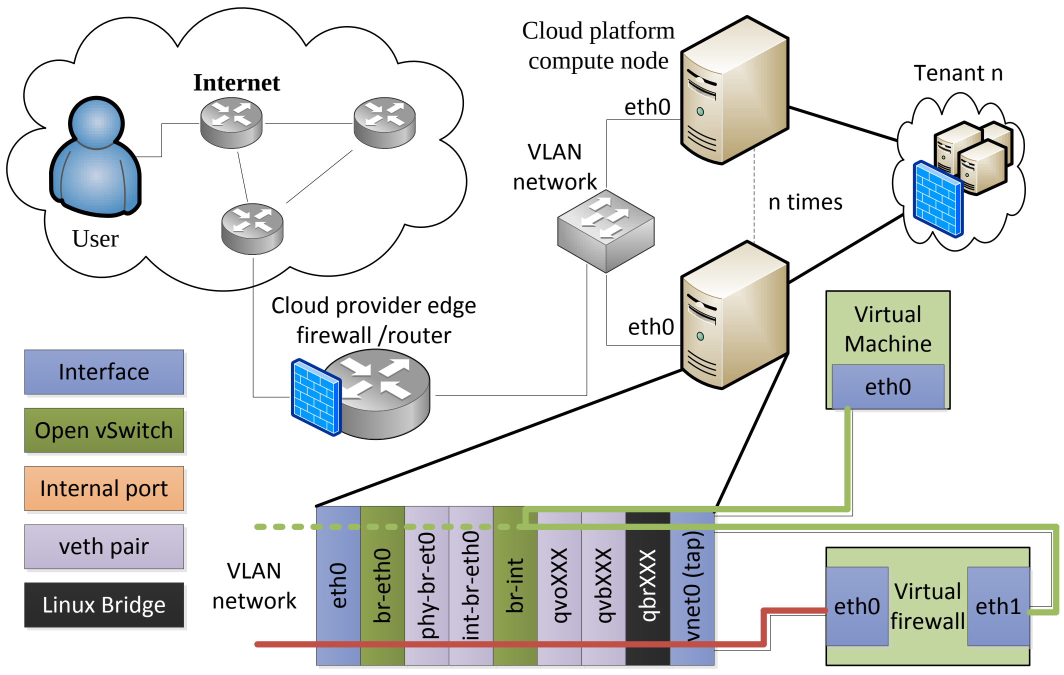

Figure 4.2: Openstack cloud network utilizing virtual firewall.

node now flows through the *br-int* bridge from one VM to another. Further, traffic between the virtual machines located in the different compute nodes flows via a physical switch plane and is separated by VLAN IDs. The solid green line represents traffic inside a node and the dotted line between the nodes.

VLAN function is necessary for building a safe and active environment. The global IP addresses need to be isolated in a specific VLAN and be connected only to a virtual firewall *eth0* interface. This way the virtual firewalls communicate directly with the cloud provider's edge router by using global IP addresses. This public connectivity is represented by the red line from the edge router to the compute node.

A major change like this provides countless opportunities. A global IP address can now be directly assigned to a single VM without a NAT and a firewall. Unfortunately it compromises security because firewalling should be done in each virtual server separately. Also modifications can be made at least to Openstack compute node networking modules. Since Linux bridge is used for gaining the *iptables* support, which is no longer needed, the Linux bridge and VETH pair can be removed from these modules. Such optimization is outside the scope of this thesis and should be done as a future work.

4.3 Security Software Selection

Firewall software for virtual machines needs to be carefully chosen to fulfill the requirements of the cloud platform and the tenant. These requirements include an open source platform, a NAT, a stateful firewall, a VPN solution, a REST API and a routing daemon. This is the basic setup each SECaaS solution should have.

Open source network operating systems can easily be found. Usually they are based either on some Linux or BSD distribution such as Okapi, Pfsense, ClearOS and Smoothwall [24, 25, 26, 27]. A NAT, a firewall and a VPN are included in all of these software distributions but they lack the API. A setup on Debian GNU/Linux was also considered but dismissed due to the time consuming and cumbersome inclusion of the REST API feature [28]. Vyatta, on the other hand, was found to contain the firewall, the routing daemon, the VPN and, most importantly, the REST API [29]. These features are only in the enterprise version of Vyatta that is updated by the vendor and is delivered with a cloud container template. Although costly, this reduces the time and effor of the cloud operator significantly. Additionally, the Amazon EC2 contains a template for Vyatta, allowing the private cloud to be extended in Amazon through a VPN [30].

Based on these characteristics, the firewall VM, also called as the security instance, chosen for this master's thesis was the Linux routing and firewall engine called Vyatta Enterprise vRouter by Brocade. It provides a multifunctional Linux distribution that can be used to secure internal network and to work as a VPN endpoint simultaneously. Vyatta's firewalling engine is done by using a Linux *iptables* package. Firewalling can be accomplished by using either per-interface policies or zone based policies [31]. The zone based policies (see Chapter 2) were chosen for this demonstration. Vyatta has PPTP, IPSec and OpenVPN features for site-to-site and remote access VPN connections. Routing in Vyatta is with the *Quagga* routing engine, which supports RIP, OSPF and BGP routing protocols. Vyatta has normally the NAT together with a firewall. In this thesis, Vyatta is placed between the Internet and a private network containing many cloud VMs to provide Internet connectivity for them. Thus the security instance operates as a gateway for all the cloud VMs running in a virtual private cloud.

4.4 Management Aspects

The enterprise version of Vyatta has the control options of Command Line Interface (CLI), Web UI and REST API. The most traditional of these three for firewall controlling is the CLI. It has been widely used in most enterprise level firewall appliances. A cloud platform requires a management interface for the tenant to make changes to the cloud infrastructure when necessary. Since Vyatta is to be a part of the cloud infrastructure, a combined cloud dashboard is needed. This is illustrated in Figure 4.3

