DETECTING CLOUD VIRTUAL NETWORK ISOLATION SECURITY FOR DATA LEAKAGE

Haifa Mohammed Al Nasseri

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews

2019

Detecting Cloud Virtual Network Isolation Security for Data Leakage

Haifa Mohamed Al Nasseri

This thesis is submitted in partial fulfilment for the degree of

Doctor of Philosophy

at the University of St Andrews

January 2019

Abstract

This thesis considers information leakage in cloud virtually isolated networks. Virtual Network (VN) Isolation is a core element of cloud security yet research literature shows that no experimental work, to date, has been conducted to test, discover and evaluate VN isolation data leakage. Consequently, this research focussed on that gap. Deep Dives of the cloud infrastructures were performed, followed by (Kali) penetration tests to detect any leakage. This data was compared to information gathered in the Deep Dive, to determine the level of cloud network infrastructure being exposed. As a major contribution to research, this is the first empirical work to use a Deep Dive approach and a penetration testing methodology applied to both CloudStack and OpenStack to demonstrate cloud network isolation vulnerabilities. The outcomes indicated that Cloud manufacturers need to test their isolation mechanisms more fully and enhance them with available solutions. However, this field needs more industrial data to confirm if the found issues are applicable to non open source cloud technologies. If the problems revealed are widespread then this is a major issue for cloud security. Due to the time constraints, only two cloud testbeds were built and analysed, but many potential future works are listed for analysing more complicated VN, analysing leveraged VN plugins and testing if system complexity will cause more leakage or protect the VN. This research is one of the first empirical building blocks in the field, and gives future researchers the basis for building their research on top of the presented methodology and results and for proposing more effective solutions.

Acknowledgements

To Allah, who provides me with unlimited strength and guidance, and who was beside me all the time when no one was able to understand me and when the PhD stress reached its peak. To you, my lord, my profound gratitude and thanks.

To my late father, Mohamed, and my mother, Aisha, the most precious gifts I have in this life, who have instilled the love of learning in me and my nine siblings, Khalid, Mahfoodah, Samya, Ibrahim, Laila, Ashraf, Omar, Isaa and Khamis. We are a big family with a big heart and strong bonds. My thanks to my lovely family.

To whom prayed for me secretly. To my grandparents, my uncle and aunt that I lost during my PhD journey.

My further acknowledgement and thanks to my beloved country Oman for supporting me in my higher education since my BSc (Sultan Qaboos University, 2004) and providing full scholarships for both my MSc (Loughborough, 2006) and PhD (St Andrews, 2018).

To all my teachers throughout my life who have participated in building my teaching and learning skills.

To my supervisor Dr.Ishbel Duncan, who has given me her best guidance and support.

To St Andrews University, School of computer science, and all its student support facilities.

To all my friends -Thamra and Ildiko- and PhD colleagues in St Andrews .

Declaration

Candidate's Declaration

I, Haifa Mohamed Khamis Al Nasseri, do hereby certify that this thesis, submitted for the degree of PhD, which is approximately **66265** words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for any degree.

I was admitted as a research student at the University of St Andrews in September 2013.

I received funding from an organisation or institution and have acknowledged the funder(s) in the full text of my thesis.

Date: **30 January 2019**

Signature of candidate:

Supervisor's Declaration

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of PhD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date: **30 January 2019**

Signature of supervisor:

Permission for Electronic Publication

I, Haifa Mohamed Khamis Al Nasseri, confirm that my thesis does not contain any third-party material that requires copyright clearance.

The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

Printed copy

No embargo on print copy.

Electronic copy

No embargo on electronic copy.

Date: **30 January 2019**

Signature of candidate:

Date: **30 January 2019**

Signature of supervisor:

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

API	Application Program Interface
br-ex	External Bridge
br-int	Integration Bridge
br-tun	Tunnel Bridge
CNG	Cloud Networking Gateway
CVNI	Cloud Virtual Network Isolation
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EDM	Evolving Defense Mechanism
ext-net	external Network
FEs	Forwarding Elements
FITS	Future Internet Testbed with Security
GRE	Graduate Record Examinations
HPC	High Performance Computing
IaaS	Infrastructure as a Service
InP	Infrastructure Provider
KVM	Kernel-based Virtual Machine
LB	Load Balancer
NaaS	Network as a Service
NAT	Network Address Translation

NFV Network Functions Virtualization

NVP Network Virtualization Platform

OVS Open vSwitch

OVX OpenVirteX

PaaS Platform as a Service

PdP Parallelizing data plane

QEMU Quick Emulator

QoS Quality of Service

SaaS Software as a Service

SAVI Smart Applications on Virtual Infrastructure

SDN Software Defined Network

SEC2 Secure Elastic Cloud Computing

SG Security Group

SR-IOV Single Root Input/ Output Virtualization

TAP Test Access Point

TVDs Trusted Virtual Domains

VBOX VirtualBox

VDE Virtual Distributed Ethernet

Veth Virtual Ethernet

VLAN Virtual Local Area Network

VM Virtual Machine

VN Virtual Network

VNC Virtual Network Computing

VND Virtual Network Diagnosis

VNE Virtual Network Embedding

VNI Virtual Network Isolation

vNIC	Virtual Network Interface Card
VPC	Virtual Router Configuration
vSDN	virtual Software Defined Networks
VXLAN	Virtual Extensible Local Area Network

CHAPTER ONE

INTRODUCTION AND OVERVIEW

1.1 Background Facts

"Cloud Computing is defined by a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms and services are delivered on demand to external customers over the Internet"[1].

Cloud computing is often described by two models; the deployment model[2] and the service model[3]. Clouds can be deployed as private, public or hybrid meaning they are for private or public use or a mix of both. Service models are identified as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) [1][2][3][4] [5] [6][7]. These concepts intersect with each other so that, for example, an IaaS service model can be deployed as private, public or hybrid. These decisions are for the cloud provider to make [2].

A simple definition of IaaS is "Infrastructure as a Service (IaaS): Cloud infrastructure services, whereby a virtualized environment is delivered as a service over the Internet by the provider. The infrastructure can include servers, network equipment, and software [8]." IaaS became an on-demand service built on virtualizing its resources (server, network and storage) and this led to "dynamically changeable topology infrastructure"[9].

Very well known IaaS examples, each with their specific technology aspects, are: physically built (e.g., Emulab) or virtually built (e.g., Amazon EC2) and services delivered in forms of storage (e.g., GoogleFS), network (e.g., Openflow), or computational capability (e.g. Hadoop MapReduce)[10].

Other IaaS are categorised as Open Source cloud technologies such as OpenStack , CloudStack, OpenNebula, Eucalyptus, OpenShift, and Cloud Foundry[11]. Some are more popular than others, OpenStack is between 60% to 97% of the business drive choice of cloud technology users according to their user survey of 2016 [12].

The early concepts of Cloud Computing started with the Data Center which was based on Storage and Computing power. Networking was not yet an adaptable technology and it is the newest aspect among the infrastructure resources provisioning within IaaS. As with other infrastructure resources, networking shared the concept of offering shared resources through virtualization and provided similar benefits such as on-demand provisioning with network load, latency decrease and optimising network performance[9].

In this research we selected IaaS as the service model and private as the deployment model of concern to investigate our research questions. The blue blocks in Figure 1.1 frames our research focus on Network Services within the IaaS, investigated using two Open Source Cloud Platforms: OpenStack and CloudStack.

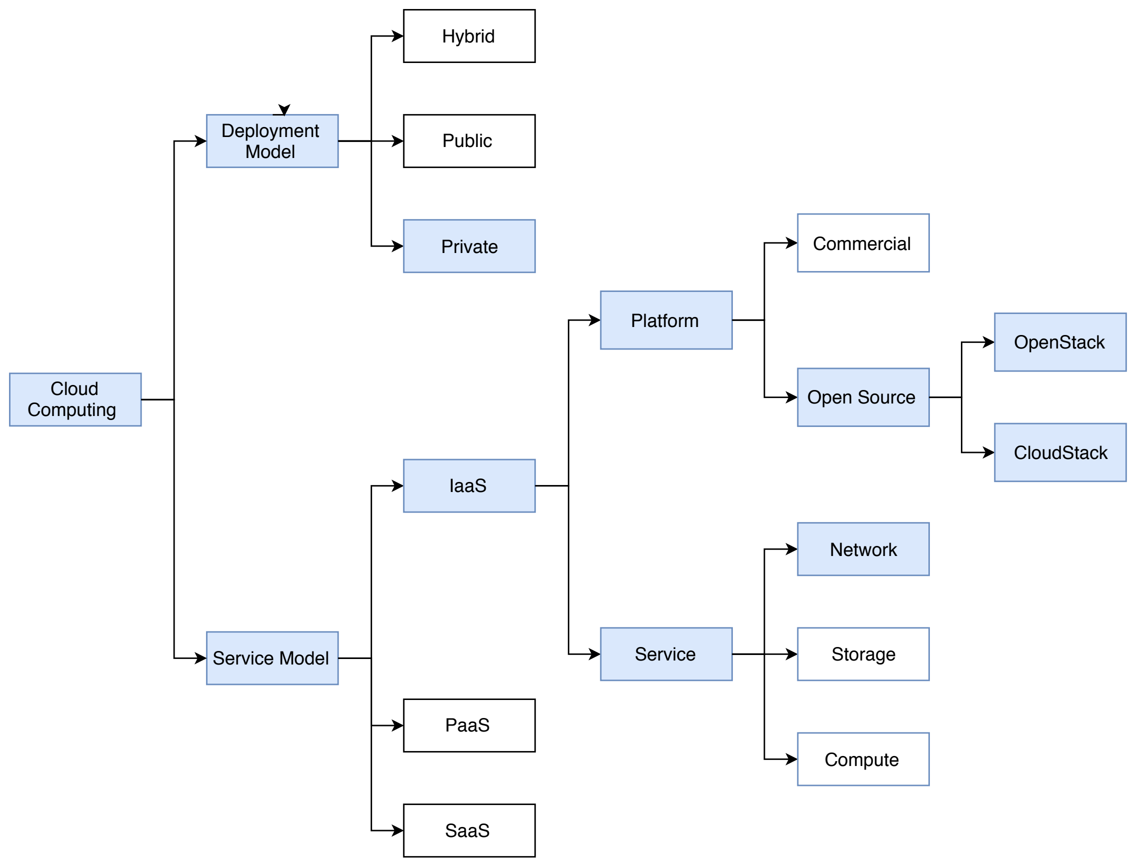


Figure 1.1: Thesis Research Path in Cloud Computing

This research uses Open Source IaaS to test our research questions: : Can Cloud virtually isolated network information leakage be detected?. As the title indicates we aim to evaluate Cloud Virtual Network Isolation (CVNI) Security (Data Leakage) via a Penetration Testing Methodology. We used the Package Software model, as detailed in Chapter 2 Figure 2.1 , to build the Testbed, however, we used the IaaS model to test Virtual Network Isolation (VNI) security (Data Leakage).

1.2 Importance of the Field

Virtual Network (VN) solutions have enabled networking abstraction, yet the compromise between isolation, performance and scalability is an endless battle[13]. VN is the best solution, so far, for network isolated slices co-existence in "a shared network environment"[14].

Multi-tenancy is a vital requirement in cloud environments and therefore isolation between the tenants is necessary and expected. Through isolation, tenant security should be implemented to a different degree depending on the tenant demands via physical and logical isolation[15].

The evolution of Virtual Networks was a result of the demand of compute (processing) virtualization, where the goal was to provide multi-tenancy to users sharing the same physical environment[16]. Compute (Virtual Machine) isolation technology is far more than developed VN isolation technology. The latter is still under development to be more scalable and manageable. In spite of this, Moraes et al[17] and Riggio et al[18] stated that the success of the tenant VN is due to the success of their VN isolation[17].

The need to introduce network virtualization is to isolate shared resources in a form of slices that prevents against tenant resource interference [19]. Isolation is a must have condition in a multi-tenancy network environment and is responsible for mapping between the physical and logical structures[20].

Network virtualization applies a logical isolation between multi-tenant networks in shared physical environments and allows independent address scheme configurations and traffic routing implementation in each tenant network[21].

The success of Virtual Networks is due to isolation. Developers need to use low level system features such as Virtual Local Area Network (VLAN) or use complex hypervisors in the control panel[22].

Cloud providers raised the level of cloud competencies after including networking provisioning along with compute and storage provisioning. Tenants were now able to create their own self-managed, isolated, dynamic, expanding VN[9] and not just a simple network provisioning service.

Existing cloud facilities pose challenges including the compute and network isolation of their tenants[23]. The rapidly changing and scalable computing and virtualization led to an inevitable development and mixing of Software Defined Networking (Software Defined Network (SDN)) and Network Functions Virtualization (NFV) technologies to form modern network structures. These are only feasible through Open Source Application Program Interface (API)s with network application services that are applicable to cloud infrastructures[24]. Several solutions, such as CloudNaaS, CNG, CloNe, SAIL, DCPortals(Ng), NaaS, SilverLine, NVP have been proposed to facilitate Cloud Networking for cloud tenants. These solutions are described in detail in Chapters 2 and 6 .

Many clients believe that vendors and providers will keep their network protected. Some clients do not have the understanding to know if they are vulnerable or where the vulnerabilities lie. Therefore, they are still likely to suffer from disruption, Data Leakage, DoS, etc [25]. A common security concern for clients is in regard to their Data Leakage or damage[26]. A case of SDN Data Leakage is caused by security isolation failure of "logical networks and their credentials", which end in VN deterioration[27]. Therefore, an important security aim is to prevent "Data Leakage" between isolated modules via implementing isolation between "data and process" over the abstraction layers of the cloud[28].

Doubts have been raised about cloud providers keeping the cloud security state under audit and questions raised included, "Does the provider carry out penetration testing?", "How often?", "What is actually tested during the penetration test - for example, do they test the security isolation of each image to ensure it is not possible to break out of one image into another and also gain access to the host infrastructure?" and "Can the Virtual Machine (VM) gain access to the cloud infrastructure?"[29]. Most clients will be worried about their personal data, but they should also be concerned about client personal Data Leakage and their network operational data. Although clients do not necessarily know about network operational data, it is essential for the operation of the client network and, if exposed by unauthorized parties, will lead to attacks to destroy the tenant system including the theft of personal information.

Privacy between tenants in the cloud is another motive to implement isolated networks. Concerns of tenants sharing physical environments include unfair resource consumption, data leaking or stealing, and the tenant side-channel attack[15]. Network security and network isolation are two separate concerns but they are two aspects that are strongly related, which makes them crucial to tenant protection[15].

It is essential to comprehend the cloud internal setup, configuration and structure, as well as how service, applications and communications are isolated in the shared infrastructure [30]. Even Google, a public cloud environment, is seeking a solution to lower the physical and operational isolation costs and find better security mechanisms[23].

Virtualized environments do not work well with fixed security plans, introduced for traditional networks, and therefore analysts should apply regular penetration tests to evaluate the current security state of the infrastructure and defend it from newly introduced threats [31]. However, among the three cloud modules IaaS, PaaS and SaaS; only IaaS and PaaS permit a penetration testing methodology. This is because SaaS providers do not allow tenants to run any penetration testing tools on their cloud environment[32]. *However, this thesis scope covers only IaaS Configuration.*

The importance of penetration test security lies in performing it on a cloud to evaluate cloud security and test the cloud processes and its implementation independently via an outsourced security entity. The granting of a cloud system security certificate should be a standard [33].

1.3 The Research Gap

Although the obvious challenge of virtual networks is in isolation, researchers are concerned with two main isolation aspects; performance isolation and security isolation [18]. The isolation mechanisms in the compute process have reached a mature level, however with network technologies, isolation remains a big challenge[34].

Accomplishing security in virtual networks is about configuring the solution to assure the primary security aspects such as confidentiality, integrity and isolation performance, hence, improving the underlying layer structure elements would prevent VN constriction [35]. However, VN isolation failure, avoidance and restoration are research area gaps in VN isolation "are all open research challenges" [36].

We therefore believe that our research is topical and urgently required to make a significant contribution to the above mentioned research topics of cloud security; Cloud Virtual Network Isolation (CVNI) and data leakage. We aim to investigate CVNI failure by leaking CVNI information and configuration details and determining what can be detected.

One fact needs to be clarified; VN multi-tenancy isolation, even if it is supported with methods such as tunnelling and encapsulating, does not mean that enough security is achieved to eliminate frequent attacks on the network infrastructure in order to gain control of it [37]. VN management operations may be discovered via the VN liabilities and weaknesses and hence risk the infrastructure security and privacy. *This is therefore the basis of our research questions;*

what are the preparations cloud providers and Cloud developers are required to do to enhance isolation from any attack that will expose the infrastructure and break VN isolation?

Carapinha et al.[38] listed several "Technical gaps and open issues" related to VN and isolation as: Scalability, Interoperability of virtual networks, Interoperability with legacy networks, Isolation of Resources and Security and Privacy. *Security and privacy is our main research focus as protecting the VN administration and its governor operations is a necessity.*

The reason for considering security as one of the VN challenges is due to the dynamic and changing nature of VN. It is highly difficult to anticipate the best security mechanism. As VNs co-exist in a shared environment, information privacy is a high priority concern as well as to properly define isolation implementation to prevent risks by, and from, any VNs [38][38].

Carapinha et al.[38] also represented a table of "network virtualisation challenges vs. use cases", where seven scenarios are listed against ten VN challenges. In the Cloud Computing scenarios, four VN challenges received three stars meaning "Crucial challenge; will surely represent an obstacle against deployment if an appropriate solution cannot be found". Those challenges are: Carrier-grade reliability, Isolation, Security and Operational complexity. Continuously hosting new VNs in the provider networks endlessly modifies the isolation and security requirements.

Part of the VN research gap is the "security and resource isolation" standardization, which makes the solution to "shared resource strict isolation", attack risk and misconfiguration migration prevention more adaptable to any environment that is capable of implementing VNs [38].

Due to the obvious research gaps in VN isolation security and the limited published evidence in the field, this research aims to contribute by enriching the knowledge bank of the field and to perform empirical evidence to determine proposed potential solutions for VN security.

Scott-Hayward et al[27] noted that the research field of "Virtual Network Data Leakage" is novel and topical. *We do not claim that we produced a complete solution to the noted research gap, but we confidently declare that we have made the first empirical step in structuring a solution for Data Leakage and potential attacks in SDN. To be more precise, we extended the scope by replacing SDN with cloud networking of the IaaS platform, as cloud networking consists of SDN, NFV and other services and components. Moreover, we believe that our research is a relevant contribution to Data Modification, based on the theory of data leakage and identification to gain knowledge about how that data could be modified. Chapter 2 expands the details of the gap presented by Scott-Hayward et al [27].*

Scott-Hayward et al also strongly encouraged security researchers to adopt two "Open Challenges: Data Leakage and Data Modification". *We have therefore worked on one of the least tackled* *challenges, SDN Data Leakage, which in our research we term Cloud Virtual Network Data leakage.*

1.4 Contributions

Our testbed is a hybrid network infrastructure with KVM, Open vSwitch (OVS), bridging, OVSDrive plugins and firewalls security, Virtual Extensible Local Area Network (VXLAN), VLAN, NFV (VM, namespace) for Dynamic Host Configuration Protocol (DHCP), routing and Domain Name System (DNS) (see glossary for terms which are explained further in Chapter 2). The testbeds demonstrate multi-tenancy and service provisioning in Ubuntu OpenStack and CloudStack environments with KVM as hypervisor and SDN/OpenFlow software technologies, as well as, dynamic on-demand provisioning for computing and networking services.

In order to clarify the contribution of this research we will list the most relevant research and illustrate where this thesis research differs from them.

1. IaaS security issue is, as Bouayad et al[39] described, the "VM boundaries" where the VM as virtualized entities share the underlying resources with other VMs. If isolation is not as demanded, co-existence is a major security issue. *They investigate the Compute service while we consider network services. Hence, we can confidently state that securing the VN boundaries is an IaaS issue that needs investigated.*

2. Srivastava[40] researched a Proof of Concept implementation of networking agents configured by the OpenStack Networking module Neutron with SDN. The aim of their research was to investigate the performance of the networking functionality of an OpenStack cloud in order to lower the TCP/IP stack load. The implementation of their cloud network was based on Overlay L2 & L3 routing and included iptables and a Linux kernel with virtualized networks using OVS and OpenFlow protocols as SDN technology. *We share a similar approach to the overarching area and the implemented testbed; yet the aim of our research is different. We built similar testbeds to investigate the CVNI security for information leakage. Moreover, we expanded their cloud research approach by implementing another IaaS cloud, CloudStack, using the exact same methodologies to test our research question.*

3. From the security perspective in OpenStack, the network isolation is appropriate if the VM reacts with an NMAP scan according to how the Security Group (SG) has been configured. Anisetti et al [41] successfully tested the management between the SG and the iptable rules implementation in OpenStack networks when a network security enquiry was run. Hence, they[41] analysed and certified OpenStack network isolation, authentication and

confidentiality. *We furthered the experiment by using two IaaS Cloud Computing testbeds - OpenStack and CloudStack- with penetration tests including NMAP amongst other tools, and analysing the collected data to to establish whether important data knowledge can be detected to cause CVNI breakage.*

4. Aderholdt et al[24] used NMAP from one tenant area to another known tenant in order to verify the "tenant isolation with respect to Compute and Storage". They tested the network management externally by attacking common ports such as Telnet, HTTP, HTTPS, NMAP and SSH as well as any API interfaces with potential vulnerabilities. *We performed a similar testing approach, except that we adopted a full penetration test and aimed for verification of tenant isolation with respect to cloud networking.*

5. Bechtsoudis and Sklavos[25] is one of the few papers that fully applies penetration testing on a general network system as a research methodology, as is more popular in commercial, business and industrial worlds. It presents rough penetration test phases and consists of four stages: "planning, discovery, exploitation and reporting". *Although the general scope is similar to ours, they aimed to "expose common traditional-network misconfiguration", their research did not focus on cloud virtualized networking, leaking VN information, Tenants VN multi-tenancy breakage or detecting the cloud virtual network isolation security vulnerabilities via pentesting as does this thesis.*

6. Scott-Hayward et al[27] listed the research gap with regard to the security aspects of leaking the SDN/OpenFlow (OpenStack rule) switches, data flow and forwarding rules. No empirical research that we know of has been done in this area. Therefore, we had no research base to build on. *We took a basic, logical step of exposing the cloud network, as a whole, by penetration testing and aimed to determine what information can be leaked from the cloud network information. We did not reach our expectation of leaking lower level information such as OpenFlow rules, actions or controller messages. However, this clarified for us our approach limitation and gives more clarity about how to plan for deeper data leakage experiments. This is not presented in this thesis due to time limitations, but is stated in the future work section 7.3(Chapter 7).*

7. Moreover, Scott-Hayward et al[27] encourages researchers to use "Cross-VM Side Channel Attacks" to cause Data Leakage and target the OpenFlow virtual switches and their configuration files. *Our internal penetration testing experiment is based on an idea similar to what is stated in this paper[27] as "a malicious VM identifies a vulnerable VM and extracts secure information from the targeted resource".*

8. The Ristenpart et al [42] presented attacks where the attacker installs a mischievous VM in the same shared environment as the targeted tenants and launches attacks to gain knowledge about the co-existence VMs. *Our empirical security testing includes internal cloud testing similar to[42] except that we specifically installed a Kali VM as our mischievous VM to perform penetration testing against the co-existing cloud resources in order to discover the co-existing VN components and VN information leakage. Moreover, we supported our test with external penetration tests. We also repeated the whole process with another Cloud testbed, CloudStack. We collected some network information including the network distance as an example of how to measure the distance between victim tenants and the attacker tenant.*

9. Mirantis OpenStack[43] is another academic research paper discussed a case where penetration testing tools are used by attackers to search for a weak company account via scanning the business network. The difference of this research from our research is that they applied a security assessment to the OpenStack dashboard and provided recommendations to enhance the dashboard security, i.e. through the user dashboard interface called Horizon. *However, we applied penetration tests on the Cloud Network Infrastructure to leak VN information and configuration.*

10. Labarge and McGuire[44] also used penetration tests on the OpenStack Horizon dashboard to detect vulnerabilities. This research recommended that OpenStack Clouds require regular penetration tests to clear up the weaknesses of the platform and proved the weakness of the dashboard for gaining access to the tenants' information. We have not encountered any other research that targeted the cloud network and revealed their mis-design, misconfiguration or vulnerabilities via penetration testing. *However, our research was extended to verify the methodology by applying it on a similar cloud, namely CloudStack, and we compared the results between the two testbeds. We wished to determine if there was a pattern that indicates if issues occur in different platforms, which leads into a general design issue.*

11. Wang et al. [45] declared little interesting research has been invested on virtual network security, hardly any in cloud networking security, and almost none on virtual network information leakage of cloud computing. *Therefore, the presented research in this thesis is a major step forward in empirical Cloud Virtual Network Isolation research.*

In general and as Figure 1.2 presents below, we summarize our research contribution as the following:

- One of the earliest empirical works on a *DeepDive* (a deep structural test) on IaaS cloud, especially on the *CloudStack Network Infrastructure*.

- One of the few empirical works on *penetration testing methodology* for cloud network infrastructure (applied to both CloudStack and OpenStack) and a demonstration of cloud network isolation vulnerabilities.

- Discovery of cloud network infrastructure *mis-design/ misconfiguration*, as is illustrated in Chapter 5 and 6.

These points are colour coded yellow, orange and red in Figure 1.2 to indicate different aspects of the thesis contribution. The yellow bocks represent the DeepDive method and Cloud Network Infrastructure graphical representation. The orange blocks represent another DeepDive outcome which we termed the Cloud Infrastructure Knowledge Guide, where VN Components and configurations were identified as well as locating the tenant isolated VNs. Finally, the red blocks show the Cloud infrastructure security tests, using Penetration Testing, in order to detect the Cloud Network Infrastructure Data Leakage, Vulnerabilities and Design weakness.

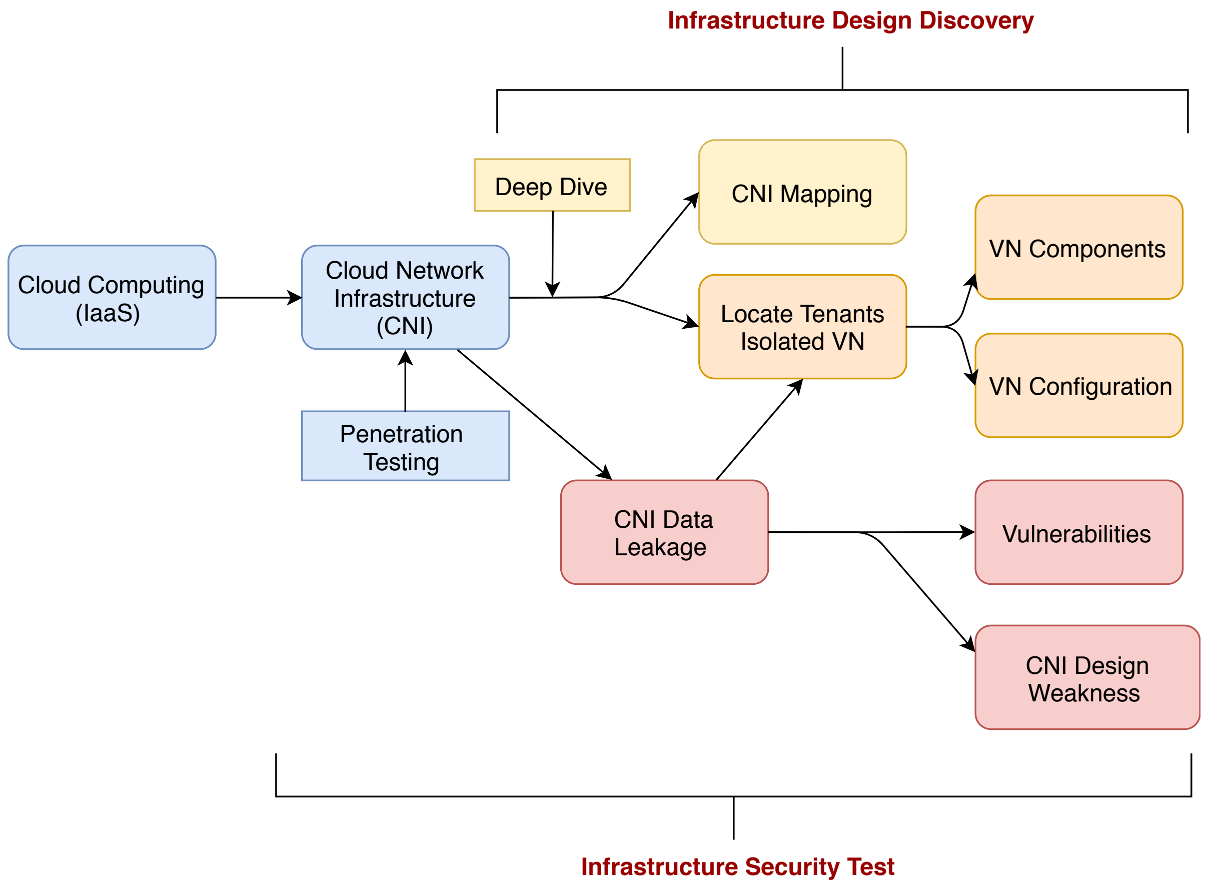


Figure 1.2: Thesis Contribution Scope

1.5 Research Hypothesis

We consider whether current penetration testing tools can detect Cloud Virtual Network Isolation issues. Hence, our basic H1 Hypothesis is as follows:

H1: Using common penetration testing tools, an attacker can discover Data Leakage of Cloud tenant information due to poor Cloud Virtual Network Isolation and infrastructure design.

Therefore, the Null Hypothesis follows as:

H0: Null hypothesis: The security of the IaaS Cloud Virtual Network Isolation **is robust** against Data Leakage caused by the current penetration tools.

1.6 Problem Statement

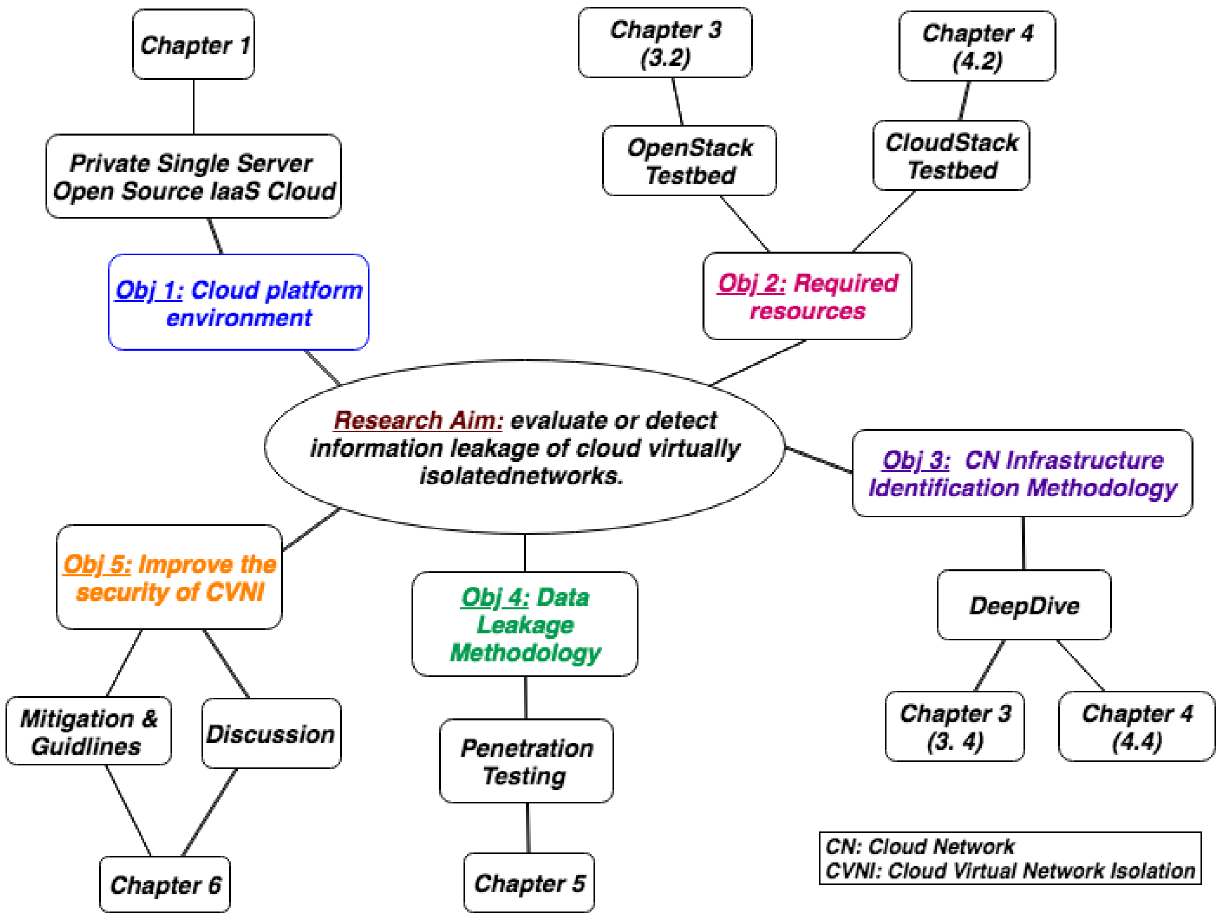

Chapter 1
Private Single Server Open Source IaaS Cloud
Obj 1: Cloud platform environment
Chapter 3 (3.2)
OpenStack Testbed
Chapter 4 (4.2)
CloudStack Testbed
Obj 2: Required resources
Research Aim: evaluate or detect information leakage of cloud virtually isolatednetworks.
Obj 3: CN Infrastructure Identification Methodology
DeepDive
Chapter 3 (3. 4)
Chapter 4 (4.4)
Obj 4: Data Leakage Methodology
Penetration Testing
Chapter 5
Obj 5: Improve the security of CVNI
Mitigation & Guidlines
Discussion
Chapter 6
CN: Cloud Network
CVNI: Cloud Virtual Network Isolation

Figure 1.3: Thesis Aim and Objectives

Figure 1.3 summarises the thesis aim and objectives; however, detailed information is elaborated below.

Research Aim: evaluate or detect information leakage of cloud virtually isolated networks.

Objectives: given the lack of existing methods to reach to the research aim, the objectives of our study were designed to address the following questions:

1. What cloud platforms could be a possible environment to apply the research question on?
2. What are the required resources to evaluate CVNI?
3. What is the best methodology to identify the construction of the cloud network infrastructure, VN components and Isolation mechanism?
4. How can Data Leakage be detected or determined? And what are the characteristics of the data leaked?
5. How can we improve the security of CVNI in the presence of Data Leakage?