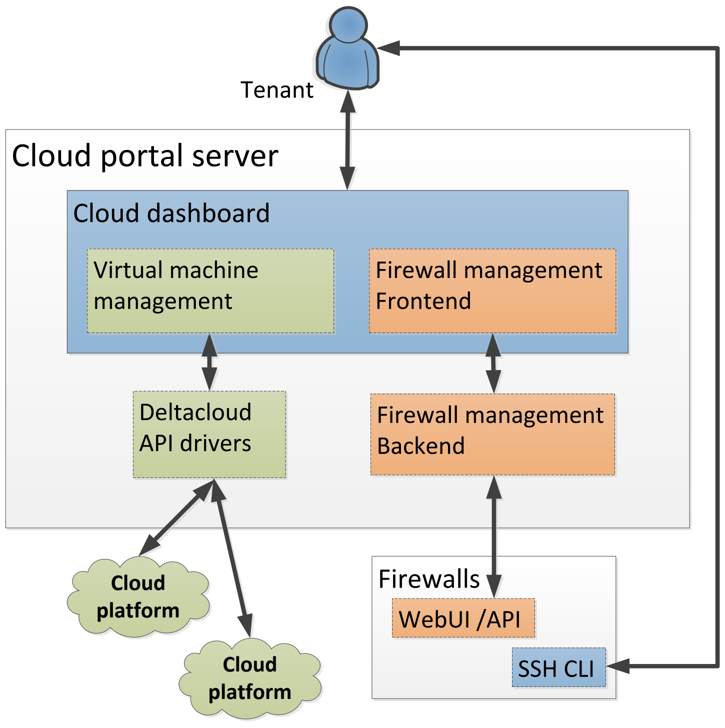

Figure 4.3: Centralized cloud platform and firewall management.

The cloud dashboard runs in a cloud portal server along with some other software components. The tenant connects either to the Cloud dashboard or directly to the firewall SSH CLI for configuration. The Cloud dashboard includes both the virtual machine management and the firewall management frontend features. The VM management communicates with the Deltacloud API drivers that further manage the cloud platforms. The firewall management frontend commands its backend that is then responsible for communicating with the firewalls. This section takes a brief look of the firewall management through either the cloud dashboard or the firewall's own WebUI and CLI.

4.4.1 Cloud Portal Server and Dashboard

Management of the security instance (Virtual firewall) is preferably done by a customized interface called a cloud dashboard. It is not meant to fully replace the firewall's own CLI or WebUI, but to make management more user-friendly and centralized. The dashboard provides configurations for most basic features as well as state information for all the firewalls managed by a specific user account.

The firewall management backend uses REST API to execute operational and configuration mode commands (Chapter 5.3) in the firewall (Figure 4.3). The REST API resembles a remote CLI, which provides multifunctionality. It thus offers a customizable controlling environment for the tenants based on their needs.

The firewall management frontend can also be called as a customized webUI. It exists especially for creating and editing general configurations, NAT rules and firewall policies. In addition to these, the frontend also provides information about the firewall status, for example firewall policies and routing table. Log files stored in */var/log/* are unreachable for the REST API as there is no appropriate command to read them. More experienced users, however, can reach them via CLI.

For the general configuration, the management frontend provides methods for modifying details such as the hostname. The firewall policy configuration is wider: the user may select a zone, a rule number, IP addressing and port details as well as a protocol. Removing a firewall rule means removing all the related settings. An individual setting is bound to the rule and thus cannot be separately removed. A NAT rule contains the rule number, IP addressing and port details together with the protocol field. The NAT rules may be removed only one by one. If the user requires more configurable objects such as routing or VPN, CLI should be used via SSH to reach them.

The central management simplifies the firewall configuration especially when the user has a number of firewalls. The end user is no longer required to make a separate CLI or WebUI connection to individual firewalls from their internal networks. The dashboard further provides parallel configuration for the firewalls. The management server can, for example, send a configuration command any number of times.

Therefore sending a new Network Time Protocol (NTP) server information to all the firewalls can now be done with only one command.

4.4.2 CLI and WebUI

A Command Line Interface (CLI) is the basic configuration interface for a router or a firewall device. Vyatta CLI is accessible from the local console of the firewall instance or via the SSH, that provides an another remote configuration method for the user. The virtual firewall including the host Linux, can be controlled by the CLI since it is set up on top of a Linux shell. The CLI is thus always available when the user has access to the shell. The local console authentication always reqires a username and a password whereas the SSH also supports a public key.

The CLI on Vyatta is similar to that of the Juniper JunOS CLI. Both contain the same basic commands and the configuration tree structure. The CLI is separated in to the operational and the configuration mode. The operational mode provides commands for viewing the firewall state whereas the configuration mode is used for editing the system configuration parameters. The aforementioned modes of the REST API follow these same principles.

The WebUI is another friendly way to configure the firewall. It is alike a regular ADSL firewall router WebUI. The WebUI operates on top of the *lighttpd* web server daemon and thus is also responsible for the message exchange of the REST API.

5 SECURITY INSTANCE IMPLEMENTATION

The SECaaS has been developed to be easily deployable by the end-user. This is accomplished by an automated firewall initiation and bonding of the firewall with the management dashboard. This chapter describes how the SECaaS can be used in common setups and how to deploy it with the Openstack or with an another cloud environment. The firewall VM's communication with the management server is also discussed. Furthermore, some answers concerning the user limitations and system security are provided.

5.1 The Common Setup and its Difficulties

When a company plans to outsource servers, a question may arise about building a secure connection between the company internal network and the cloud server network. A VPN is often chosen to provide such a secure multifunction tunnel. However, the Openstack Neutron for instance does not provide VPN tunneling with the Grizzly release and thus a virtual firewall that supports VPN tunnels, such as Vyatta, is required. Figure 5.1 shows a common way to extend the company internal network to a cloud network.

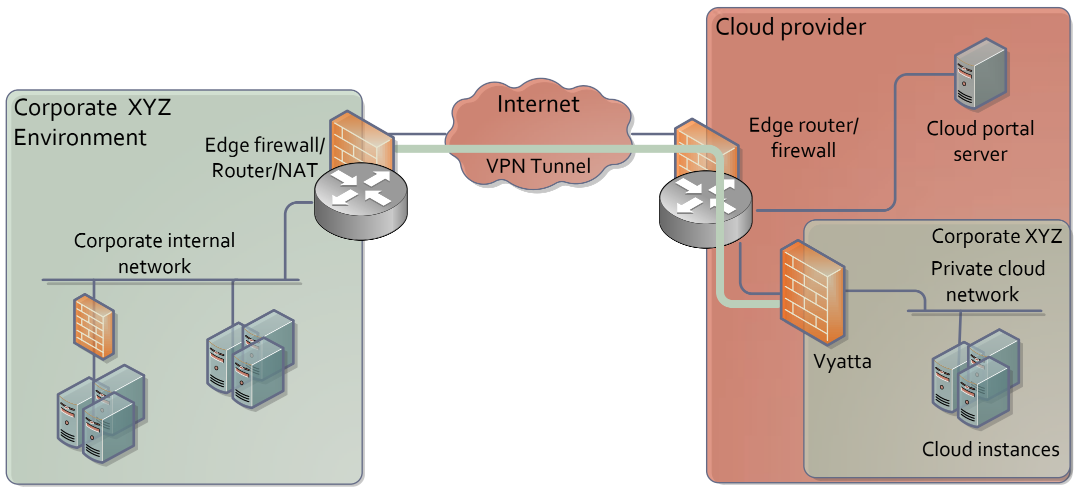
Corporate XYZ Environment
Edge firewall/ Router/NAT
Corporate internal network
Internet
VPN Tunnel
Cloud provider
Edge router/ firewall
Cloud portal server
Corporate XYZ Private cloud network
Vyatta
Cloud instances

Figure 5.1: Internal network extended in a private cloud.

The Corporate XYZ Environment in the figure represents a common enterprise level internal network and the Cloud Provider the cloud hosting company. Hosts owned by a certain company are placed in the cloud in a network segment called the *private cloud*. All hosts of a private cloud share the same subnet and communicate with the Internet through the same router. The multifunctional virtual firewall is placed between the Internet and the private cloud to protect servers as well as to operate as a gateway to the Internet and as a VPN endpoint to the corporate internal network.

A cloud network requires a NAT to be able to save IP address resources when a private cloud network is set up. However, a NAT behind a NAT can cause trouble. The Neutron allocates private IPv4 addresses for the VMs and provides the public IP addresses as floating IPs using 1:1 NAT. The VM firewall also implements NAT and thus overlapping IP addressing may occur. As described in Section 4.2, the Neutron networking module in Openstack cloud needs to be removed to ensure proper network functionality.

Figure 5.2 shows firewalling and relationships of common zones. There are four different zones: external, internal, local and VPN. The external zone is connected to an untrusted network (the Internet) and the internal zone to a trusted network (private cloud). The local zone is meant for traffic to the Vyatta firewall. The VPN zone operates as a zone for networks linked by VPN connections. The end user may modify the firewall rules using *from-zone* and *to-zone* statements. Each zone contains also a non-configurable *deny all* rule. Its number is 10000, being the last rule. Other rules (1-9999) are freely editable by the user.

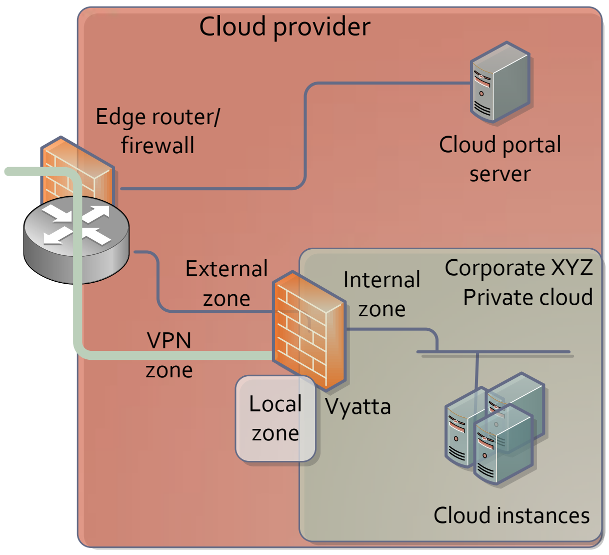

Figure 5.2: Vyatta firewall zone assignment for a private cloud.

The cloud portal server is located in the cloud provider network and the users are able to control their firewall VMs through the cloud dashboard operating in this server. The clients connect to the cloud portal server which then communicates with the firewall VMs. The message structure used for communication between the cloud portal server and a firewall is explained in Section 5.3.

The cloud portal server requires connectivity to the Vyatta REST API from the external zone and thus a proper firewall rule needs to be created. This rule is called a *management permit rule* and applied from the external zone to the local zone. No other rule should block it.Therefore the management permit rule is set as the first rule. The rule provides security by allowing connections only from the cloud portal server IP address. The destination port and the protocol should also be set to make the rule stricter. An example ruleset is shown in Table 5.1.

Rule #	Src. addr.	Dest. addr.	Src. port	Dest. port	Protocol	action
1	v.x.y.z	0.0.0.0	0	56443	TCP	PERMIT
10000	0.0.0.0	0.0.0.0	0	0	ANY	DROP

Table 5.1: Example firewall ruleset from External zone to Local zone.

The VPN tunnel from the company internal network is terminated in the Vyatta firewall and so another firewall rule needs to be created for it. The protocol and the port follow the settings of VPN software. The firewall rule for the VPN connection is set up between the external zone and the local zone. However, data inside the tunnel flows from the internal zone to the VPN zone and vice versa.

IPsec is commonly supported in firewall appliances and thus often utilized in networks like that in Figure 5.1. The cloud provider network may prohibit the negotiation of the VPN tunnel between the private cloud and the company internal network. The IPsec faces this problem when the connection is made through a NAT. The IPsec Authentication Header (AH) invalidates the authentication phase, leading to malfunction [32]. Internet Engineering Task Force (IETF) has tried to approach this with a NAT traversal (NAT-T) by forcing the IPsec to use the UDP in the port 4500 [33]. This however has no effect in the Openstack because the inbuilt security group firewall does not allow even the Encapsulated Security Payload (ESP) protocol, necessary to the IPsec, to pass through. Therefore the modification to remove the NAT (Section 4.2) is needed.

Another approach is to use other technology like SSL VPN for tunneling. OpenVPN is an open-source SSL-based VPN product that operates on top of either TCP or UDP protocol in a chosen port, making it compatible with the NAT. The firewalling in the Openstack allows rule creating for corresponding traffic. Using SSL VPN eliminates the need for modifications of the Openstack Neutron, yet achieving the VPN functionality.

5.2 Instance Deployment Phase

An automated firewall instance deployment is achieved when a cloud platform is independently able to modify firewall instances and a cloud portal server. Without this the SECaaS requires a lot of manual configuration. Launching an instance to the cloud is simple and setting up the connection between the management server and the firewall requires little extra effort. This section covers the automatic configuration of the cloud portal server and the firewall instance. Also manual configuration is discussed.

5.2.1 Deployment in an Automated Cloud Environment

An automated firewall deployment in SECaaS only needs a cloud platform with an ability to run custom scripts. The Openstack is one of these platforms and thus was chosen as the demonstration platform. The deployment phases are shown in Figure 5.3. First, the end user launches a new firewall instance to the cloud. Then the cloud dashboard provides authentication and other metadata information for the firewall management backend and the firewall instance. Finally an initialization script sets up the firewall and applies default configurations to it.