To answer the questions that drive the objectives of this research we state the following:

1. Cloud platform environment:
   - We used a *private single server IaaS Cloud* to investigate the occurrence of Virtual Network (VN) configuration and functionality information leakage.
2. Required resources:
   - To build *two IaaS cloud Testbeds (OpenStack and CloudStack)* with advanced VN isolation provisioning services and compare their structures.
3. The best *methodology* to identify the construction of the cloud network infrastructure, VN components and Isolation mechanisms
   - *DeepDive Methodology* (our own version of White box Testing).
4. Data leakage *detection and characteristics* identification:
   - Collect leaked information from running *Penetration Tests (Internally and Externally)* and *analyse information to determine* the components and configuration of the tenant VNs within the Cloud Network infrastructure of the two Testbeds.
5. Improve the security of CVNI:
   - Present the *vulnerabilities* and *misconfiguration or mis-design* of the Cloud Network infrastructure and propose *mitigation* and security *guidelines.*

1.7 Thesis Map

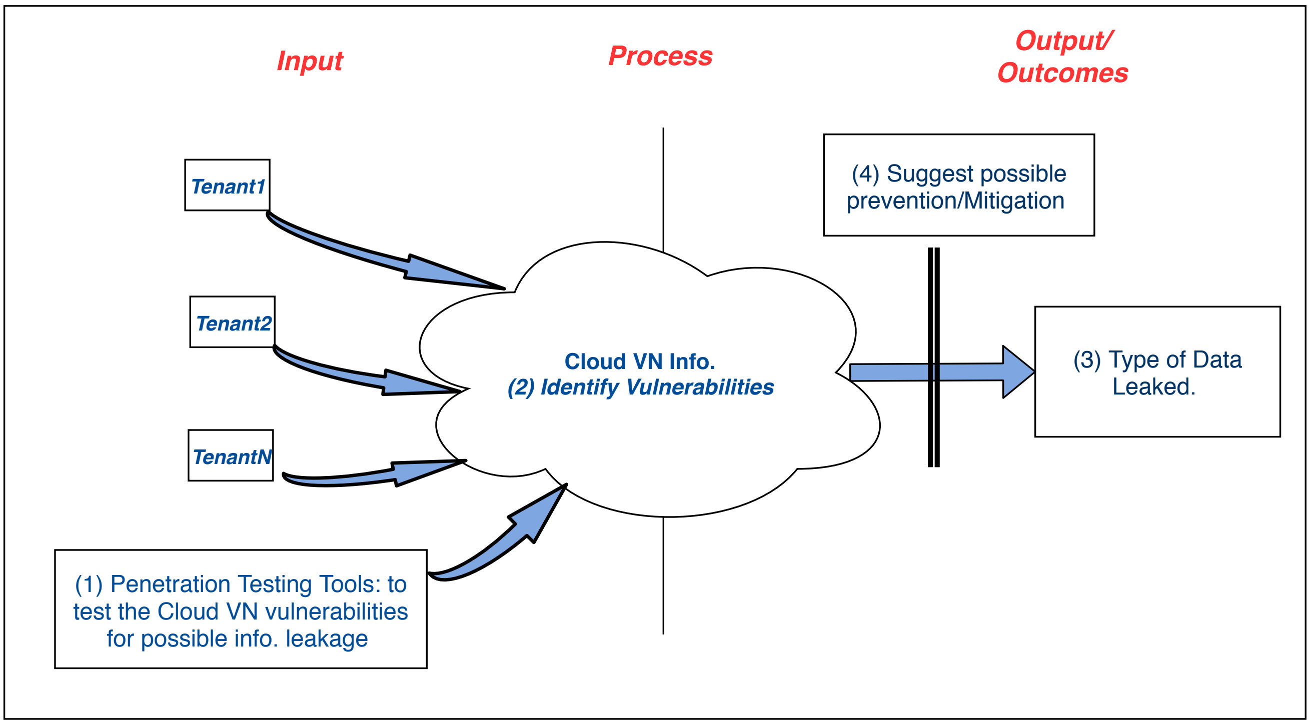

Input
Process
Output/ Outcomes
Tenant1
Tenant2
TenantN
(1) Penetration Testing Tools: to test the Cloud VN vulnerabilities for possible info. leakage
Cloud VN Info.
(2) Identify Vulnerabilities
(4) Suggest possible prevention/Mitigation
(3) Type of Data Leaked.

Figure 1.4: Research Map

The research was mapped as shown in Figure 1.4 to show the flow of the empirical work to answer our research questions and problems. We divided the research into several stages, which included:

- To build the advanced VN Cloud Testbeds with multi-tenancy scenarios.
- To select one tenant to be a rogue tenant and use the initial information to perform bi-directional pentesting attacks (External and Internal attacks).
- To enable the rogue tenant to launch currently available penetration tools, we used Kali Tool kits, to test Cloud VN Vulnerabilities for possible information leakage.
- To identify the vulnerabilities, their relevance to VN and the impacts on identifying the co-existing VN of the other tenants, or of breaking VN Isolation. At this stage we make our decision to accept or reject the Null Hypothesis as stated above.
- To demonstrate leaked data and their characteristics as well as their source and the cause of the leakage.

- And finally, based on the results above, we suggest possible prevention or mitigation guidelines.

1.8 Thesis Structure

This thesis consists of six chapters and two appendices.

Chapter 2: The literature review and background is presented in this chapter.

Chapter 3: This chapter is reserved for illustrating the OpenStack Testbed construction and the DeepDive methodology used on the OpenStack Cloud Testbed.

Chapter 4: Focuses on the CloudStack Cloud Testbed with the same DeepDive Methodology used to map the Cloud Networking Infrastructure.

Chapter 5: Discusses the Penetration Testing applied to determine the Cloud Virtual Network Data Leakage on both testbeds (OpenStack and CloudStack) and presents the results found.

Chapter 6: Discusses the results found in Chapter 5 for both testbeds and lists the vulnerability mitigation strategy and potential solutions.

Chapter 7: presents the thesis conclusion and potential future work.

1.9 Publications

Published Conference Proceedings

Al Nasseri , H M K & Duncan , I M M 2016 , ' Investigation of Virtual Network Isolation Security in Cloud computing : Data Leakage Issues ' Paper presented at International Conference of Big Data in Cybersecurity , Edinburgh , United Kingdom , 10/05/16 - 10/05/16

CHAPTER TWO

CONTEXT SURVEY

This chapter covers many areas including cloud computing, virtualisation, virtual networks, network components, security testing and data leakage. It also notes the technologies currently available and explains why decisions were made as to the experimental testbed and test requirements. Subsequently, the chapter is long and detailed but grounds the experimental evidence in current theory and practice. Indicators of decisions made for the research described in this thesis are written in *italics*.

2.1 Cloud Computing

Cloud Computing was introduced in Chapter1 as comprising two models which are the deployment model and the service model. One of the service models is Infrastructure as a Service (IaaS) which is the focus of this research. This section discusses more of the terminology and outlines the complexity of a cloud system.

Argyropoulos et al[2] showed that the highest energy usage layer of the cloud stack was in the Iaas and PaaS with 75% of a cloud's energy consumption. As the cloud model, Figure 2.1[8], is built on a hierarchical concept [4], enhancing the lower layer (IaaS) will enhance the upper layers (PaaS and SaaS). *Consequently, Although we constructed the Cloud Testbed using the Package Software - we managed everything-, the security research described in this thesis is focussed on the IaaS layer.*

Clients are tenants of a cloud and each tenant can host many users and many VMs. An individual user could be a client hosting tenants or they could be a user of one or more VMs within a tenancy. There are presumed isolation of storage, compute and networking services. Users are expected to be isolated from each other and tenants should be isolated from each other, i.e. isolation through physical or logical, virtual or non-virtual components is a basic security requirement.

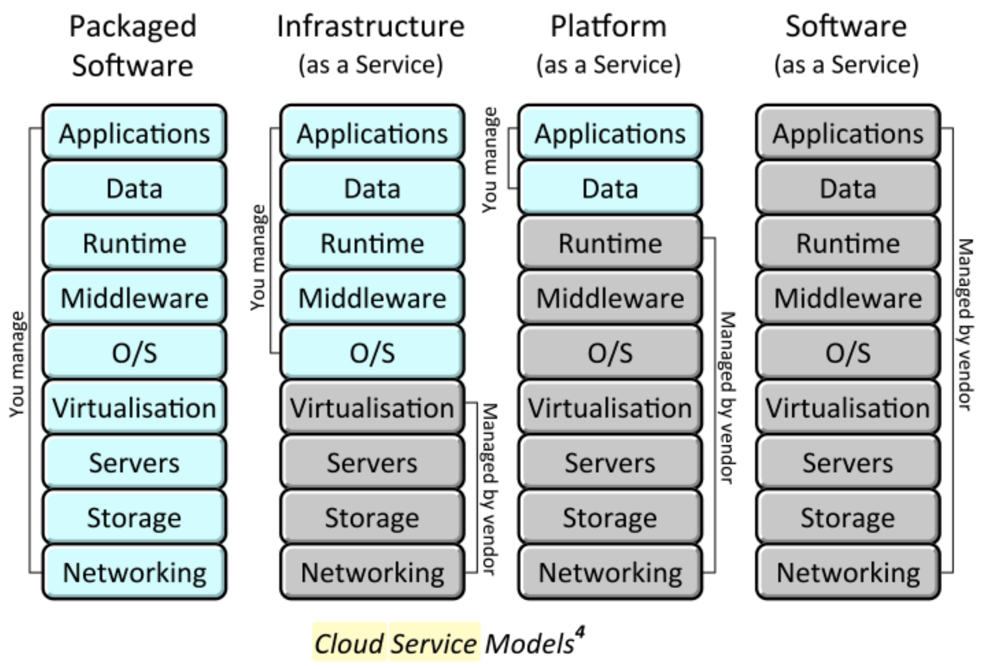


Figure 2.1: Cloud Service Models [8]

Multi-tenancy in cloud service models is the construct of many tenants and many users through policy-driven enforcement, segmentation, isolation, governance, service levels, and chargeback/billing models for different consumer constituencies [32].

Network virtualization is a networking environment that allows service providers to dynamically compose multiple heterogeneous virtual networks that coexist together in isolation from each other [36].

2.1.1 Infrastructure as a Service (IaaS)

The IaaS layer offers hardware, software and services as shown in Figure 2.1. This is represented to the users as software applications [1] [8]. On-demand provisioning of infrastructure is sometimes called **Hardware as a Service** [8]. The significance of this service as a solution lies with its virtualization usability, which leads to cost reduction via resource utilization effectiveness [1]. The users do not need to consider the infrastructure below as they launch and manage their own VMs [4], which are configured with their choices of OS, software and applications [46]. User control starts from their own VMs but is never extended to the infrastructure underneath [47]. Infrastructure resources such as storage, virtualization, hardware and network are only under the control of IaaS provider [5].

2.1.2 IaaS Examples

IaaS by itself is divided into different categories either by their deployment, offering or source. For example, a well known, publicly deployed IaaS with a para-virtualization offering is Amazon EC2 (Elastic Cloud Computing), with a storage offering service is S3 (Simple Storage Service), with full virtualization (GoGrid, Skytap), or with grid computing (Sun Grid), or with management (RightScale) [1][4].

Some other very well known IaaS examples are services delivered in forms of storage (e.g., GoogleFS), network (e.g., Openflow), or computational capability (e.g. Hadoop MapReduce) [10].

Sefraoui et al's comparison study [48]was done on Open Source cloud technologies. The study was a comparative study of four open source cloud technologies: Eucalyptus, OpenStack, OpenNebula and Cloudstack. The comparative study was focussed on the following parameters: storage, network, security, hypervisor, scalable, installation, documentation and code openness. Both CloudStack and OpenStack received the highest rating with only one point difference between the two in the favour of OpenStack. *This study had a major impact on our decision to select CloudStack and OpenStack to be our private cloud IaaS testbeds.*

2.1.2.1 CloudStack

CloudStack is an open source IaaS cloud stack, established by cloud.com and later taken over by the Apache Software Foundation. It supports multiple hypervisors such as VMware, Oracle VM (OVM), KVM , Xen and XenServer. It is scripted mostly in Java and the code is licenced under Apache2. It supports EC2, S3 and Vclouds API besides its own APIs [48]. Due to its hierarchical structures CloudStack has the ability to manage physical resources, networks and VMs. The CloudStack architecture is structured into Zones, Pods, Clusters and Hypervisors [49].

2.1.2.2 OpenStack

OpenStack has been a major focus for researchers in cloud computing and networking. OpenStack is a computing and storage IaaS which uses Open Source software enabling functionality, infrastructure, resource management and solution monitoring [50]. It was established by Rackspace and NASA, then became the OpenStack Foundation [49]. Amazon Compute and Storage provider infrastructure are alternative solutions, mostly developed in python and licensed under Apache2. Due to its popularity, it has gained a good reputation in the field and is considered “a standard in effect” [8] . It supports a wide range of hypervisors such as VMware, ESX, Hyper-V, KVM, LXC, QEMU, UML, Xen and XenServer. It is not a hierarchical architecture and yet it has been designed to support very large infrastructures. Its main architectural component consists of interconnected modules representing services and they are Nova (compute), Glance (image service), Keystone (authentication and authorisation), Neutron (network), Horizon (dashboard) and Swift (storage) [48].

OpenStack cloud networking uses a variety of networking functions and mechanisms such as Overlay L2 routing & L3 routing (Network Address Translation, NAT), Linux kernel namespaces, Linux bridges and Open vSwitch (OVS) bridges. Most the L2 and L3 routing are processed by namespaces, which are effectively the isolated containers of network devices. Although the network functions run inside the namespaces, physically they are run by the physical infrastructure IP/TCP stack and IPtables. This is viewed as load on the underlying cloud structure as all the traffic generated by the VMs and directed to namespaces are processed while sustaining the isolation on the host machine [40].

In OpenStack the integration of several network technologies and implementation gives the system the ability to react to customer demands. For example, the VLAN and Namespaces provide tenant isolation, Security Groups are responsible for the VM security against illegal access. VXLAN and GRE are L2-to-L3 tunnelling implementation to extend multi-tenancy and isolation beyond the tenant local VN and the single host. All the above and the isolation assurance are managed and monitored by a module called **Neutron**. The path of communication from VMs to the Internet and back starts from the traffic generated by the VM in a compute. This is sent to the L2 vSwitches with their designated VLAN. Leaving the compute to the network node, the traffic goes across the L3 vRouter which is, in effect, a namespace, in order to route the traffic via the gateway to the Internet. The process is reversed in the other direction [51].

2.2 Terminology Definitions

There are many terminologies in cloud computing, virtual networking, isolation and security and therefore the following definitions are the most important for the research fields.

A ***slice*** is a user-defined subset of virtual networking and computing resources. A slice has the basic property of being isolated from other slices defined over the same physical resources, and is dynamically extensible across multiple domains. On top of each slice, a specific set of control tools can be instantiated, depending on the specific domains it traverses [52].

A ***Virtual Local Area Network***(VLAN) provides elevated levels of trust, security, and isolation [36].

A ***Layer 1 Virtual Private Network (VPN)*** allows each service network to have an independent address space, independent Layer 1 resource view, independent policies, and complete isolation [36]. A VPN is usually a solution for traffic isolation in a bigger network environment, but can be used in VNs [53].

A **VPN** is defined [38] as a generic term that covers the use of public or private networks to create groups of users that are separated from other network users that may communicate among them as if they were on a private network. There are two basic types of VPN including the Customer Edged (CE) based VPN, in which the shared service provider network does not have any knowledge of the VPN. The VPN specific procedures are performed in the CE devices and Provider Edge (PE) Based VPN, in which the service provider network is used to connect customer sites using shared resources and the Provider Edge device (PE) maintains the VPN state, isolating users of different VPNs.

Enhanced layer 2 switches, are referred to as **Forwarding Elements (FE)**. These are deployed only at certain aggregation points to provide the required virtualization functions. FEs are basically Ethernet switches with enhanced Application Programming Interfaces (API) that allow them to be controlled [23].

In order to allow multiple external parties to run, possibly conflicting, code on the same network elements, ***active and programmable networks*** also provide isolated environments to avoid conflicts and network instability [36].

2.3 Cloud Networks

In addition to the cloud computing benefits of Iaas, PaaS and SaaS, an additional service of cloud networking was added which provisioned network functionality to the cloud tenants (and users) in a similar way to the compute and storage services. The new service delivers advantages such as lowering the network delay and connection creation. However, it adds more challenges to the security aspects [54].

A **Data Centre** (DC) is a facility composed of networked machines for processing and storage. main goal of the IaaS is to effect users accessing a DC's physical resources over the Internet using VMs. Adding virtual networks to the IaaS management system ensures efficacy for users and cloud providers giving compute and storage services and supporting DCs with a static network infrastructure that supports basic, preliminary network features [9]. Later, the cloud service was revolutionized though interface standardization to include a network infrastructure able to cope with the escalated demands of technology [55].

The first step in restructuring the cloud network infrastructure was to redefine the network capabilities and requirements from the users rather than the cloud admin/provider roles [55]. Providing networking services to cloud users leverages multi-tenancies through virtualizations. Moreover, it gives an on-demand, scalable service with less delays. There are also more benefits to the cloud providers in terms of service performance assurance [9].