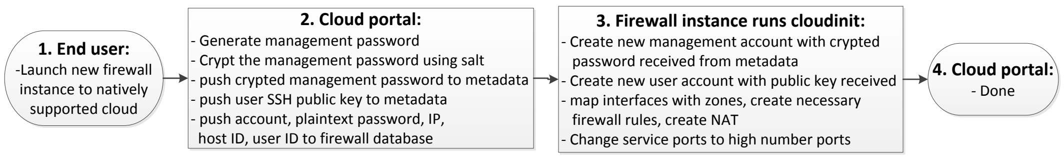
1. End user:
-Launch new firewall instance to natively supported cloud
2. Cloud portal:
- Generate management password
- Crypt the management password using salt
- push crypted management password to metadata
- push user SSH public key to metadata
- push account, plaintext password, IP, host ID, user ID to firewall database
3. Firewall instance runs cloudinit:
- Create new management account with crypted password received from metadata
- Create new user account with public key received
- map interfaces with zones, create necessary firewall rules, create NAT
- Change service ports to high number ports
4. Cloud portal:
- Done

Figure 5.3: Deployment phases in an automated cloud environment.

When launching a firewall the end user has to take care of a proper interface assignment and the choice of a right flavor to match the needs of the instance. The first Ethernet interface (*eth0*) should be placed in an untrusted network and the second (*eth1*) in a trusted one. A public SSH key should also be assigned to gain SSH connectivity to the firewall.

The password of the firewall management account is generated and stored in plaintext in the cloud dashboard because of the REST API, then encrypted and delivered to the firewall. The encrypted password is transferred to the firewall using cloud instance metadata information. Metadata is generated for every created VM and saved to a cloud platform, and it remains stored as long as the instance exists. As the Vyatta firewall is based on Linux, the password is encrypted with a salt value and stored in a *shadow password* file compatible form. The end user SSH public key is delivered to the firewall instance along with the password. The cloud dashboard also stores instance information such as the account details, the IP address, the firewall machine ID and the user ID in the firewall management backend database.

The *cloudinit* is a script in the client machine that runs default configurations during the first boot. Vyatta is based on a CLI and a specific configuration tree structure, thus the changes done by the cloudinit have to be committed to */config/config.boot* -file. The cloudinit script reads the metadata to set up the management account and the SSH public key for the firewall VM.

Next the cloudinit script assigns IP addresses found in the metadata to the instance. The script maps the *eth0* to an *untrusted* zone and *eth1* to a *trusted* zone. Further, the script enables stateful packet filtering, creates a basic outbound NAT rule as well as basic firewall rules, and binds them to interfaces. An example setup using a CLI for Vyatta is displayed in Appendix C.

The basic firewall rules allow all connections from a trusted network to an untrusted one and deny all opposite traffic. The ruleset allows the incoming management connections (port 56443) from the *untrusted* network to the *local* zone. It also allows incoming connections from the *untrusted* zone to port 56022 for SSH connectivity. This leaves the ports 443 (HTTPS) and 22 (SSH) free for web and SSH services in the cloud VMs. Also security is enhanced because these common service ports are often targeted by automated vulnerability scanners.

5.2.2 Manual Configuration

The cloud dashboard also has the ability to manually add firewall management information to the backend. Some cloud platforms (e.g. VMWare vCloud) do not support running a custom script when an instance is launched. In these cases automation cannot be done, making manual firewall insertion necessary. Also the customer may run Vyatta in the EC2 or in a server without virtualization. Cloud dashboard can be used for controlling Vyatta firewall regardless of their location.

The manual steps for adding a firewall VM are shown in Figure 5.4. First the instance needs to be launched and configured to operate on its own. When network connectivity and account information are established the firewall is ready to be added under the control of the cloud dashboard.

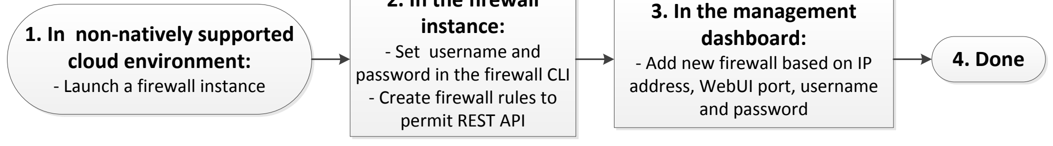
1. In non-natively supported cloud environment:
- Launch a firewall instance
2. In the firewall instance:
- Set username and password in the firewall CLI
- Create firewall rules to permit REST API
3. In the management dashboard:
- Add new firewall based on IP address, WebUI port, username and password
4. Done

Figure 5.4: Adding firewall to central management system.

The setup phase of the firewall instance is done manually by the VM administrator after the launch. A username and a password need to be set in the CLI together with the firewall rules to allow connections from the cloud portal server. The WebUI port for the REST API should be changed in the *lighttpd* daemon configuration file.

The cloud dashboard needs the global IP address and the port of the firewall VM along with a username and a password to operate with REST API queries. When these have been added to the firewall dashboard, the performs a few queries to the firewall to verify the connection. These queries request for details such as a hostname and an uptime. When these steps have been completed the firewall is ready for use.

5.3 Centralized Management

The central firewall management of the cloud portal consists of frontend and backend software. The frontend is a part of the cloud dashboard whereas the backend is independent software. These two can be distributed but more commonly they operate on the same server.

The firewall management frontend is set up with two different web programming languages, HTML and Javascript. The HTML provides the page layout and Javascript is needed for the web forms that are necessary for user interaction. The user fills in information, it is gathered from the Javascript web forms, and sent to the backend PHP code. When the frontend and the backend are on the same server, an HTTP daemon that understands PHP language is essential. PHP performs well as a backend because it is stateless and transparent to the end user. The PHP backend requires a firewall database that contains information on all the firewalls with their IP addresses and authentication details. The firewall database can either be a text file or a fully featured SQL database. A sequence diagram containing the client browser, the frontend, the backend, the database and the firewall which illustrates the interactions between the client, the dashboard and the firewall is displayed in Figure 5.5.

The sequence diagram in the figure begins with the user authentication in the frontend. It then delivers the authentication information to the backend which in turn makes a query to the database, validating the user and returning information about the manageable firewalls. The backend PHP stores this information for the duration of the session, keeping the authentication valid whether the user requests for fetching information or changing or removing configuration. After the user control is released from the dashboard, the PHP session is destroyed and re-authentication is required.

After a successful authentication, the frontend expects the user to pick a number

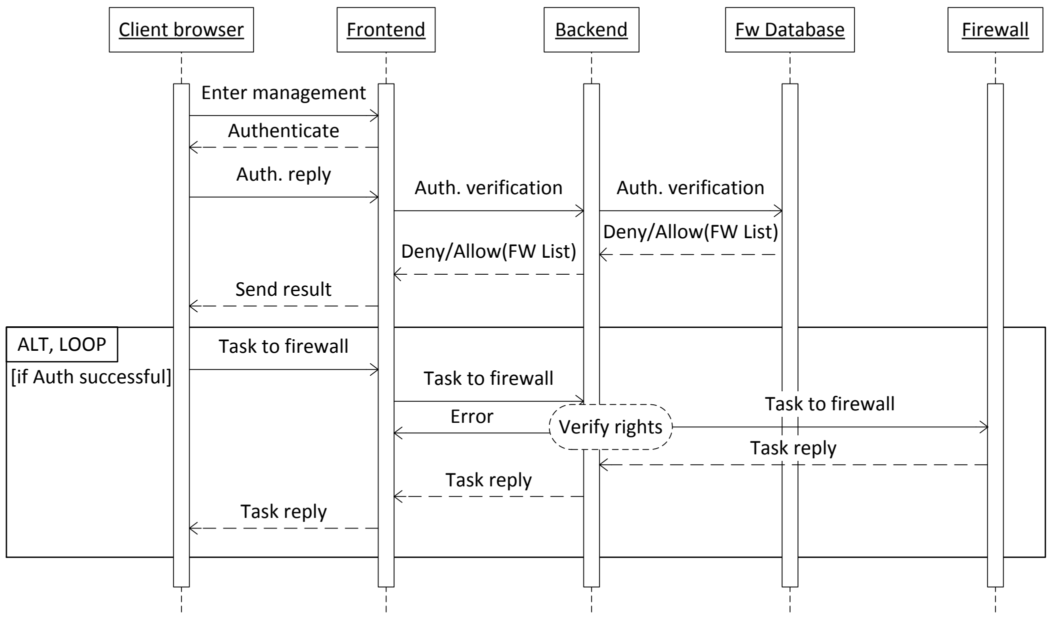

Figure 5.5: Cloud dashboard firewall authentication and operation sequence diagram.

of firewalls to edit and a task to perform. The backend verifies the firewall selection against the user information that was received from the database during the authentication. Tampering with the target firewall information is noticed and illegal requests are prohibited. When the user input is valid the task is sent to all chosen firewalls. The backend receives a reply message from each firewall and delivers them to the frontend to be displayed to the user.

The backend validates the user input. The frontend delivers the user input from Javascript to the PHP backend using *$_POST* -method. These messages are user generated so the input validity needs to be verified. The validation in the backend is done by building a configuration tree using arrays and then verifying the user input against it. This procedure protects the firewall from faulty commands.

The backend also protects the firewall from removing or overrunning any vital configurations. The protection includes the basic outbound NAT and the management permit rule. However, the CLI via local console or SSH can be used to remove any of these configurations. The user limiting balances between protecting the firewall and not totally restricting the user.

The PHP backend communicates with the firewall using REST API. The REST API operates with HTTPS messages that can be built using *cURL*, for example. The cURL is a library for URL transfer that provides multifunctional message creation for web services and thus performs well with REST API. A cURL message requires a header containing authentication, application and HTTP request information together with a destination address. A valid cURL query from a Linux shell to the Vyatta REST API requesting active resource information is as follows:

```
curl -k -i -u username:password -H "content-length:0" -H "Accept:
 application/json" -X GET https://10.11.11.1/rest/op
```

The cURL is based on flags. The *-k* flag stands for *insecure* connection where the server certificate is not verified. With the *-i* flag, the output shows also the usually hidden HTTP header. The *-u* flag assigns authentication information to the REST API query. The *-H* flag sets the HTTP header content-length to zero and specifies the data to be JSON. The *-X* builds a custom query, for example *get* or *delete*. The same message can be written in PHP as follows:

```
$headers = array ("Accept: application/json", "content-length:0");
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "https://10.11.11.1/rest/op");
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_USERPWD, "username:password" );
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
// curl_setopt($ch, CURLOPT_POST, true);
// curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
$output = curl_exec($ch);            // Firewall answer
curl_close($ch);
```

The PHP code first creates a handle named *$ch* that is then used in the query. Second, details like address and authentication information are added. The line *CURLOPT_SSL_VERIFYHOST* sets the query to avoid certificate verification. Third, plaintext username and password are inserted. A query is executed with given parameters and the response gets stored in the variable *$output*. Finally, the resource is released using the *curl_close* command. Other messages (Ch 3.3.1) can be sent using the same structure by uncommenting either the *CURLOPT_POST* or *CURLOPT_CUSTOMREQUEST* field.

A REST API sequence diagram displaying information fetching from a firewall is shown in Figure 5.6. The firewall management backend makes a *post* query to the instance containing the requested information resource identifier in the URI. For example, the full URI to receive IP routing information is:

```
https://ip.address.of.firewall:56443/rest/op/show/ip/route
```

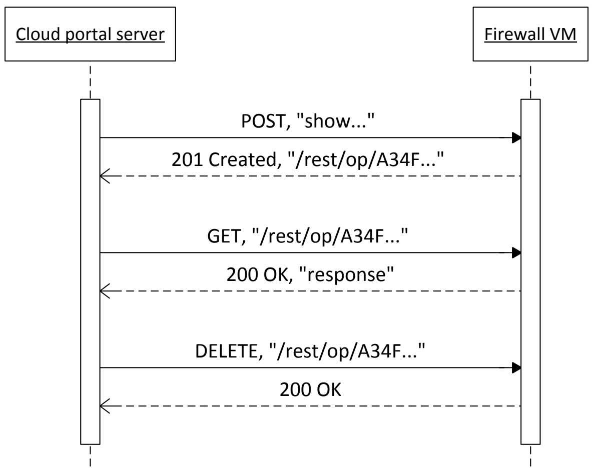

Figure 5.6: Sequence diagram of firewall information fetching message exchange.

The firewall responds to the *post* message using *201 Created* message containing a resource ID. The backend fetches the real data from there using a *get* message. A small delay is needed between the *post* and the *get* messages to allow the firewall time to collect and process requested information. The delay verified during prototype experiments to be long enough for every task is two seconds. The Vyatta's remote access API guide claims the *delete* message to be useless because supposedly the firewall automatically removes the reserved resource [34]. However, it turned out during these experiments that this does not take place necessitating the use of manual delete message.