Virtualization is a main component of IaaS and includes a hypervisor, virtual switches and VMs. These are often commonly called the Cloud Virtual Infrastructure (CVI) [46]. *The focus of this research is this area; the virtualization that covers the networking service such as vSwitches.*

2.3.1 SAIL and CloNE

A current service requirement of a cloud is on-demand available network resources and cloud resource network management. Hence the need to create a system/module/service to address networking in the cloud and orchestrate it within the cloud management system. This was the main aim of the SAIL project when it was established [9].

Originally cloud services were introduced from a business perspective to rent resources as a service such as renting storage or compute resources as service from IaaS through on-demand & pay-as-you-go lease. The SAIL project goal was to provide a similar facility for the network service within the cloud and leverage such services by empowering the cloud network capabilities to exceed the static basic cloud networking the DCs used to have and offer [55].

CloNe [56] was one of the early projects to add cloud networking provisioning and a facilitating approach, similar to the compute and storage resource provisioning in the cloud, through a web-based interface or API.

Before the SAIL project, cloud networking only satisfied the multi-tenancy requirement by virtualizing the cloud infrastructure and introducing several virtual elements such as **vRouters** and **vLinks**. This merely managed the virtual network in a similar fashion to managing a physical network, and users were not able to have their on-demand, dynamic network requirement fulfilled. The SAIL project combined the capability of managing cloud computing and network facilities between the cloud distributed resources along with cloud network utilization rather than just focusing on improving only the cloud networking [57].

2.4 SDN

According to Kloti et al [58], Software Defined Networking **(SDN)** “has been proposed as a drastic shift in the networking paradigm, by decoupling network control from the data plane and making the switching infrastructure truly programmable”. As the SDN Architecture Overview from the Open Networking Foundation [59] states "In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. As a result, enterprises and carriers gain unprecedented programmability, automation, and network control, enabling them to build highly scalable, flexible networks that readily adapt to changing business needs".

Cloud computing increasingly attracted more customers which led to raised demands and services specifically for IaaS cloud providers. One of the main demands was for providing multi-tenant networks with traffic isolation assurance. SDN was the proffered solution to make the new demands feasible, as it does not demand specific hardware networking devices, but merely requests the cloud physical servers to be configured with SDN vSwitches [60].

The key ideas of SDN functionality are [58]:

- The network logically splits into two planes; control and data.
- A programmable centralized controller, which is resident between the control and data planes and uses standard defined interfaces and protocols to serve the communication.
- Controller protocols are created based on the abstraction concept to serve the distributed system environment.

One of many advantages of SDN is promoting is multi-tenancy and as clouds are based on multi-tenancy with tenant resources isolation, SDN is a good combination of functions to leverage the efficiency of such systems. Tenant resources are connected via virtualized sliced networks (VN) created by SDN, which is then capable of managing them [61].

2.4.1 OpenFlow

Ibrahim [46] stated that, “The OpenFlow architecture is a proposal from the Clean Slate initiative to define an open protocol that sets up forward tables in switches. It is the basis of the Software Defined Network (SDN) architecture, where the network can be modified by the user. This proposal tries to use the most basic abstraction layer of the switch; it is the definition of forward tables, in order to achieve better performance”.

OpenFlow [7] is now a standardized protocol [4]. It is used for the interaction between a network switch, constituting the data plane, and a controller, constituting the control plane [58]. Switches in the data plane execute the traffic communication based on L2 & L3 switching as determined by the flow rules installed in the flow tables. Each switch is associated with a set of flow tables that contain sets of flow rules installed by the controller. Switches match the received packet headers with the flow rules and used an associated action section to determine how to forward the traffic. The controller, which exists in the controller plane, installs the flow rules in the tables either as pre-configured or as a response to the switch alert of unmatched traffic received.

OpenFlow is added as a feature to commercial Ethernet switches, routers and wireless access points and provides a standardized hook to allow researchers to run experiments, without requiring vendors to expose the internal workings of their network devices. OpenFlow is implemented by major vendors, with OpenFlow-enabled switches now commercially available [62][24].

SDN allows network programmability via a software controller. Hence, proposing the OpenFlow protocol along with OpenFlow switches and controllers and the communication strategy between the switches and controller was a major step forward for SDN [45].

As noted earlier, flow rules are defined by the SDN controller via flow tables within the OpenFlow switches. SDN executes those flow rules as a form of access control to traffic forwarding/handling through the OpenFlow switches using defined rules based on the inbound and outbound trusted domain traffic. Security can be enhanced by forwarding the traffic to a firewall that works with the OpenFlow switch, where the traffic is analyzed and dropped or passed to the forwarding rules. Note that the firewall does not distinguish between the data plane and the control traffic, so the SDN controller must take responsibility in handling that traffic by itself [58]. Flow rule configuration may be defined by policies set by the SDN administrator or a third party developer.

SDN/OVS switches isolate the traffic via VLAN tagging that is added into the packet Ethernet header [63]. *We are interested in the security aspect of SDN isolation components for this research because of the multitude of different technologies and methods used by users and providers.*

2.4.2 Control plane and Data Plane

In traditional networking, the control plane and data plane traffic share the same path. In SDN, the control and data are separated to facilitate an abstract network design. The control plane traffic consists of L2 and L3 protocols, management traffic such as the Simple Network Management Protocol (SNMP) and Secure Shell (SSH). The data plane is traffic containing the data exchanged between applications, i.e. application data [24].

2.4.3 Abstraction

The concept of network abstraction is primarily focused on supporting network policies and controls rather than specific methods that can be used to deploy the controls through physical hardware. In the context of SDN, it refers to connections, ports and data flow policies rather than the physical connection descriptions such as VLANs, IP addresses, and physical networking devices. This network abstraction layer facilitates APIs that can be used to configure details about the network [24].

2.4.4 Northbound and Southbound interfaces

Aderholdt et al [24] introduced the concept of North and Southbound- Figure2.2 [64] - traffic which refers to the information exchanged between the decoupled control and data planes of the SDN. Northbound specifically refers to information from the data plane to the control plane, and Southbound refers to information from the control plane to the data plane.

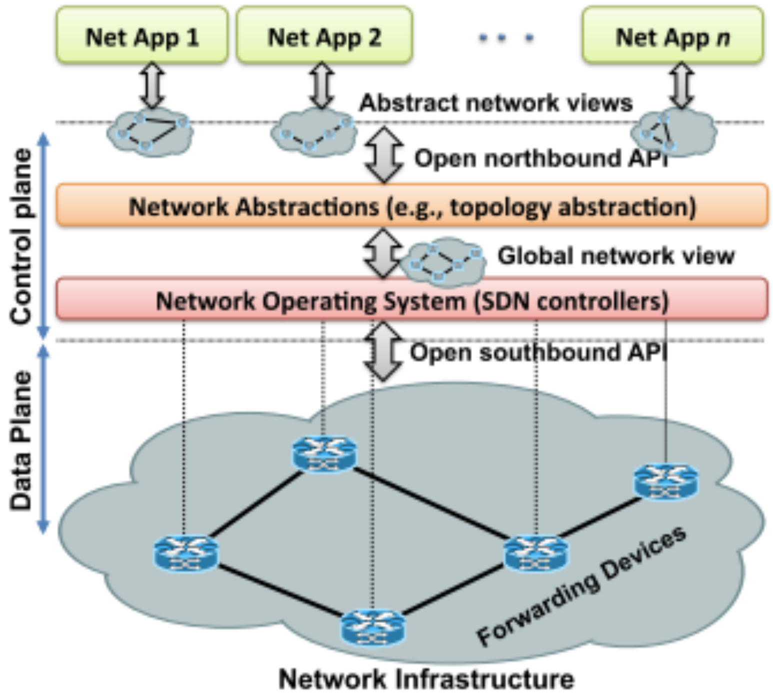


Figure 2.2: SDN Architecture and its Fundamental Abstractions[64].

2.5 Network Function Virtualization (NFV)

The NFV initiative defined standard elementary building blocks to implement service chains capable of adapting to the rapidly changing user requirements [46]. SDN is another important enabler, aimed at de-coupling the network data plane from a logically centralized control plane, thus providing a more flexible and programmatic control of network devices [7] [62]. In the OpenStack User survey in 2016 [12] 95% of the cloud users showed interest in SDN and NFV as evolving technologies to the cloud.

Networking virtualization in the cloud was necessary for multi-tenancy, flexibility, manageability, performance and configuration. Yet it was not straightforward to plug in virtual networking within a cloud. This required diverse protocol layer implementations; VLANs for local implementation, VPN for public connections, overlay networks and tunneling for distributed cloud hosts, embedding NFV and SDNs in the cloud to provide the integration vision of the cloud networking and satisfy the demands of the cloud users. Adapting multiple network technologies, mechanisms and functionalities shifted the network architecture to a whole new level [51].

To distinguish between the two we can think of NFV as the upper layer of the network as it represents the network services through virtualizing the services of load balancing, firewalls, gateways etc. SDN is the lower layer of the network where it handles the network functions and processes through separating the forward plane from the control plane. This means slicing the network into multiple flexible, programmable and manageable virtual network slices [51].

Argyropoulos et al [2] noted an SDN and NFV in clouds survey, in which VLAN technology was placed first for traffic isolation used by NRENs (National Research and Education Networking organisations), followed by both **VXLAN** and **GRE** with equal weight. Other mechanisms used were MAC address space separation, proxy ARP or MPLS. Some clouds used specific isolation implementation and tools such as OpenStack Neutron, ML2, OVS, FlowVisor, Cisco Boxes and OpenNaaS/OpenVirteX above Floodlight.

A full virtualization technique may result in considerable overheads that will affect the performance negatively. The use of hybrid-SDN (an SDN and NFV combination) not only improves the performance because of its lower overheads but implements an instant security and isolation for provisioned compute and storage resources via L2 and L3 configuration [24].

We did not use a hybrid approach because we wanted to focus on single virtual network building blocks before investigating more complex combinations of strategies.

2.6 Virtual Networks

Virtual Networks (VN) are defined as a "Networking environment that allow one or multiple service providers to compose (in a dynamic or static way) multiple heterogeneous virtual networks that co-exist together in isolation from each other and to deploy customised end-to-end services on-the-fly, as well as manage them on those virtual networks for the end-users by effectively sharing and utilising underlying network resources leased from one or multiple infrastructure providers" [38].

A VN is a separation of the control and data planes. A traditional network would have one control and one data plane, however another approach is to virtualize the control plane only where every virtual network has its own control plane but shares the data plane with other networks (sharing-weak isolation). Several approaches are possible to build different levels of VN by virtualizing and slicing one or both planes. The control plane handles network control applications such as routing protocols like ICMP. The data plane handles the forwarding tables and the flow rules for traffic forwarding decisions. Another approach is to virtualize both planes where each network has its own control and data plane (slicing – strong isolation) [65].

The benefits of using VNs are [9]:

- On-demand customized network launching, establishment and creation.
- User defined requirement network creation: bandwidth, delay threshold, security feature and specific protocol supporting.
- Realtime connected network reconfiguration.
- Realtime connected virtual component migration

As an extension of introducing VNs within the cloud, beyond the above list, cloud service capabilities have been enhanced in two ways; the cloud services to user connectivity and the geographically separated cloud service interconnections [9].

Parts of the user requirement flexibility given by cloud networks that users can define [9] are:

- The network and computing requirements.
- The networking assets and the way it is configured for cloud resource connectivity.
- Their infrastructure component location, and how to connect them or to them.

All of these should be easily accessible by users through one comprehensive interface that is friendly and clear, complete with reporting. Other services were added for scaling, reconfiguring, and reclaiming unused resources by the cloud management system without user interaction.

VN of different tenants, possibly rival businesses, concurrent on a shared infrastructure should be prevented unless the isolation mechanism is assured and tenant data is not redirected to another tenant VN [66].

Different VN solutions have been implemented and each target may have specific requirements or demand better isolation [67] [68], better dependability[69], customer manageability[70], performance[71], and implement a collection of mechanism to offer enhanced IP address apace isolation [66].

VN architecture is a collaboration of network elements, procedures, functionalities, and policies. For example, VN must be isolated, routed, switched and NATted where the isolation could be implemented by different mechanisms or a combination of them, applied in different elements such as ports, bridges/switches, firewalls, etc. VN Isolation should enclose the communication and ensure traffic is directed only to the physical or abstracted resources allocated to a client network [31].

By establishing such a vast range of freedom and flexibility, a cloud could support heterogeneous systems with unlimited application diversity and changing requirements. This will result in an extremely dynamic system that is not appropriate for a centralized control approach. Therefore, distributed storage, processing and management control are deemed to be dynamic as well, rather than overloading the system and reducing performance and service availability. Consequently, a distributed service utilization control brought closer to the user is considered a better approach [9].

Although isolation is the ultimate requirement of virtualization, it is also considered to be the biggest challenge. **Isolation** prevents an interaction between two VMs in a shared environment, yet VMs are obviously connected to the infrastructure, other VMs or outside the cloud, and therefore isolating the VM by itself is not desirable. Hence, isolating the connection is another requirement. Therefore, a VN is an isolated logical network methodology in a shared infrastructure. VM hypervisors such as Xen and VMware closed the gap between VM isolation and VN isolation and secured the VM physical layer via VN mechanisms [72].

One of the earliest attempts of network virtualization was the Xen virtual router, and due to the client isolation in Xen, the virtual router operation does not affect the performance of other vRouters negatively [65]. VM hypervisors such as Xen and VMware closed the gap between VM isolation and VN isolation and secured the VM physical layer via VN mechanisms using logical dedicated channels and unique static IDs assigned as tags [72]. Essentially the authors isolated each VM using a dedicated host-VM link.

Several Virtual Network (VN) architecture frameworks have been suggested such as VINI , CABO , 4WARD VNet , and FEDERICA to provide on-demand, user defined and controlled VN facilitated by cloud providers [9].

FlowVisor is an example of network virtualization within a shared physical environment. It is based on the OpenFlow protocol and acts as a hypervisor to virtualize the hardware within an infrastructure and splits the control plane and data (forwarding) plane within the networking components [73].

Virtualizing a network is not merely a simple software configuration, it must take into consideration the underlying physical resource. Any component involved in the forwarding path needs to be virtualized and sliced. If the forwarding traffic happens in the software level then the CPU is involved in processing the software sessions and hence the CPU must be virtualized [73]. Other than FlowVisor, other attempts at network virtualization, with different strategies, includes those based on NOX: OpenRoads, PlugNServe, and OpenPipes [73].

As with computing virtualization, network virtualization needs a virtualization layer injected between the hardware and the software with a set of control instructions to control the operation between the software and the hardware to be shared by different users, applications, process, etc. In FlowVisor, a component called a programmable controller controls the switch traffic forwarding between the software level and physical level of the network [73].

FlowVisor is a network hypervisor that is resident in between the control (controller) and data (switches) planes. FlowVisor extends the control of a network by capturing the traffic between the data and control planes then checking then to confirm that no other similar traffic from another application is interfering with them. However, FlowVisor is not compatible with some SDN implementations which leads to inflexibility in network reconfiguration [22].

VLANs are one of the approaches to scale multi-tenant cloud networks, such as with the Quantum plugins in OpenStack . VLANs are mechanisms used to isolate different levels of multi-tier applications between different tenants within the same cloud environment. VLANS offer a better solution than Layer 2 network mechanisms as they provide an isolated broadcast domain but may delay the service response [21].

VLANs are one of the most popular isolation mechanisms used to support virtual network designs. VLANs depend on virtualizing the physical ports and slicing them into virtual links to create multiple Ethernet L2 domains. They enable trunking of multiple virtual links to go through a single virtual port. VLAN is a basic logical forwarding, while FlowVisor extends this with L2 forwarding algorithms to give it more flexibility [73].

Another addition to the virtual network components are software virtual networks as was researched in the VINI testbed [73]to expose the benefit of using virtual routers instead of hardware for programmability, configurability and throughput manageability.

Integrating active, dynamic on-demand user VNs is not an easy software plugin, yet it is vital to prepare and restructure the cloud infrastructure necessary to accept and support the new requirement.

The basic logic of introducing the user VN in IaaS is that the VN are mapped as one unit in IaaS regardless of the physical resource pool underneath and its geographical location within the cloud Infrastructure. There is no need for the provider to have knowledge of the VN infrastructure built by the user in the cloud, other than its infrastructure components [74].

A standard service is where any tenant can design, configure and manage their own network infrastructure in the cloud to their own requirements. Adding VNs allows accumulative communication segregation, service performance monitoring and reporting [75]. However, it added another challenge in terms of the management complexity and this was escalated with service automation.

2.7 Network as a Service

Network as a Service (NaaS) is a concept introduced to provide service connectivity offered by the cloud network infrastructure though a virtualization facility layer. Such a service covers the gap of the non IT-knowledgeable user and network on-demand connectivity creation through a **Network Virtualization Platform (NVP)** [7].

With NVP the main aim was to enhance the Internet and Cloud Computing facilities throughout the adaptation of Virtual Networks to provide dynamic, logical network topologies that are manageable, flexible and cheap [76].

An early implementation of Openstack Network as a Service (NaaS) is the Nova-network which has a simple isolation mechanism handled by different network managers such as FlatManager, FlatDHCPManager, and VlanManager. The isolation is applied at the level of vSwitches/bridges with only one default gateway for all VMs. The basic OpenStack network is referred to as a flat-network and it is physically mapped in a compute node, yet tenant traffic is not isolated and all share the same IP range. A better network was when VLAN was used and implemented on one VN element (bridge), but that raised the scalability issue and was not applicable to other VN/NFV services such as a firewall. With the Quantum module development, the current VN requirement was successfully attained especially with the support of SDN and the plugins [63].