Configuring a firewall requires a more complicated message exchange. A common example is illustrated in Figure 5.7. The configuration phases are the same whether using REST API or CLI: entering the configuration mode, adding configurations, and applying the *commit*, *save* and *exit* -commands. During the first phase a *post* message representing the *configure* command is sent to the firewall that again replies using *201 Created* containing the resource ID. In the second phase, the backend loops through the *put* messages until there are no more configuration parameters to be sent. After this task, a *commit* is sent to the firewall using a *post* message. The firewall answers with *200 OK* containing the *commit* response. In case the *commit* is not approved, firewall management stores the return message, displays it to the user and sends a *delete* message to remove the resource reserved for the configuration session. Otherwise backend sends a *save* message to the firewall before the *delete* message.

The sequence diagram in Figure 5.8 displays message exchange when the user

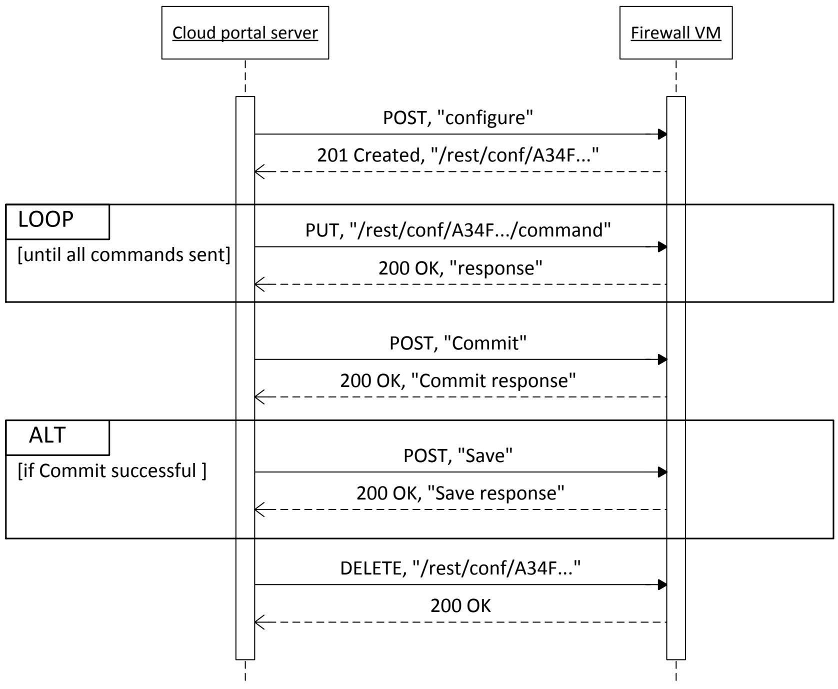

Figure 5.7: Sequence diagram of firewall configuration message exchange.

requires removal of configurations from the firewall. This is very similar to the configuration insert phase. However, only one *put* message containing a *delete* task is required. The configuration is hierarchical so an entire configuration tree branch can be removed at once. *Commit* and *save* commands are sent using a *post* message to apply and save the configuration and they are followed by a *delete* to remove the allocated resource in the firewall.

5.4 Security Aspects

The cloud portal server and all virtualized firewalls can be reached from the Internet. Thus the users can connect to the cloud dashboard from anywhere in the Internet and the cloud portal server in turn can reach the firewalls. To be more exact, by default the WebUI and the REST API of the firewalls are only connectable from the cloud portal server's IP address. This protects the firewall management and the cloud dashboard operates as an application proxy firewall. However, CLI can be used to create a new REST API firewall rule to allow connections from a wider IP address range.

The Vyatta firewall contains in this demonstration two user accounts: a separate configuration account for the cloud dashboard and a user account for the end user.

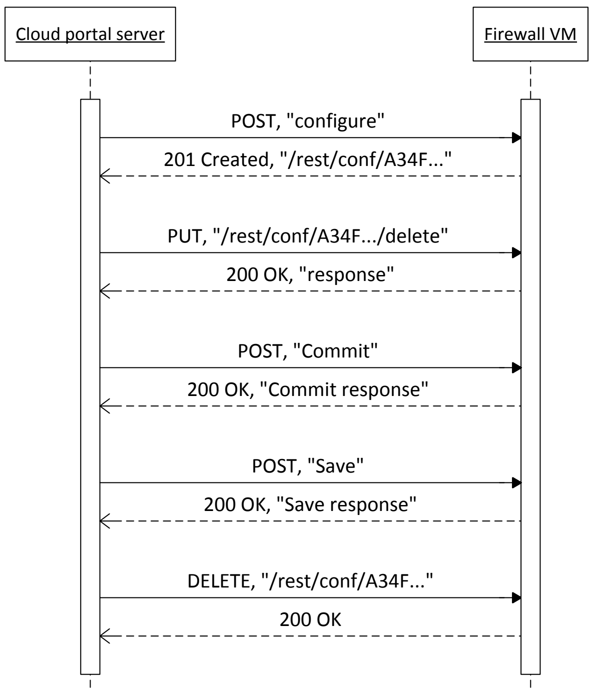

Figure 5.8: Sequence diagram of firewall configuration removal message exchange.

The password for the former is 20 characters long and stored non-encrypted in the backend database in the cloud portal server, which is safe as long as the database remains intact. This is also a weakness that can easily compromise security. The passwords are stored encrypted in the firewall and the cloud metadata thus remain very secure.

The end user account is generated in the cloudinit phase and an SSH private key is added to it. The account is created using a commonly known password to enable login through the local shell, the SSH and the WebUI. The SSH only allows logins with a private key, making it safe as nobody can enter the firewall via SSH without this key. The firewalling rule protects the firewall from WebUI and REST API connections from the Internet excluding the cloud portal server. An additional rule to allow these connections can be created and thus it is highly recommended that the end user should change the password right after launching the instance.

6 DISCUSSION

This thesis demonstrates how a firewall can be virtualized to an IaaS cloud and how a central management service is used to control a number of firewalls simultaneously. The virtual firewall offers an on-demand service and extends beyond regular firewalling functions. This is called Security-as-a-Service (SECaaS).

The prototype implementation was successfull as a proof-of-concept to offer knowledge on the SECaaS management and the Openstack Neutron. However, it is not yet ready to be used in production. Convenient use would require more features, and a security review should be performed.

The firewall, the VPN and the routing features of the SECaaS are likely to be replaced in the near future by some built-in software of the IaaS cloud. A milestone is already accomplished by the new release of the Openstack cloud platform (Havana) with the Neutron network plugin that uses multiple service plugins such as Load-Balancer-as-a-Service (LBaaS) and Firewall-as-a-Service (FWaaS). The LBaaS is already in the Grizzly release and the FWaaS is being developed for the Neutron. The Openstack networking currently contains security groups for firewalling. The FWaaS includes stateful zone based firewalling and also next-generation firewalling is being considered. [35]

When the FWaaS plugin is in use the Vyatta VM firewalling becomes unnecessary. This compromises the required VPN feature, giving rise to another new Neutron module called VPNaaS, VPN-as-a-Service. The VPNaaS, however, only supports IPsec site-to-site tunneling which, on the other hand, usually gets the task done. Also, IPsec is standardized technology, enabling cross-platform setups. The use of FWaaS and VPNaaS means in practice that the cloud modifications required by Vyatta, such as the network environment and the firewall management in the central management portal, are no longer needed. [35]

The virtualized firewall is included in the Software Defined Networking (SDN) and the Network Functions Virtualization (NFV). The SDN specifies how network devices are managed by software. The main principles in the SDN are to separate the control and data planes and to use a flow based traffic forwarding. The control plane is for management whereas the data plane is for operating with the traffic filtering, forwarding and routing. Further, the control plane management can be centralized to cover all devices in a data center. This automates network management tasks.

This enables rapid network deployment and provisioning through the data center. The SDN controller can handle either virtual or physical devices, which makes the environment highly elastic. Therefore The SDN is a cunning research topic for providing on-demand data center services. [36]

The term NFV is often related with the SDN. It is complementary to the SDN but neither depends on the other. The NFV consists of a number of network functions such as the firewall, the NAT and the IDS just to name a few, and thus attempts to transfer hardware appliances to hypervisors or the cloud. Furthermore, the SDN and NFV are combined in the Software Defined Data Center (SDDC), a data center which is elastic and highly virtualized. All the SDDC services are *-as-a-service* and as such deployed on-demand. Thus the time-to-market is minimized. [37, 38]

The commercial and the open-source software costs, benefits and time consumption should be compared to find out the best possible outcome for cloud computing. However, the inevitable requirements such as customized central cloud portal may elevate the costs unexpectedly. What comes to the pros and cons of virtualization, the NFV and SDN reduce the time required for configuring and cabling devices, but these are likely to need new hardware and thus cause extra expenses.

7 CONCLUSIONS

This Master's Thesis introduced a Security-as-a-Service concept that contains a central management dashboard and an IaaS cloud supported virtual firewall. Vyatta was chosen as firewall software due to its properties (the firewall, the VPN, the routing daemon and the REST API). The Vyatta firewall security policies were considered comprehensive and reliable thanks to the stateful inspection. Vyatta has also been proven to handle complicated routing and VPN scenarios. OpenVPN and IPsec VPN both perform well and site-to-site tunneling can be done with any participant supporting either of these techniques.

The most commonly used firewall features can be configured from a central management dashboard. The key to the central management of Vyatta is the REST API. It also allows the centralized management to control many firewalls in parallel. Thus the dashboard is able to send a specific command theoretically to any number of firewalls simultaneously. In practice, this is limited to a few thousand depending on the available hardware and software resources.

A severe discovery was made with the cloud network. Cloud platform providers often rely on a separate network component, like Openstack with Neutron. The cloud platform software and even the whole cloud network needs to be modified to enhance the firewall operation. Without modifications the network traffic may cause extra load on the cloud as well as difficulties with the VPN function.

The SECaaS concept is likely to extend from the mere cloud to the entire data center. This would mean combining the tenant's physical network in the data center with the cloud's virtual network. Therefore a single firewall VM could protect both the virtual and the hardware servers of a data center. Further, the firewall management is transferred from the administrators to the end users.

The customers will require more and more on-demand services in the era of cloud computing. SECaaS gives rise to the future data center model whereas the virtual firewall heads for NFV. The multi-platform cloud's security and usability especially benefit from the SECaaS. These can however be achieved cost-effectively in a single platform cloud as well. The Openstack cloud has made progress in this with its new release but still does not have any ultimate solution. The race for the best cloud platform is going on, which elevates the requirements for the benefit of the tenant. The need for more virtualized network functions and **-as-a-Service* remains.

REFERENCES

[1] D. Liu, S. Miller, M. Lucas, A. Singh, J. Davis, *et al.*, *Firewall policies and VPN configurations.* Syngress, 2006.

[2] C. K. Wack John and P. Jamie, "Guidelines on firewalls and firewall policy," tech. rep., DTIC Document, 2002.

[3] R. Di Pietro and L. V. Mancini, *Intrusion detection systems.* Springer, 2008.

[4] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST Special Publication*, vol. 800, no. 2007, p. 94, 2007.

[5] I. Dubrawsky, "Firewall evolution - deep packet inspection," *Security Focus*, vol. 29, 2003.

[6] J. Pescatore and G. Young, "Defining the next-generation firewall," *Gartner RAS Core Research Note*, 2009.

[7] P. Lindstrom and R. Director, "Intrusion prevention systems (ips): Next generation firewalls," *Spire Security*, 2004.

[8] R. Y. Ameen and A. Y. Hamo, "Survey of server virtualization," *arXiv preprint arXiv:1304.3557*, 2013.

[9] J. Matthews, T. Garfinkel, C. Hoff, and J. Wheeler, "Virtual machine contracts for datacenter and cloud computing environments," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC '09, (New York, NY, USA), pp. 25–30, ACM, 2009.

[10] V. Seppänen, "Elastic build system in a hybrid cloud environment," Master's thesis, Tampere University of Technology, 2011.

[11] V. Choudhary, "Software as a service: Implications for investment in software development," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pp. 209a–209a, 2007.

[12] G. Lawton, "Developing software online with platform-as-a-service technology," *Computer*, vol. 41, no. 6, pp. 13–15, 2008.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[14] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[15] Openstack cloud computing software [WWW]. [Retrieved 2013-09-05]. `http://www.openstack.org`.

[16] Openstack conceptual architecture [WWW]. [Retrieved 2013-09-05]. `http://docs.openstack.org/grizzly/openstack-compute/admin/content/conceptual-architecture.html`.

[17] E. Wilde, "Putting things to rest," *Recent Work, School of Information, UC Berkeley*, 2007.