Network as a Service with an OpenFlow controller as a plugin was used to strengthen cloud networking using SDN/ OpenFlow technology in order to provide cloud customers with logically isolated networks under management and configuration privileges. This solution requires an OpenFlow controller specifically designed as cloud networking plugins to avoid the lack (scalability) of the existing OpenFlow controllers such as NOX and FloodLight. The first cloud that would have the ultimate benefit of this proposal was the OpenStack Network service (Quantum) [21].

CloudNaaS was a solution proposed to allow an on-demand, dynamic cloud networking service. The main aim was to facilitate Virtual Network security via isolation and adaptable inserted **middleboxes**. CloudNaaS was based on the OpenFlow protocol that gave it the ability to establish the networking roles without dropping the infrastructure performance even with a considerable failure of provisioned services, devices and links [77]. The main functionalities of CloudNaaS services are to utilize network services to connect to their implemented servers in the cloud, effect network isolation and tenant defined network ranges, and install any middleboxes for intrusion detection, load balancing, etc [56]. It also permits users to configure their network devices with a monitoring facility with reporting and logging [77]. *However, the CloudNaaS evaluation was more focused on QoS than on isolation and security aspects.*

The **Cloud Networking Gateway (CNG) Manager** provided “dynamic on-demand intra and inter cloud networking” on a separated service cloud system which dealt with the cloud network infrastructure technology used by the cloud provider(s). With CNG Manager [78] the cloud user had a choice of technologies and function options to implement and program their own virtual network structure.

Virtualizing a physical network gives a cloud service heterogeneous features. In order to achieve a Quality of Service (QoS) between cloud service communication isolation in a virtual cloud network, a virtual router must be created and configured with various networking protocols and connected to the required service as suggested by Ahn et al [76]. Virtual networks require a management system that manage virtualized network elements, such as vRouters [76].

Part of the duty of the virtual network controller is to manage the bandwidth of the traffic based on the amount of bandwidth given to each virtual network. Bandwidth requirement assignment or re-assignment might differ from one network to another depending on the cloud service assigned to that network. This is where performance isolation is achieved [76].

2.7.1 VN Technologies

Tenant VMs are connected to each other with Layer 2 networking functions. Different tenant VMs are isolated from each other even though they share the same IaaS environment and use Layer3 functionality to route their traffic to be forwarded outside the cloud. These specifications are added to the cloud as networking services by adopting virtual network technologies such as

VLANs, VPNs and overlay networks. **OpenStack** is a good example of implementing virtual network technologies as a cloud network service [62].

Layer 2 switching with VLAN technologies is one way of cloud Virtual networking implementation. **Linux Bridge (LB)** and **Open vSwitch (OVS)** are the most common Layer 2 switches used in cloud computing. Both switching technologies are software-based switches within the system kernel; LB is a native kernel component and OVS uses **OpenFlow** based switches to utilize the kernel performance. Both virtual switching facilities are used by the hypervisor to provide the connectivity of the VMs virtual Network Interface Card (**vNIC**) to the cloud's physical network [51].

Control plane SDN-enabled compatibility is a vital requirement for any virtual network technology to be implemented in a cloud computing network. A widely used example of such a technology is **OpenFlow**, where it provides the networking service to host the tenant network manageability and programmability through protocols [62]. OpenFlow is the most widely used SDN standard protocol, used by both researchers and vendors [61].

Another of the cloud computing network technologies is called **server-side network virtualization**, which is applied by the cloud host hypervisor. Additional virtual network layers are added to provide Layer 3 communication, such as network **namespaces**, as a virtualization tool technology to ensure virtual network isolation. Another mechanism could be added to ensure interconnection between VM virtual interfaces to the cloud physical network by using Layer 2 Linux bridges. This approach reduces latency by managing the local traffic within a cloud computing host without involving the external network devices, but it might not be adaptable enough to satisfy traffic isolation. An OpenFlow-based switching technology called OVS satisfies the isolation goal as well as kernel-level performance when layers are added or replaced. This approach is commonly adopted in OpenStack and CloudStack IaaS clouds [62].

Callegati et al [62] claimed that server-side VNs compromise the security and isolation needed by multi-tenancy in cloud environments. Hence, they suggested using the Single Root input-output virtualization (**SR-IOV**) technology by having external switches working with a Virtual Ethernet Port Aggregator (**VEPA**) that will receive and process all the traffic, including local, to guarantee a better performance, monitoring and filtering of the cloud traffic.

Cloud Network APIs are cloud infrastructure dependent and are used to deploy, integrate and manage the network configuration and technologies established in private or hybrid clouds such as defining NICs, VLANs, OpenFlows, OVS, VPNs, etc [56]. Cloud network infrastructure services should combine management of the virtual network resources and any legacy networks. It should provide application layer functionality for the tenants to flexibly select their network requirements and manage them. Moreover, such services handle the communication between the cloud controller management and the tenant component interfaces for cloud computing module efficiency [56].

Cloud computing technologies and terminologies may have become confused as they have been used differently for different frameworks. For example, SDN was introduced as a networking service for DC/cloud computing, where SDN was meant to provide the cloud framework and network orchestration through applications and functions. However, Network Function Virtualization (**NFV**) was introduced for network providers to provide network functionalities such as routing, gateways, firewalls, etc. Today, cloud computing uses both SDN and NFV to provide cloud networking services as part of service provisioning besides the compute and storage services [2].

2.7.2 Network Management

Decusatis and Cannistra [79] proposed a network management layer, which offered the following:

- Diverse APIs existing on the same layer.
- Each VN assigned different controllers from different vendors without conflicting with other controllers in other VNs.
- A network management upper level APIs to guide the controllers in the lower levels of the network structure.
- Supporting the vertical hierarchal structure and hence improving delay resolutions between network components and controllers.
- Functionality specialization in different levels to simplify the network complexity operation, e.g, upper level controller focussed on functionality such as re-routing flows throughout the network, while lower level, localized controllers concentrated on similar size flows between the network devices.

2.7.3 Customer Requirements

Clients in a shared environment should not be aware of the co-existence of other clients. However, in reality, tenants are aware of the existence of other tenants. Even if they are not visible to each other, and a trust agreement is usually agreed with the provider. Providers must ensure a justifying level of resource isolation between clients, and clients now demand better and provable isolation, with isolation failure measurements based on environmental service characteristics such as privacy, performance, robustness and security [80].

Cloud network researchers adopted three categorizations to describe the cloud network infrastructure solutions; traditional, converged and cloud-only [81]. Traditional cloud networking uses a basic approach via logical separation using VLANs between the hosts within the hypervisor environment. A converged network designs the IP network and the storage network in conjunction with the physical underlying network. The Cloud-only category uses the physical environment as a resource and segments it logically using VLANs and ports grouping configurations to form the logical customer network segmentation. This delivers an on-demand customer network and dynamic network configuration.

Unlike traditional networks, where the edge of the customer network is well defined by its demarcation point, cloud structures are different. In a cloud, the customer network **perimeter** is determined based on the tenant's virtual network components, that is, the limit of the tenants and the traffic flow managed and processed within the cloud. That led to uncertain definitions of the network border/edge/ perimeter in cloud systems, as the network design is dynamic with regard to the tenant's easily changing requirements [81].

Customers are attracted to on-demand, self controlled service hosting because of the benefit of reduced costs. The customer requirements that pushed developers to invest in cloud computing and gain the full benefits of cloud computing are stated by Schoo et al as [57]:

- The need to support applications which need to be connected to the user smoothly, as well as managing the dataflow between different cloud sites.
- The need to deploy applications that build interactive features, which demand lots of bandwidth, and strive for capacity, quality and service availability.
- The need to migrate live applications between different cloud sites, which need a flexible network environment with more advanced network infrastructure features.
- The need for dynamic, promptly reconfigurable customer networks that are service procedure establishment free.
- Ensure the on-demand provisioned network resources are compatible with the cloud computing resources provisioned to the tenants.

The goal of introducing cloud networking supported virtualization technology is to allow users access to cloud services and connected cloud services even if they are geographically distributed.

With this, a major cloud infrastructure transformation is a must; the new infrastructure should give the user the ability to select their choice of the virtual infrastructure and network components, and select their infrastructure components location within the cloud along with the interconnection of those components in a "dynamic, on-demand, single control interface" [57].

The most challenging type of traffic for VN are those produced by gaming services and applications, as they critically require the lowest low latency but highest performance with extreme resource isolation [38].

2.7.4 Cloud Networking in CloudStack and OpenStack

CloudStack is considered to be a stand alone cloud network service structure solution that provides a list of networking services to fulfill most cloud network requirements.

OpenStack is a cloud stack that adopts CloudNaaS principles. CloudNaaS provisions network services associated with the data plane such as traffic isolation, QoS, etc. Openstack is one of the widely used open source IaaS cloud modules[77].

According to Mechtri et al [78] OpenStack Cloud Computing is one of the earliest cloud stacks to develop the use of Cloud Networking though the Neutron framework, while other stacks concentrated on assigning VMs with static or public IP addresses and without further networking facilities.

A cloud network in OpenStack is an API based system that provides external (floating) IP addresses with different approaches to assigning tenant network type options e.g Flat, VLAN, etc and user on-demand network management and creation, SDN/OpenFlow support and plugging of network features such as firewalls, load balancing, etc [44].

Through a dashboard the user has the flexibility to control their allocated services in the cloud such as storage and virtual network infrastructure facilities [51]. OpenStack has a web-based dashboard allowing the tenant to manage and control the resources and services provisioned to them, such as the compute and network facilities, with great speed, flexibility and transparency. OpenStack is designed to control clusters that consist of controller nodes, compute nodes and network nodes, which prevision the network services [49][62].

Vaughan-Nichols [82] stated the importance of deploying OpenFlow in an organizational network to reduce implementation conflict in a network. Moreover, public clouds will implement OpenFlow (rules) as their main network configuration requirement, and this is used in currently deployed cloud frameworks such as OpenStack and Cloudstack [82]. OpenFlow become the official protocol of SDN and the market is heavily loaded with devices supporting

SDN/OpenFlow[83].

As network virtualization has become an aim of the network community, researchers have been heavily invested in developing mechanisms to segment and isolate hardware forwarding paths in commercial equipment, in order to achieve network virtualization. This is a physical environment, with multi-tenancy through logical isolation with locally configured addressing schemes and traffic forwarding rules [73].

2.8 SDN Management

An SDN controller is a software controller defined as a network hypervisor to isolate the tenants network traffic. The benefits of SDN controllers [60] are:

- Less switch memory utilization during the forwarding process.
- **Tenant** network isolation.
- VM and VN configuration and management throughout the network hypervisor in OpenStack.
- Expansion and configuration of new network functions become easier.

Bozakov and Papdimitriou [14] introduced an advanced terminology of SDN called SDN virtualization, or vSDN. The idea was to virtualize the SDN platform then lease the slices to the service provider who would then lease VN slices to the end users. SDN isolates the traffic of different VNs, however vSDN isolates the flow space where each vSDN has its own isolated vSDNs [14]. Traditional SDN provides isolation and protection to the data plane between the multi-tenancy VNs. vSDN provides isolation and protection for the data plane in the physical layer as well as the SDN Control plane [84]. **OVX** is a form of vSDN and is implemented in the OpenStack Cloud as a plugin.

As SDN functionality depends on the abstraction of the physical layer and SDN is a standard technology, there should be a way to adapt different vendor proprietary physical hardware embedded in a heterogeneous system. An intermediate layer of communication, an API, is provided by the vendors as an interface[24]. The SDN framework is extended via the use of open source APIs to allow the interaction between the system applications, physical environment, network component and other services [50].

SDN supports two types of network approach; the Traditional SDN and the Hybrid SDN. The Traditional SDN focuses on the network operations control/data plane communication without being concerned about any virtualized networks services. Such structures are flexible but not scalable, therefore, the best application of such a structure is in a system that requires great elasticity [24].

Hybrid SDN use the same strategies as the Traditional for operating the network by separating the control from the data plane, yet differs from the Traditional SDN which uses a single centralized controller as the network operational control point. The Hybrid SDN uses the data and control plane as well as the network appliances and the devices are managed by their own control plane. Also it functions separately from the central SDN controller. The communication between the network applications and devices and the SDN central controller is via an API. This approach provides not only elasticity, but scalability and accessibility, due to the operation of the network being distributed amongst the individual network device controllers as well as the central controller. If the network lost the connection to the central controller, the functionality is handled by the other controllers using their current configuration until the central controller is revived [24].

However, controllers are independently implemented without necessarily following standard "northbound structure" (API) and hence controllers can not always work together in the same environment [79].

SDN is designed for wide range, complicated network systems, where policies and flow rules may be reinstalled or reconfigured and this is only possible via the separation of control and data plane. However, routers and switches are only responsible for receiving and forwarding the traffic based on the flow rules installed/defined/configured by the control plane. Hardware such as switches and routers with SDN capability are configured to allow the control plane to send instructions of traffic forwarding and handling decisions to the data plane via the router/switches [61].

With the structure of the SDN central control, network adminstration gains more control on the network for defining the policies and rules of traffic handling as well as flagging the flow with administration defined categorization such as priority, allow or block. All of this is implemented in the central module, which means it saves time on configuring each network device connected to operational infrastructure [61].

Clouds adopted the Hybrid SDN approach. For example, OpenStack uses network applications as networking devices configured to be stand alone components that handle tenant L2 and L3 traffic for the network components configuration. Moreover, vendors implemented their network facilitation to the OpenStack networking system in a form of plugins, which perform port creation and VLAN isolation. OpenStack has its own network control module called Neutron that handles the SDN networking operation and implements the routing engine to facilitate communication

[24]. Neutron is connected to the Compute database to keep network related information, agents and plugins for services and appliances [85].

An example of a network hypervisor module implemented in OpenStack is called DCPortals or DCPortalsNg. This module gets the required information for OpenStack DB such as the tenant ID, VMs info and location, VNs, etc, and uses this information to create flow rules and inserts them into tables in the OVS switches in order to handle the traffic flow. The advantage of using DCPortals is not to be concerned about OpenFlow compatibility in the physical network environment of the cloud [60].

2.9 Virtual Network Isolation

Virtual networks cannot run alongside each other without isolation. In a perfect world perfect network isolation will exist where networks are endlessly independent from each other on shared environments. However, absolute network isolation, with all the benefit of performance, scalability, flexibility, etc, is not feasible with current technology at the current time. Hence, a lower than perfect isolation is what is currently feasible [13].

Isolation is not as standard as routing, which can be configured in any network L3 device in any topology, it is similar to network services such as **QoS**, policies or **ACL**. It is a network topology dependency implementation and that is why isolation mechanism implementation and management are challenges [20].

Isolation of network virtualization can be categorized as resource isolation or security isolation, as defined by Kanada et al [19]. They define resource isolation as a function that enables a slice to use the required resources to provide the expected performance even when congestion occurs on the physical network or on other slices. Security isolation is defined as a function that avoids disruptions and intrusions against a given slice caused by adversaries.

One of the main criteria of network virtualisation is the **isolation performance** between the virtualized entities. One implementation is to manage the traffic queues of the exit ports, except that the performance might negatively be affected if several entry ports forward the traffic to a single exit port and the bottleneck effect is un-avoidable. Therefore, doubts have surfaced with regards to the isolation level of virtual network implementation. The implementation variables of the virtual network isolation include buffers, rate limits, and queuing on the outer ports and exit ports of the physical shared environment [86].

Address space and flow space isolation are two isolation mechanisms that are used as logical isolation in virtual networks. Those mechanisms are vital for the flexibility and the scalability of managed multi-tenancy network structures. Although the obvious challenge of the virtual network is isolation, researchers are concerned with two main aspects; performance isolation and security isolation. Even with aiming for performance isolation, there must be shared connections which results in a reduction in performance isolation and raises problems with resource equality consumption [18].

Cloud tenant network performance isolation was supported with several suggested solutions e.g. providing “eastbound interfaces” between SDN controllers, and adding additional layers in the cloud for tenant orchestration. Another solution was to add a virtualization layer that provided the tenant with full control over their network that is projected over the real network. The most popular solution was to abstract the virtual network via SDN for network performance isolation [18].

Virtual network isolation is more likely to be altered due to certain systems implementation, traffic jamming or VN overburden functionality. Solutions that depended on the VN traffic process scheduling in a multi-core framework gave better performance against bottleneck incidents. VN implementations usually come with a list of aims to achieve performance, isolation, flexibility, and fairness [86].

A traffic isolated network is done via network partitioning wherein a physical layer physical port is dedicated for specific traffic, However, the partition is logically configured via VLANs and virtual interfaces. Moreover, isolated traffic needs to be placed, along with a vNIC, within the logically isolated networks [15].

Network isolation management is a gap to be investigated in order to move the network status from the manual to automating the isolation of network devices configuration. Another gap is to solve the isolation through programming the flow policies and network segmentation implementation. Platforms to tune the software level network isolation solution is another gap to be considered, as there is no standard platform for a universal programmable approach with compiler support [22].

Domain isolation is when the provider implements the multi-tenancy by isolating groups of resources as a domain and allocates a specific domain for a tenant. Each tenant owns a domain and is not aware of other tenants in the shared environment [80]. Considering the high cost of the isolation for the service provider, it is understandable that they apply a partial isolation on their infrastructure [80].

Much research in VN isolation and security is concerned with how to ensure the controller better implements the isolation mechanism and protection approach, creates efficient isolation between network applications and controllers, and develops security domains and data access protection mechanisms to protect network functions from any security risks [64].

2.9.1 VN Isolation Approaches

The original idea of VN was to enhance the Internet with flexible demands such as elasticity, manageability and isolation and use it as a mechanism in the cloud to empower traffic isolation, enhance security and provide lower costs [75].