[18] Deltacloud system overview [WWW]. [Retrieved 2013-09-05]. `http://deltacloud.apache.org/about.html`.

[19] Amazon Elastic Compute Cloud [WWW]. [Retrieved 2013-09-05]. `http://aws.amazon.com/ec2/`.

[20] A. Corradi, M. Fanelli, and L. Foschini, "Vm consolidation: a real case based on openstack cloud," *Future Generation Computer Systems*, 2012.

[21] Openstack networking service, security guide [WWW]. [Retrieved 2013-09-10]. `http://docs.openstack.org/trunk/openstack-security/content/ch032_networking-best-practices.html`.

[22] Openstack documentation, open vSwitch, OpenStack Networking Administration Guide [WWW]. [Retrieved 2013-09-10]. `http://docs.openstack.org/trunk/openstack-network/admin/content/under_the_hood_openvswitch.html`.

[23] Openstack Networking Neutron page [WWW]. [Retrieved 2013-10-15]. `https://wiki.openstack.org/wiki/Neutron`.

[24] Okapi router feature page [WWW]. [Retrieved 2013-10-24]. `http://okapirouter.net/index.php?option=com_content&view=article&id=222&Itemid=19`.

[25] ClearOS Community edition feature page [WWW]. [Retrieved 2013-10-24]. `http://www.clearfoundation.com/Software/overview.html`.

[26] Smoothwall Express community version feature page [WWW]. [Retrieved 2013-10-24]. `http://www.smoothwall.org/about/express-feature-list/`.

[27] Pfsense Opensource Firewall Distribution feature page [WWW]. [Retrieved 2013-10-24]. http://www.pfsense.org/index.php@option=com_content&task=view&id=40&Itemid=43.html.

[28] Debian GNU/Linux home page [WWW]. [Retrieved 2013-10-24]. http://www.debian.org.

[29] Vyatta firewall distribution feature page. Brocade company [WWW]. [Retrieved 2013-10-24]. http://www.vyatta.com/product/vyatta-network-os/open-source-vs-enterprise.

[30] Vyatta Network OS for Amazon. Brocade company web page [WWW]. [Retrieved 2013-10-24]. http://www.vyatta.com/product/vyatta-network-os/amazon.

[31] Vyatta, INC. Vyatta System, Firewall Reference guide [WWW]. [Retrieved 2013-09-15]. http://www.vyatta.com/downloads/documentation/VC6.5/Vyatta-Firewall_6.5R3_v02.pdf.

[32] B. Aboba and W. Dixon, "IPsec-Network Address Translation (NAT) Compatibility Requirements." RFC 3715 (Informational), IETF, Mar. 2004.

[33] T. Kivinen, B. Swander, A. Huttunen, and V. Volpe, "Negotiation of NAT-Traversal in the IKE." RFC 3947 (Proposed Standard), IETF, Jan. 2005.

[34] Vyatta, INC. Vyatta System, Remote access API 2.0 Reference guide [WWW]. [Retrieved 2013-10-02]. http://www.vyatta.com/downloads/documentation/VC6.5/Vyatta-RemoteAccessAPI2.0_6.5R1_v01.pdf.

[35] Openstack Networking API 2.0 Reference [WWW]. [Retrieved 2013-10-28]. http://docs.openstack.org/api/openstack-network/2.0/content/ch_preface.html.

[36] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks.* " O'Reilly Media, Inc.", 2013.

[37] Network Function Virtualization (NFV), White paper. ETSI [WWW]. [Retrieved 2013-10-31]. http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[38] Horizontal Integration - Unlocking the cloud stack. Nixu Software [WWW]. [Retrieved 2013-10-31]. http://www.nixusoftware.com/unlocking_the_cloud_stack_whitepaper.pdf.

APPENDIX A: CISCO NOTATION ACL EXAMPLE

```
access-list 101 permit icmp any any echo-reply
access-list 101 permit icmp any any ttl-exceeded
access-list 101 permit icmp any any unreachable
access-list 101 permit udp any any eq domain
access-list 101 permit udp any eq domain any
access-list 101 permit tcp any any eq domain
access-list 101 permit tcp any eq domain any
access-list 101 permit tcp any any eq www
access-list 101 permit udp any eq bootps any eq bootpc
access-list 101 deny   ip any any
```

APPENDIX B: SMTP APPLICATION PROXY RULESET EXAMPLE

```
# Helo checks
mail.tld REJECT You are not mail.tld
mx1.mail.tld REJECT You are not mx2.mail.tld
mx2.mail.tld REJECT You are not mx2.mail.tld
localhost REJECT You are not me

# Recipient checks
/^\@/ 550 Invalid address format.
/[!%\@].*\@/ 550 This server disallows weird address syntax.

# Sender checks
mail.tld 554 Spam not tolerated here
```

APPENDIX C: VYATTA FIREWALL EXAMPLE PRE-SETUP

```
# Make the firewall stateful as default
set firewall state-policy established action accept
set firewall state-policy related action accept
set firewall state-policy invalid action deny

# Create a NAT to firewall
set nat source rule 10 outbound-interface 'eth0'
set nat source rule 10 source address '192.168.10.0/24'
set nat source rule 10 translation address 'masquerade'

# Assing interfaces to zones, create rule sets and bind them to zones
set zone-policy zone Untrust interface eth0
set zone-policy zone Trust interface eth1
set firewall name OUTBOUND rule 100 action accept
set firewall name INBOUND rule 9999 action deny
set zone-policy zone Untrust from zone Trust firewall name OUTBOUND
set zone-policy zone Trust from zone Untrust firewall name INBOUND

# Also the local-zone requires firewalling
set zone-policy zone LocalZone local-zone
set firewall name L_From_T rule 1 action accept
set firewall name L_From_U rule 100 action accept
set firewall name L_From_U rule 100 destination port 56022
set firewall name L_From_U rule 101 action accept
set firewall name L_From_U rule 101 destination port 56443
set firewall name L_From_U rule 101 source address 192.168.24.24
set firewall name L_From_U rule 9999 action deny
set zone-policy zone LocalZone from Trust firewall name L_From_T
set zone-policy zone LocalZone from Untrust firewall name L_From_U

# Create an account for the management server or the user
set system login user configureaccount authentication encrypted-password
   '$1$r/s97RG3$MRpnD8CoOOGBwO2BduyLz.'
set system login user configureaccount authentication public-keys
   user@hostname key 'key'
set system login user configureaccount authentication public-keys
   user@hostname type 'ssh-rsa'
set system login user configureaccount level 'admin'
```