Traffic isolation network mechanisms might differ from one environment to another, for example, in a Hypervisor cloud environment, it is usual to use virtual bridges as Layer 2 switches to forward traffic based on rules and policies to ensure a secure traffic isolation [15]. A variety of solutions have been proposed with different mechanisms such as isolation via network processor, isolation through logical programming and network processor scheduling [35].

The establishment of VN encouraged several vendors to upgrade and redesign their products, such as Cisco, Juniper and OpenFLow. One of the Cisco products called CRS1/16 was a hardware-based isolation virtual router which implemented firm isolation in the data and control panel and guarantees fault isolation [38].

Gutz et al [22] state that maintaining powerful boundaries between networking portions in a shared environment will enhance security around running applications and that required configuration of inner security criteria would not be needed [22]. For example, a Python daemon, which manages user requests (and exposes the API) is configured with a plugin that implements OpenStack Networking API operations using a specific set of networking mechanisms. A wide choice of plugins are also available; the open vSwitch and Linux bridge plugins utilize native Linux networking mechanisms, while other plugins interface with external devices or SDN controllers [85].

Security isolation methods are not VN isolation but a layer built on top of the network isolated traffic for reasons such as using SSL or HTTPS for authentication purposes [15].

The justifications behind introducing different solutions for network virtualization might differ, for example NRI aimed to solve the issue of the network slices intrusion via “per-slice queuing”, Other projects which developed virtual networks as slices were PlanetLab [87] and GENI [19].

Kanada et al [19] promoted a solution to a straight forward isolation, called a per-link policy, where several slices passed their traffic using a single queue controlled by carefully defined policies.

Introducing network hypervisors such as FlowVisor that are resident between the control and data plane in SDN, overcome some of the isolation mechanisms lacking a programming solution for network isolation such as NOX and NetCore. An SDN hypervisor is designed to provide isolation mechanisms as well as giving developers and researchers the ability to edit, modify or customize any feature or program to fit with the network requirements. Although FlowVisor is a better solution, some research has found several flaws that require better resolutions [22].

The term *slicing* the isolation can mean isolating the application communication from another application or isolating a specific communication type in applications from another communication type in the same application [22].

Another solution is via network slicing for programmable isolation. Moraes et al [17] defined slices as two slices in a shared network not being aware of the other's existence. Slicing criteria must contain integrity, confidentiality, and isolation throughout the slice. Firmware is another way for slices to filter the non related traffic and hence enhance the slice isolation [22].

Several network isolation solutions have been implemented differently [22]such as:

- VL2: application level isolation using encapsulation and tunnelling for traffic forwarding.
- NetLord: virtualization and encapsulation.
- CloudPolice: security imposed on host architecture for customer isolation.
- Seawall, SecondNet, and Oktopus.
- SDN:
  - Controller modules: Beacon and Maestro, NOX, POX and Nettle.
  - SDN hypervisor: Onix and FlowVisor.

DCPortalsNg used a traffic rewriting mechanism to isolate tenant VN, which results in protecting traffic from other tenants and better performance as the load on the network hardware device will be lower as the physical addresses of the system VMs will not be processed all at once [17].

HP labs were the first to attempt a VN isolation solution and introduced Trusted Virtual Domains (TVDs)) to develop VN isolation fully handled by VMs. Other solutions use VLANs and EtherIP, while DCPortalsNg used an SDN model without the need of OpenFLow enabled hardware to implement a VN isolation[17].

The use of vRouters in VN will enhance isolation but does not provide complete isolation. Hence, more solutions for VN isolation are still needed [76]. Vrouter abstraction was originally a logical portion of the physical router [88]. Router virtualization does not mean a full isolation, however, a router can be virtualized but shared by different tenants for a specific resource sharing. A vRouter could have a shared session. The degree of virtualized network elements isolation is controlled based on the requirements and not by the fact they are virtualized [88].

The Juniper approach is to use Router Engines for physical resource isolation between vRouters and implement a broad protection [38].

Several isolation solutions use L2 isolation such as EtherIP and GRE encapsulation protocols, however, those are applied in a private network with low performance isolation and static bandwidth provision. To overcome this an L3 isolation is required [76].

For reliable VN with entirely diverse features to simultaneously run in the same environment, two requirements must be guaranteed; VN isolation and adaptable performance. Ahn et al [76] proposed a platform that guarantees VN definition, efficient QoS bandwidth supervision, VLAN ID network isolation and performance isolation. Performance isolation is a method to supply, dynamically allocate and manage bandwidth between isolated VNs. Ahn et al [76] used "BCN, weight-based bandwidth allocation, and virtual-channel bonding" to ensure the isolation performance for its proposed solution.

Rathore et al [89] stated that an isolation mechanism is the best solution for complicated continuously growing networks in order to confirm just resource consumption between the shared network tenants. They also established a Virtual Distributed Ethernet (VDE) mechanism software solution to increase isolation performance between cloud users in the Eucalyptus cloud private network.

DCPortal was one of the attempts to provide a physical shared network abstraction with multi-tenancy network logical isolation. To ensure network isolation between tenant traffic and lower the overhead of the physical network structure from processing all the VMs at once, a packet rewriting method was an appealing solution [60]. DCPortals was tested on a DC and was implemented in Openstack clouds with the Quantum module for VN provisioning.

Data Center resource isolation is not sufficient for cloud storage and compute multi-tenancy. The framework needed to be backed up with network isolation, and Brassil [80] proposed a physical layer network isolation.

One of the earliest attempts of VNI is by HP Labs, where the VMs perform and manage network isolation using three methods; VLAN tagging, EtherIP encapsulation, and MAC rewriting. These solutions lack scalability and need more work on the hypervisor reconfiguration [60].

Virtual networks depend on isolation and performance. To ensure that parallel networks are not negatively affected by virtual router misconfiguration or attacks, isolation must be implemented properly. XNetMon proposed to overcome the lack of isolation and performance of VNs in a Xen environment. XNetMon focused its research on virtual routers to provide flexible and secure isolation with high performance network [90].

FNS is a Flash Network Slice which is a concept associated with the CloNe project for Dynamic Virtual Network resource and service provisioning. OpenFlow is used in FNS to ensure the isolation via the controller by determining the flow rules and actions for the NFS traffic by using exclusive packet header fields to match [55].

Neutron Openstack accepts plugins to provide isolation at the virtual link level in order to perform instantly adapted traffic routing [91].

DVN is a solution to ensure cloud VN isolation among tenants, but is more effective in distributed cloud sites. DVN provides isolation on two levels; a link level and a network level. The cloud tenant network is ensured isolation throughout the cloud environment and through the different layers of the network operation at L2 & L3 [92].

Solutions such as DVN proposed to do the same for network service as SDN provision in the cloud. Previous solutions such as NDB [47], OFRewind [77], Anteater[55] and HSA [9] suffered from fundamental deficiencies in exposing critical infrastructure information due to inappropriate implementations. Tunnelling protocols in virtual switches such as VNGRE, VXLAN and STT were proposed as encapsulation methods to strengthen tenant isolation. In DVN the cloud controller schedules and isolates (with tunneling) the bandwidth and tenant traffic to preserve the cloud management data and platform [74].

An L2 isolation solution is by using GRE where each tenant will be designated with its own GRE tunnel and is not sharable with others. L3 isolation is created by configuring the Gateway node to give each tenant their own IP subnet. This would be included in the firewall rules for packet routing and forwarding, and for preventing exchange packages between other isolated networks [92].

A network slicing mechanism not only provides VN isolation but it gives the tenants the flexibility to customize their own network topology. Solutions such as FlowVisor [54], OpenVirtex [55]or VeRTIGO [74] are successful. Rather than open the shared resources for everyone, Jacob et al [52] suggested to group tenants involved into a domain which would then be isolated from all the other tenants. They also suggested that OpenNaas have the ability to create a multi-domain that would group users to use specific shared resources per to their request. They called their solution Slice Oriented Multi-Domain SDN.

VIOLIN is an application-level virtual network architecture, where isolated virtual networks are created in software on top of an overlay infrastructure (e.g., Planet-Lab). VIOLIN logically isolates the network targeting different sides; administration, address space and protocol, attack and fault impact, and resources [36].

Genesis is a spawning network which allows multiple heterogeneous child virtual networks to operate on top of subsets of their parent's resources, and provides isolation among them [36].

FEDERICA aims to provide an agnostic and transparent infrastructure, which supports isolated, coexisting slices with complete user control to the lowest possible layer [36].

VINI synthesizes container-based virtualization technologies together with a tunnelling mechanism into a coherent platform to achieve design goals of performance, scalability, flexibility and isolation. It allows each virtual network to define its custom topology, routing protocols, and forwarding tables [36].

UCLP developed physical isolations on a optical physical network environment. Physical isolation targets the environment with higher security demands. A resource scheduling algorithm is another isolation methodology some providers use to give the impression of isolation within the networking devices [36].

In cloud environments, in order to insure multi-tenancy applications, controllers must verify the correctness of the performance isolation throughout the cloud infrastructure [61].

textbf FlowVisor is midway-virtualized controller that is responsible for the network slices policy and handles the slices and controller communication to ensure isolation [55]. FlowVisor is a set of language abstraction tools, implemented to create vSDN blocks and create separated flow space after the switch flow tables have been divided. FlowVisor handles many large tasks in SDN topology representation, reducing the burden with vSDN mapping, configuring the tunnelling flow rules while maintaining the flow table isolation [61].

FlowVisor implemented slice isolation by configuring the traffic header with a part that is only defined for the corresponding slice path flow. **FlowN** is a Cloud solution for resource isolation with logically customized controllers for each tenant. FlowN used another isolation mechanism on the address space by encapsulating the client packet headers with VLAN information [93].

Although network virtualization is a standard definition, the strategies differ depending on the approach the developers selected. OpenFlow and FlowVisor are well known, though they were not flawless for manageability, isolation, flexibility and QoS. Flowviser controls only the data plane, which means the control and network planes are out of its responsibility, and hence the platform user must find a solution for this gap. OpenFlow also suffers from standard management tool support [61].

Hu et al [61] stated that FlowVisor creates slice based ports on switches, IP address schemes, physical addresses, etc. The slices are administered and assigned to different controllers dynamically and physical infrastructure multi-tenancy is provided via sharing network hardware in a form of VN.

FlowVisor offers full isolation with VN slices and is considered to be a middle layer between the data and control planes. An OpenFlow switch and a controller are connected at an intermediate layer, the network hypervisor. A slice here is defined as a set of flows running on a topology of switches, which are isolated but may overlap if this is desired [58]. FlowVisor applied virtualization technology to create software switches to be part of VNs and implemented isolation as well as physical traffic forwarding. Moreover, this solution did not require dedicated switching hardware, specific programming tools or network CPUs [73].

Flowspace is a header that defines slices. Sherwood et al [73] state that "The set of flows that make up a slice can be thought of constituting a well-defined subspace of the entire geometric space of possible packet headers." Moreover, " FlowVisor defines a slice as a set of flows, thought of as being defined by a set of (possibly non-contiguous) regions, and called the slice's flowspace.

FlowVisor ensures that network controllers are not aware of the virtualization system, slices must be solidly isolated, and slices are characterized by well configured rules. FlowVisor invented dynamic active mechanisms for monitoring, modifying and setting regulation on the transferred messages in order to confirm the slice's isolation and transparency. As FlowVisor aimed for deployment standardization, its design led to different isolation mechanisms being adapted such as Bandwidth Isolation, Topology isolation, Switch CPU Isolation, FlowSpace Isolation, Flow Entries Isolation and OpenFLow Control Isolation [73].

FlowVisor implements Bandwidth Isolation through the use of VLAN and edited forwarding tables with flow rules entries to process priorities as set in the VLAN priority bit and executed accordingly. However, despite the effort and the improvement in bandwidth isolation, it can not be claimed as a full QoS achievement [73].

As FlowVisor is built on OpenFlow technology and each slice used its own controller, OpenFlow control isolation was a necessity through abstracting the control path and isolating it [73].

ADVisor is similar to FlowVisor, as it is also a software solution. No control plane isolation mechanisms were proposed by this work. VeRTIGO adds more flexibility in provisioning vSDNs, and also increases the hypervisor complexity. However, it does not address the Control plane isolation limitations of the previous solutions. Auto-slice is a solution targeting software deployment via a partial control plane offloading and could be offered by distributing the hypervisor over multiple proxies. However, within each SDN domain, the control plane isolation problem still persists. In OpenVirteX the full flowspace could be provided to each tenant. The hypervisor layer is still software and extended by edge switches. However, no isolation schemes were discussed in these schemes [84].

AutoSlice aimed to create network resource slices, such as a topology forwarding table, bandwidth, etc, with variable controller delegation for isolation confirmation [94].

Once traffic or a specific category of traffic has been dedicated to a certain VN, this is a traffic isolation implementation. In vSwitches, flow tables contain lists of forwarding policies, which must be isolated between slices otherwise the traffic forwarding will fail [73].

SDN controllers such as Beacon and Maestro (based on Java), NOX and POX (based on Python), and Nettle (based on Haskell), were implemented for SDN networks but they did not solve the issue of isolation virtualization. FlowVisor and Onix were better but they lacked accuracy and security [22]. SDN technologies and any protocols and programs are still maturing and considered to be on going work. *Security aspects, and isolation in particular, are not as well developed or refined as cloud QoS or flexibility and hence is an area requiring more research. Hence, the security aspects are not even stable enough and require further research.*

For VM isolation in the cloud, a best practice recommendation is to use a combination of mechanisms such as VLAN, firewall, IDS/IPS as well as data classification and policy-based management [32]. Further, the cloud provider should create an evidential report in case of an isolation break.

Bouayad et al [39] declared that multi-tenancy and isolation is one of the “open research problems” in cloud computing, alongside with “vender-lock-in, data management and cloud security". Isolation and multi-tenancy are not issues to be taken lightly as they extend throughout the cloud layers (SaaS, PaaS, IaaS) and such challenges require a solution throughout the three layers in a cloud [39].

One of the overarching questions in VN is how isolation is confirmed and specified in a network abstraction. Gutz et al [22] have attempted to resolve this on a software level via programing slices but are still developing semantics for confidentiality and integrity.

Maintaining a virtual network isolation on a traditional physical layer is a challenge by itself, yet configuring the same on a wireless environment for mobility and unbroken connection adds another level of challenges [91]. *Hence, for this thesis a decision was taken to focus on non-wireless environments.*

2.10 Isolation Configuration Issues

Isolating the traffic is not enough to have isolated VN, other VN resources must be isolated such as links, flow tables and rules and CPU and address spaces [95].

Splendid Isolation [22] has been proposed as a means of verifying the isolation of program traffic across network slices. It is a formally proven programming abstraction for defining slices.

In some cases static isolation is preferable to dynamic isolation when a specific requirement is demanded such as a greater performance and simpler isolation such as offered in Splendid Isolation [64].

It is not feasible and overly expensive to have all the VMs of a single tenant located in a single physical server or connected to a designated NIC that is connected to a designated switch for that tenant. It is vital to have a strong isolation solution for a multi-tenancy infrastructure while maintaining the flexibility and mobility. A cloud provider such as Amazon uses their physical infrastructure to abstract the resources and logically isolate for multiple-tenants that share the same environment.

El-azzab et al [96] researched balancing the isolation with performance with respect to the required demands, by providing different levels of isolation with an evaluation utility to harmonize the performance and guarantee flexibility. The biggest challenge in VNs is isolation and the Data and Control planes require strong isolation. The downside of strong isolation is that the performance and flexibility is sacrificed, the bandwidth is overloaded, and communication delays occur.

An Isolation Model can implement isolation levels according to tenant performance and flexibility preference. The isolation model of El Azzab et al [96] responsible for defining one of eight isolation levels, and instructs the isolator to identify the packet slices that are built based on the interface, memory and process resources.

Although the model provided three isolation mechanisms, the slices are created by the compilation of the three mechanisms and hence there are eight isolation levels [96]. The three isolation mechanisms are implemented differently; the isolator inspects the interface status in interface isolation, it acts as a firewall by filtering the traffic in the process isolation, and it guarantees buffer and queue allocation per slice. Moreover, there is a utility function to examine and stabilize the performance and flexibility in each isolation level.

VNs use different isolation mechanisms such as a low-level mechanism such as VLAN to perform traffic isolation, or via a special purpose device like a firewall for physical isolation, or a complicated hypervisor layer solution such as FlowVisor to implement control isolation. However, each mechanism carries some issues; VLAN makes the network configuration denser and a firewall will carry its vendor proprietary adjustments. FlowVisor adds another layer of virtualization which is risky as it slow. Hence, Gutz et al. [22] believe the way to overcome such issues is by approaching the isolation at a higher level with instinctive programming languages and supported by compilers.

Similar to FlowVisor, XNetmon provides isolation as a hypervisor solution. Other systems with different isolation approaches in a form similar to the OpenFlow virtualization structure are discussed by Casado et al. [20], Zarifis and Kontesidou [97], and Foster et al. [98].

The **CloudNaaS** solution is designed to fulfil tenant demand by creating customized isolated VN configured with their address scheme, VLAN isolation, and network service, applications and tools. CloudNaaS is a Cloud Network Service manager for the cloud network data plane to provide VLAN traffic isolation, middleboxes and QoS [77].

Amazon's attempt on network segmentation as a service to their clients used VPN with private address schemes and adaptable access control rules [77]. In the cloud commercial world, Amazon and Azure offer network specialized service provisioning including traffic isolation. Although VPN with adaptable ACL policies for their networks is not considered as a complete VN isolation method, it does secure the connection by traffic isolation, as has been provided Amazon for their customers [99]. Unlike VN , VPN does not provide genuine isolation as it is a rather deceptive isolation [100].

An example of a private cloud platform is when an organization has different departments and each has their own specific performance and security requirements and resources that need to be implemented and isolated in a safe and controlled way to communicate or share defined resources [79]. The most obvious solution to achieve a private cloud platform is to apply isolation mechanisms on the cloud infrastructure to accomplish the multi-tenancy condition and ensure that the data flow is isolated across the entire platform. SDN based on the OpenFlow technology successfully applied such an isolation via its network controller, which dynamically isolated the tenant flow up to the physical layer [79].

Future Internet Testbed with Security (FITS) is a project that was built specifically as an experimental platform to provide network slicing, protected admission and QoS [101]. FITS is a resource for researchers to test their networking development projects. FITS emphasized its work on isolating VN and migration, so researchers could implement their network protocols on its L2 VN environment.

The FITS environment was built with a Xen virtual environment to provide resource abstraction, and implemented isolation via segmenting the network space address with VLAN. Traffic isolation was performed with a traffic forwarding mechanism with allocated queues and bandwidth for better resource management. In order to deploy the packet forwarding isolation mechanism "OpenFLow Global Controller" was used [101].

Traditional Data Centre networks are usually isolated in three levels of access, aggregation and core. Therefore, VLANs and vSwitchs are controlled by hypervisors such as VMWare and VM communication is isolated within its own VLAN [23].

In the Secure Elastic Cloud Computing (SEC2) environment, network tenants are isolated and use their own designated local IPs and Network address space. However, if the tenants wish to communicate with each other the proper way is to establish the connection using Forwarding Elements (FEs) with a NAT policy within a firewall and then processing the communication in middleboxes to complete the connection path. This way the isolation is intact and the security hazard is kept to a minimum. However, isolation breakage might occur, not because of the weakness of the isolation, but by compromising the other related entities such as switches, hypervisors, and middleboxes [23].

Bozakov and Papdimitriou[14] proposed a vSDN layer solution to organize the installation and administration of vSDN, using a hypervisor that produced forwarding entities for vSDN implementation and switch flow tables during the migration event. While vSDN management believes in self-government, behind the scene the SDN hypervisor modified the control instructions for a flowspace isolation guarantee. One of the SDN hypervisor features is **Control Message Processing** as the tenant is only aware of their own isolated network space.

As part of the isolation assurance functionality, the SDN hypervisor assessed the vSDN setup. Meanwhile the control proxy regulates the flow table rules to confirm that the flow entry mapping does not intersect with other vSDN flowspaces. By mapping the vSDN topology with the required flow entries, the manual configuration of the FlowVisor SDN operators is not necessary. In addition this project is the building block for implementing other platforms such as OpenVirtex [14].

One of the risks caused by cloud tenants is using their resource privileges in the cloud, such as VMs, to screen other tenants or any resources belonging to other tenants and so eventually cause an isolation break. Wu and Winer [72]declared that one of the network security risks in VMs is via monitoring other VMs or the host machine itself.

The safest isolation protection is to devote a physical communication path for each VM, which is economically not feasible. Isolation breaking events are likely to occur by breaking the logical isolation of the communication path [72]. Isolation of hardware or abstracted resources is a natural prevention against data leakages between isolated components in a shared environment, whether they are VMs or network entities [6].

VLAN is a good isolation method to protect tenants from each other, however, as the normal case suggests cloud instances need to communicate with each other or to the Internet and although a router facilitates this option, without a firewall for instance, the VLAN isolation does not protect users [30].

Diverter is an approach that applies network abstraction based on L3 communication to provision the on-demand isolated, multiple subnet customer VN [66].

A big portion of network virtualization is subject to vSwitches either by creating a hypervisor or a VM implementation such as Dom0 in Xen, in order to have authorized access, or resource and namespace isolation in which case vSwitches are required [102].

In terms of isolation and security, virtualization enhances the security isolation application responsibility, however, it might cause a potential threat if it is misconfigured [31].

Carapinha et al. [100] argued that vRouters provide the illusion of isolation to the VN, and must be replaced by virtual nodes which act as routers, and yet they are abstracted from physical resources and use the slicing methodology for isolation. The common way for the Infrastructure Provider (InP) to perform VN provisioning is by virtually slicing the network infrastructure via abstraction technology.

vRouter isolation adds the ability to migrate the router without negatively impacting other vRouters. During the migration of vRouter bandwidth isolation between the migrated traffic and data traffic must be intact [103].

For VN to achieve isolation mostly low-level mechanisms are used, such as VLAN for traffic isolation and a firewall for physical isolation. FlowVisor and **XNetMon** both shift the network isolation to another level by enhancing the OpenFlow-based VN feature beyond the VLAN and firewalls, and so overcome the flowspace complexity issue [2].

A solution for cloud network infrastructure involves network technologies such as Mellanox with the corroboration of InfiniBand, Ethernet interconnect and OpenStack services. In such scenarios, a system will be able to perform isolation and physical security as well as network, compute and storage provision with their exclusive services and functions. The Mellanox ConnectX-3 Adapter is used for Virtual function instance provision. With the use of vSwitch, security is performed by VLAN network isolation and anti-Mac spoofing , which prevents the Media Access Code address of a network interface being changed to conceal identity [104].

Decusatis and Cannistra [79] used an OpenStack cloud as a platform for their solution to maintain a multi-tenancy isolation system. They built a hybrid OpenStack testbed with multiple hypervisors, several SDN controllers and supplementary physical and network configuration such as a management network per tenant with firewall protection for multi-tenancy isolation maintenance.

When some experimental scenarios needed to take real world data, the best option was to run them in a production environment and ensure isolation from other tenants. However, the risk of a potential negative impact that the experiment might cause on the production network was quite high, and hence a proper isolated VN was the best solution for this case [38]. *We considered a production environment for our research but as we needed to expose the physical infrastructure and VN isolation we decided to build testbeds to answer our research questions.*

Conventional isolation was an issue in High Performance Computing (HPC), and to overcome this issue SDN and NFV were established with better features to define multi-tenancy via remote resource isolation. The demand was not pure isolation but a dynamic isolation, which turned out to be rather problematic to accomplish without SDN and NFV. A Networking API was needed to complete the isolation goal in HPC [24] with flexibility, performance and scalability.

2.10.1 Attacks

Natarajan and Wolf [35] considered physical resources isolation to introduce unknown attacks and therefore warranted a logical isolation between VN for enhanced management and security.

In NICE [112], security testing research was used to attack the VMs in the cloud by building a testbed with OVS/OpenFlow SDN network and VLAN. NICE is an agent application attached to the cloud controller to measure how relative VMs are attackable if there are in an isolated network and how the virtual networks adapt and reconfigure themselves to withstand a malicious flow.

A DOS attack on a network slice should have no impact on another network slice. However, a performance attack on a slice might affect the performance of another, and hence measurable information can be collected, learned, analysed and then used for the purpose of isolation breaking [58].

Costa and Costa [113] examined the VN space isolation mechanism implemented by FlowVisor by including a mischievous controller. They disrupted the network isolation and changed the traffic forwarding rules targeted by other network slices in the shared environment. The FlowVisor slices exposed several vulnerabilities as part of the isolation configuration by showing the functionality and compromising the isolation. The malicious controller acted as a transparent proxy between switches and controllers, and rewrote control messages according to the user defined policies.

One of the security risks in public cloud computing is relevant to multi-tenancy and resource sharing, and is the “guest hopping attack” which is caused by an isolation failure [30]. This attack is when an attacker attempts to identify two VMs hosted on the same physical server and uses access to one to gain access into the other.

Argyropoulos et al [2] stated several security issues that a cloud provider needs to consider; tenant domain violation, DDoS attacks, confidentiality of tenant data, isolation breakage and abuse, hacking tenant VMs in the cloud or the system VM components to find any vulnerabilities.

A jailbreak attack is a form of hacking the system service in order to overcome the isolation between tenants and control their applications, or leak or steal their data, screen or amend their information or configuration without doing it from inside the tenant domain. Such a security risk does not only harm the customer it compromises the whole cloud. Another concern is an issue called the “Lack of Resource Isolation”. If this occurs the attacker can control other client resources, establish a DoS attack, cause data leakage or data modification and steal cloud resources and assets unfairly. Another case of a “Lack of Resource Isolation” is to apply a side-channel attack in order to discover tenant co-existent assets in a shared environment despite being isolation-protected [29].

Real cases of attacks and security breaches incidence on cloud computing, include a side channel attack against co-existing VMs in the Amazon EC2 service[42] to steal confidential data. Another example is a DoS attack[36] on a physical infrastructure that caused a shutdown of all running VNs in the infrastructure [35].

Ristenpart and Tromer [42] noted a cross-VM side channel attack that had occurred in a shared area resource infrastructure. Most of the leaked information were key encryption or security certificates. They also discussed an attack where the attacker installed his mischievous VM in the same shared environment as the targeted tenants and launched attacks to gain knowledge about the co-existent VMs. A resource showed leaking information called (time-shared) caches to calculate the co-existing VMs processes to learn their locations or other relevant information. The authors also ran an experiment on EC2 VMs called keystroke timing attack, as a form of side channel attack to leak data.

Saeed and Garraghan [122] shared a similar aim to Ristenpart and Tromer's work [42] where both investigated co-residence VM security and vulnerabilities. However, they differ from each other in methodology and the testing process involved. Ristenpart et al applied cache side-channels, but Saeed and Garraghan used network channel attacks via mirroring and TAP interface impersonation. We shared some details with Saeed and Garraghan such as implementing the research on an OpenStack Cloud Network structure and investigated Tap interfaces as one of the CVN targeted data elements to be leaked.

2.10.2 Security Aspects

Isolation is to keep a partition of a network separated from another part. Hence, confidentiality is hiding information of one slice from another and integrity is the act of separating the network traffic from another network or the entire system [22].

Security is the ultimate concern in cloud computing despite its inordinate benefits. Due to the nature of multi-tenancy and resources sharing, lack or weakness of isolation is the most alarming threat in the cloud. Based on some public studies [105]it has been acknowledged that the greatest risk raised by cloud tenants is leaking sensitive information as well as being exposed by neighboring VMs. Data leakage is the unauthorized transmission of information, be it customer data, machine addresses or labels, to outwith the system or organization. Hence, tenants have been demanding physical isolation with resource detection to reduce the security risks.

Virtual platforms require a Security as Service (SecaaS) mechanism above the management operation, where management actions should be logged and audited, as the management level with administrator privileges has the power to provision resources to multi-tenants, manage infrastructure functionality and control resource separation which could cause data leakage with misconfiguration [106].

In 2013, Kloti et al [58] used the **STRIDE** vulnerability assessment guidance to uncover some issues in an OpenFlow/SDN network setup. They concluded with discovering two vulnerabilities: Denial of Service and Information Disclosure [58]. However, OpenFlow has since been updated and these security issues have been addressed in the new versions, yet there is concern about whether OpenFlow is secure against network information and configuration data leakage. There is no evidence or research performed on data leakage especially when it is embedded in a large scale distributed system such as a cloud system[58].*These concerns led to our research into data leakage in cloud systems.*

Chau and Wang [107] pointed out that most cloud research was on Virtual Network Embedding (VNE) but neglected security aspects. The authors discussed issues of security aware network embedding but did not themselves note the issue of VN information and configuration data leakage in cloud computing.

With regard to security research on OpenFlow, there are papers on exploiting the OpenFlow visibility and attacks of the end system [108][109]. Solutions are proposed using a dynamic virtual IP for the inner side to be translated to an actual IP for outer communication [58].

Clouds perceive basic security by using network isolation, while the resource provisioning is done via the Internet. **Security Groups** are a supporting mechanism for Network Isolation in OpenStack Neutron, where the admin and users have the ability to configure security policies for VN traffic categorizing and routing [41].

New network technologies such as SDNs have brought newer security and privacy challenges. Due to the vast difference between traditional networks and the dynamic, programmable elastic SDN, the old security aspects are more complex with more targets to protect such as components in the both the data plane and the controller (switches, controllers) and the connections between them. Moreover, the introduction of data centres and clouds having the responsibility of protecting shared environments with isolated tenant resources and services provided to them, raises even more concerns.

Isolation of a VN affects the security of the network positively, as DCPortalsNg proved in the experiment of a DoS attack which reduced lost traffic by 90% in the isolated VN compared to the result found in a non isolated network [17].

Isolation failure is one of the defined potential security risks in cloud services and applications. Such risks may lead to critical data and system acquisition, especially in an environment where multiple tenants share resources. Moreover, it is even more hazardous when similar businesses use the same service in a cloud, as isolation of all the resources, compute, storage and networks should not be broken [32].

With SDN controllers, isolation functions become a vital part of the network security due to the controller functions of assessment and communication rebuilding for the entire network. Some controllers showed weaknesses with OpenFlow applications, which leads to vulnerabilities in controlling the whole network [111].

A case of SDN data leakage was caused by a security isolation failure of the logical network and credentials, which ended in VN deterioration. Due to the structured nature of SDN and the data and control plane decoupling, the data modification issue is one of the VN security concerns, easily targeted if the isolation methodology showed weakness, as with FlowVisor [27].

Even with isolation that helps to contain any security threats in a multi-tenancy structure, there is the risk of compromising the hypervisor and breaking the isolation [102].

Modi and Patel [114] considered Cloud Virtual Network Security, *similar to our research*, except that they differed in the research aim. While they approached the cloud virtual network security challenge by introducing an intrusion detection system (IDS) solution, *our research aimed to investigate CVNI security for data leakage.* Modi and Patel claimed that it is the first research to tackle IDS detection of external and internal attacks while other researchers tend to focus on the internal. *We used a similar approach (external and internal attacks for security detection) to investigate our research question except that we used a penetration testing methodology while Modi and Patel used DOS/DDOS attack detection whereas we investigated data leakage.*

Lopez et al [115] proposed a security real-time threat detection via machine learning for an NFV challenge in an Open Source platform. Lopez et al worked on NFV with the use of an Open Source platform (OpenStack) as a testing environment. This is similar to the research presented here where we considered NFV as part of a cloud network and we also used OpenStack as one of two testbeds. Lopez et al also used Kali Linux distribution to launch their attacks.

There are several works in progress on Virtual Networking, cloud Networking, NFV and SDN security research *that intersect with our research but do not cover the same gaps*:

1. Multi-tenant isolation in a cloud environment using Software Defined Networking [116]:
   - Multi-tenancy and isolation between the tenants using vSwitch, should handle more control in terms of the isolation compared to current configurations. Control allows the vSwitch to save and utilize the tenants ID and handle the isolation from within the vSwitch itself.

2. Virtual Network Function (VNF) hardware trust in a network function virtualization (NFV) software defined network (SDN)[117]:
   - To sustain communication trust between the Source and Targeted NFV SDN, the source and targeted controller and source and targeted vSwitch is created via a Hardware trust verification.

3. Methods and apparatus for provisioning virtual network functions from a network service

provider[118]:

- This research is intended to expand the current facilitation of VNF provisioning provided services and functionalities flexibly beyond the provider service location and to activate the service in WAN and beyond. That is, providing NFV in any location with any deployment environment and communication paths.

2.10.3 Testing and Evaluation

textbfGEANT tested SDN-Enabled and Open Cloud Exchange scenarios to test multi-tenancy confirmation by implementing VN traffic isolating using FlowVisor, VeRTIGO or OpenVirteX [2].

A virtual infrastructure threat model can be created by an attack plan or a penetration test method. Threat models could be used to tackle all attacks, penetration attempts or leakages in virtualized environments or focus on specific attack scenarios for specific goal and target specific resources [31].

In order to prevent virtual system design risks, risk modeling is a logical step to evaluate stability, malfunction, misuse, or information leakage. OWASP is Microsoft's threat modeling site to evaluate web applications security and it works well in a virtualized environment. In virtual environments the lack of information security may result in resource controlling and privilege escalation. However, VM data leakage goes two ways; into or out of the VM. Side Channel attacks are specialized in leaking out of VMs[31].

2.10.4 Open Issues in Isolation

Carapinha et al [38] listed several "Technical gaps and open issues" related to VN and isolation and these include isolation promise after expansion, standard platforms to ensure isolation, interoperability, levels of isolation and isolation administration.

Cloud VN infrastructure building blocks is an on-going challenge especially when it comes to isolation. Where a VN is implemented in a server it was usually done via a hypervisor operating on the server. Instead of using a physical switch, VLAN was configured on a host server with VN technology such as an L2 kernel-level bridge/switch which was resident in between the VMs and the existing physical host interfaces. VSwitches were implemented on one or both of the Linux hypervisor deployments such as Linux bridges or OpenFlow dependencies such as OVS. Both are designated to provide traffic isolation and flexibility between several tenants co-existence in a cloud network infrastructure. L3 network capabilities and isolation used lightweight virtualization tools such as Linux Containers or namespaces where multiple numbers of them may be installed and configured each with their own network specifications (network stacks) [51].

2.10.5 Data Leakage

As stated before, Data (or Information) Leakage is the unauthorized transmission of information to outwith the system or organization. It is also essential to ensure that there is no *data leakage* occuring during the authentication and authorization processes, that is the initial stages of a client initiating a link to a cloud and the setup required. When clients need to validate themselves to get access to their resource, a large amount of data is exchanged between the client and the resource host. Based on the user's privilege levels they are authorized to access their resources. If the data exchange is not well protected, data leakage is more likely to happen [28].

Nimkar and Ghosh [119] stated that one of the security issues in IaaS VN is the routing protocol confidentiality between tenants, that is those protocols that send packets around a network. The authors investigated several protocols applied to virtual routers and switches specifically considering security issues such as origin authentication, path validation, hop integrity, address attestation and identity request server communication. They considered protocols such as MDR, ROFL, SVTR, S-BGP, IRV and So-BGP with virtual software routers on the control and data planes and hypervisor-based or hardware-based virtual switches. The paths drawn by the VN topology needs to be verified by the router with its routing protocol so these protocols must be equipped with some verification and authentication to prevent leakage or expose information during routing within the shared physical environment.

Another data leakage issue is one that Gutz et al [22] stated in their Splendid Isolation research. Their isolation approach, which they called separation, used virtual LANs (VLANs) to separate out the processing and maintain isolation to prevented data leakage between slices running parallel on the same shared environment. They coded different scenarios through programming to modify a network to satisfy client demands, despite demand diversity. When some programs ran side by side from each other the network interaction was different from running individual programs. The speculation was that if a function failed to perform, it might effect the network behaviour of the network and allow other programs to run and access resources that were denied to the broken program. Hence critical local information would be leaked, sniffed by mischievous clients and then used to investigate other tenants isolated resources. The authors noted that guaranteeing confidentiality of packet leakage and integrity was future work.

In Scott-Hayward et al's survey paper [27] data leakage is represented as a vulnerability in SDN,

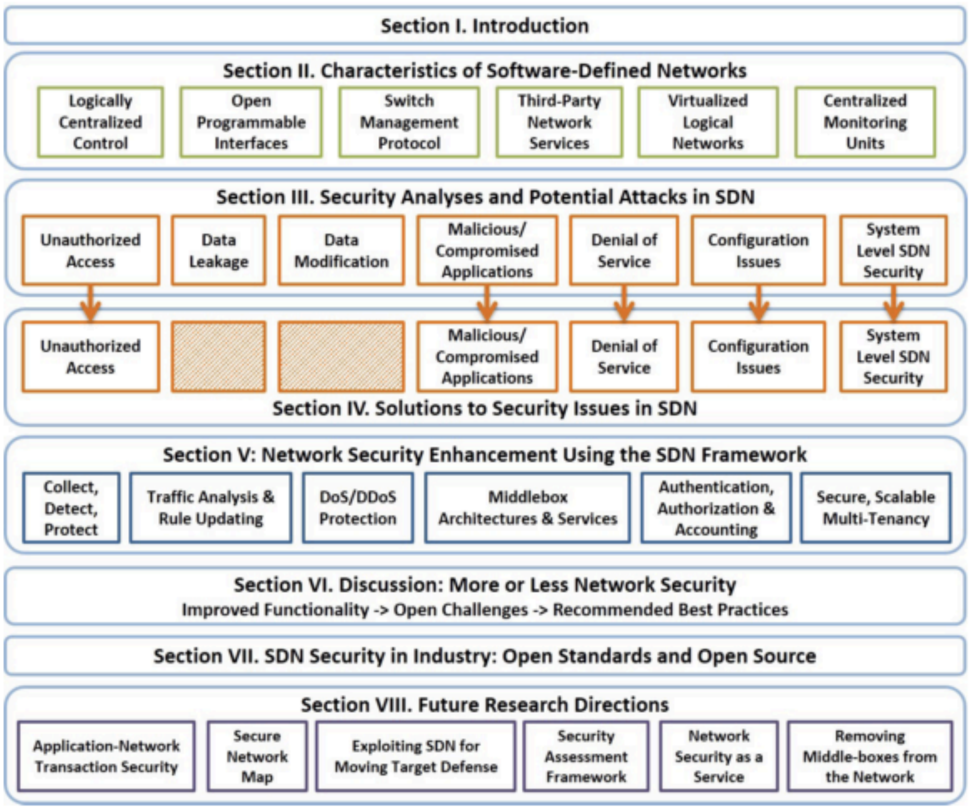

Section I. Introduction
Section II. Characteristics of Software-Defined Networks
Logically Centralized Control
Open Programmable Interfaces
Switch Management Protocol
Third-Party Network Services
Virtualized Logical Networks
Centralized Monitoring Units
Section III. Security Analyses and Potential Attacks in SDN
Unauthorized Access
Data Leakage
Data Modification
Malicious/ Compromised Applications
Denial of Service
Configuration Issues
System Level SDN Security
Unauthorized Access
Malicious/ Compromised Applications
Denial of Service
Configuration Issues
System Level SDN Security
Section IV. Solutions to Security Issues in SDN
Section V: Network Security Enhancement Using the SDN Framework
Collect, Detect, Protect
Traffic Analysis & Rule Updating
DoS/DDoS Protection
Middlebox Architectures & Services
Authentication, Authorization & Accounting
Secure, Scalable Multi-Tenancy
Section VI. Discussion: More or Less Network Security
Improved Functionality -> Open Challenges -> Recommended Best Practices
Section VII. SDN Security in Industry: Open Standards and Open Source
Section VIII. Future Research Directions
Application-Network Transaction Security
Secure Network Map
Exploiting SDN for Moving Target Defense
Security Assessment Framework
Network Security as a Service
Removing Middle-boxes from the Network

Figure 2.3: Overview of the SDN Security Survey[27].

see Figure 2.3 , section III [27]. The data targeted was not the user's personal and sensitive data, but was the OpenFlow switch traffic controlling rules to forward or drop or to communicate with the SDN controllers. Attackers could determine valuable knowledge through learning about the packet types and actions. Such an approach allows attackers to expose the proactive/reactive configuration of OpenFlow switches through analyzing the process time of traffic forwarded between input/output interfaces or directed to the controllers. Based on the knowledge gathered of the packets type from the leaked traffic, a potential DoS attack could be generated with forged traffic to cripple the SDN/OpenFlow functionality [27].

[27] also categorized security issues for SDN security issues. They listed three main categories affecting Data Leakage:

1. *Flow Rule Discovery* which can be exposed by a Side Channel Attack on an Input Buffer, shich is expected to target only the SDN Data layer.

2. *Credential Management*, such as Keys and Certificates required for every VN, which only affects the data layer.

3. *Forwarding Policy Discovery* via Packet Processing Timing Analysis, affects the Control layer, Control-Data interface and the Data layer .

Scott-Hayward et al [27] also noted seven SDN issue categories which are without a research plan or proposed solutions, as of 2015, and they are Data Leakage and Data Modifications. A best practice recommendation by the authors was to secure a shared network environment from various security issues such as unauthorized access, data manipulations, and data leakages prevention by isolating the network resources via slicing. Slicing provides a better resource isolation and allocation mechanism than does VN.

Different Data Leakage sources will have different impacts based on the data which has been leaked. For example, Data Leakage from the data plane will cause data to exist in a place where is does not belong and where there is no protection plan for it. Leaking data from the control panel and the management plane is more devastating as it will extend the impact onto the data plane for manipulating/modifying the forwarding rule or changing routing functionality. Hence, a network virtualized structure should be at same level as the hardware equipment in terms of *control and feature* to secure the related information from leakage [6].

Data leakage between virtual elements is not important for the following cases. When two virtual elements share the underlying structure Data Leakage can happen if one element gathered information about a co-existing element. Attackers may gain knowledge from compromised leaked data in order to cause more damage. Another case is when attackers find a vulnerability in the underlying structure that makes it easy to collect critical information about co-existing components. Mis-design and mis-implementation are other issues for a shared infrastructure. Shared resources require a proper isolation in order to avoid leakage between abstracted elements in components [6].

A misconfiguration in a cloud shared environment component such as mistakenly connecting two tenants to the same resource, or failing in isolation implementation, will lead to tenant critical information leakage. One of the impacts this may cause is a tenant domain being attacked by external tools or by another tenant taking advantage of that unnoticed weakness [80].

A Data Leakage attack is not categorized as a high security threat impact, but what makes it dangerous is the value of the information leaked to the attacker and the knowledge to launch a far more severe attack. VMs are the most vulnerable element in the cloud, and are likely to be, the biggest source of data leakage incidents [120].

Another case of data leakage is VM duplication in compute on-demand provisioning. Amazon EC2 provided a facility for creating a template of an image based on a created and fully functioning VM and made it public to be used by other tenants. The issue was the sensitive data content of the image when it was running as a VM. It foolishly shared the tenant contents to other tenants [121].

Rong et al [5] conducted a comprehensive survey on cloud security challenges and noted that cloud service tenants wanted answers to their doubts about their sensitive and personal data privacy. Concerns included :

- How and where their data is stored?
- Is the data reachable by anyone other than themselves?
- Is it misused by any entity even the system admin?

In order for the cloud tenants to trust the sharing data storage over an untrusted storage provider, an incremental encryption mechanism was proposed [5]. Hence cloud storage services could be trusted even if the tenant does not trust the service provider. This mechanism is one of the suggested solutions to data leakage prevention.

The Cloud Security Alliance also reported a hot list of cloud computing threats and this list included shared technology issues, i.e isolation mechanism limitations and data loss leakages. Moreover, the list did not specify which mitigation resolves which issue with the use of IP monitoring, better authentication mechanisms and stronger firewall, encryption and well programmed APIs [1].

Beggs et al [123] state that one of the overlooked components is IPv6 as much solution development is still based on IPv4. Developers apparently forget to include IPv6, mis-implement them or disable them in the system. Therefore, cases of data leakage via IPv6 are highly possible. The authors also state that metadata can be categorized as a source of data leakage, as some tenants are not interested in them but they are most likely publicly available and hold valuable information that attackers can gather and use to design their attacks.

DNS leakage is one of the network critical information that exposes systems when DNS information is requested from the ISPs. Even if the system uses applications such as TOR , The Onion Router, to hide, DNS leakage is still possible according to Beggs et al [123] they indicate using dns.leaktest.com to check for DNS leaks after a DNS request is made as well as using the command tools dnsdict6 and dnsrevenum6 tools.

Another solution for data leakage prevention is selective encryption. Based on the information content format, encryption rules are designed to encrypt the data according to their type, and hence even if the data is leaked, the content would not be reachable [32].

2.10.5.1 Security as a Service (SecaaS)

Multi-tenancy and data leakage are strongly related due to sharing resources. Even with a **Security as a Service** (SecaaS), data leakage is still a concern. The service needs to monitor all the processes and data in the system but clients might not appreciate their resources being monitored by a source that is not part of their control. Data could be anonymized before they are monitored by SecaaS [32].

A data leakage vulnerability in an IaaS cloud may be triggered from the hypervisors in use and their associated APIs. Weaknesses in them could lead to illegal access of cloud tenant resources, information stealing, resource misuse or using the leaked data to form further attacks according to Catteddu and Hogben [29][103]. The authors also discuss how to deal with information leakage , as data goes through several stages between process, transmission or storage. Some stages, process data for example, should not be deleted or encrypted [29].

Thimmaraju and Rétvári's paper [124] was presented in August 2018. They researched Virtual Network Isolation by investigating 22 vSwitches and evaluated the network isolation, enforced by vSwitches, for security. Their analysis revealed four security vulnerabilities namely: Co- location, single point of failure, privileged packet processing and manual packet parsing. They planned to implement three security designs to enhance the virtual network isolation via implementing a better robust vSwitch. *Our work intersects with their research in several ways, but our research aims are different. Our research involved one of 22 vSwitches and OVS, which according to the authors presented one of the four major security weaknesses. The research differs in other ways:*

1. *While their research is focussed on pure VN, we tested the Cloud Virtual Network,*

2. *Our aim was to determine the CVNI for Data Leakage while they focussed on vSwitch design for security.*

3. *They implemented their scenario with special hardware, SR-IOV, which we listed in our future plans.*

Our research, described in this thesis, analyses the cloud virtual network isolation to determine if tenant co-residence VN elements and configuration can be leaked.

2.11 Penetration Testing

Allen et al[125] define Penetration Testing as a process to conduct an in-depth security assessment or audit. A **methodology** defines a set of rules, practices, and procedures that are pursued and implemented during the course of any information security audit program. A **penetration testing methodology** defines a roadmap with practical ideas and proven practices that can be followed to assess the security posture of a network, application, system, or any combination thereof.

A common misconception is that hacking is about coding scripts to open any computer anywhere. Pentesters choose their method to discover and exploit target security vulnerabilities. **Ethical hackers**, such as Broad and Bindner, use a four phase structure penetration-testing framework (Figure2.4,[126]) in order to experimentally extract data from a target system and exploit weaknesses to report them [126].

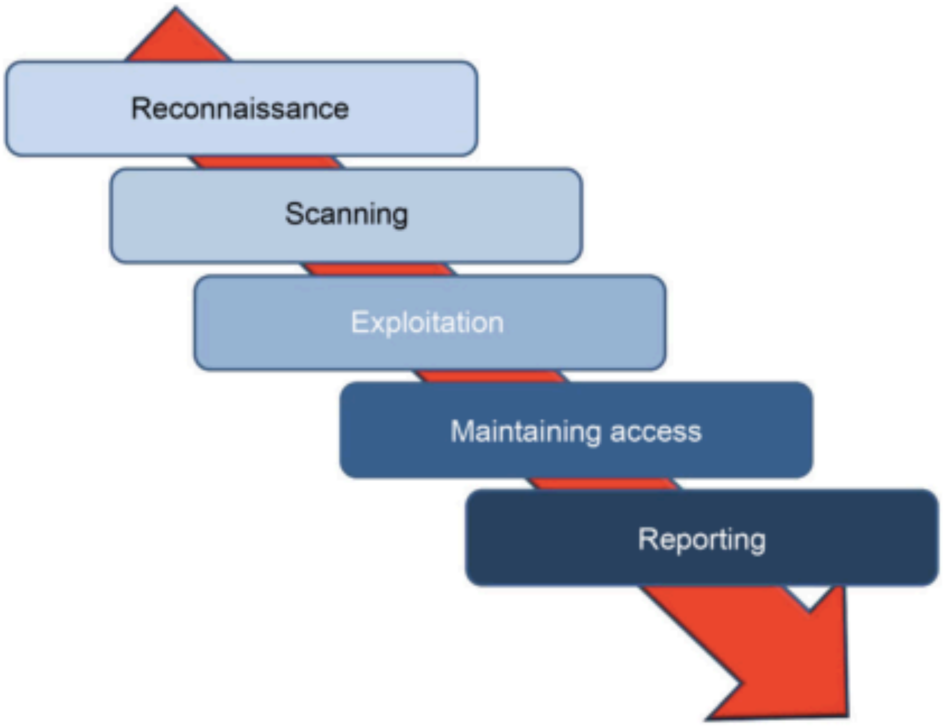


Figure 2.4: The penetration testing life-cycle[126].

The Penetration Testing Execution Standard (PTES [1]) is the current standard for performing penetration tests [127] and offers a standard and accepted model which consists of seven main phases:

1. Pre-engagement Interactions

[1]http://www.pentest-standard.org/index.php

2. Intelligence Gathering
3. Threat Modeling
4. Vulnerability Analysis
5. Exploitation
6. Post Exploitation
7. Reporting

The PTES and the Broad and Bindner Penetration Testing life-cycle as described above in [126] follow similar strategies. Initially a reconnaissance stage is performed which involves information gathering without a network connection to the target. This is followed by an analysis of the target by threat modelling, vulnerability analysis and passive scanning. Only after a lot of passive work is an active exploitation performed.

Information needed -**Information Gathering**- to be collected for the early phases of a penetration test includes [125]:

- Fundamental client information.
- Penetration test aims.
- Penetration test approach: Back box or white box testing, External or internal testing, social engineering, DoS attack, etc.
- Count of compute and network hardware to be included in the test, e.g servers, routers, switches, etc.
- Instance OS, Software and other technologies.
- Network and security service consideration required to be tested such as routing, switching, etc.
- Risk analysis.

Testing the target's code is often done under scanning or vulnerability analysis. The two main types of testing are white box and black box testing [125]:

- Black boxing testing is when an auditor/tester analyses a system without knowledge of the structure or software within. Vulnerabilities found are categorized and ranked according to their risk severity. The security auditor then produces a report consisting of a list of the most important security risks.

- White box testing means a tester has access and knowledge about the internal targeted infrastructure. White box is easier to examine and find vulnerabilities. As a result it is easier to pinpoint any internal security challenges. White box testing requires a clear access to the system infrastructure.

This research adopted the while box testing approach partially in the DeepDive methodology that is mentioned in details in Chapters 3 and 4. Moreover, Black box testing, internal and external testing are other approached used by this research to detect the Cloud Virtual Network Isolation Data Leakages, and these approaches are extensively detailed in Chapter 5 of this thesis.

Generally, penetration test cases are planned in order to accomplish a certain objective, such as vulnerabilities disclosure. A not very well defined security policy or an incorrectly configured service will eventually lead to data leakage and critical information exposure [25].

Penetration testing is often abbreviated to **pentest**, but alternative names are PT, Hacking, Ethical hacking, White hat hacking, Offensive security and Red teaming [128].

Penetration testing can also be defined as a legal and authorized attempt to locate and successfully exploit computer systems for the purpose of making those systems more secure [128].

A penetration test is a good technique to analyse whether the existing security platform is suspected of being inefficient. Penetration tests do not only expose the vulnerabilities, as vulnerability assessment does, it also extends the job to identify which vulnerability is "exploitable" [125].

2.11.1 Security Testing Methodologies

The types and option of selecting **Security Testing Methodologies** are numerous, as there are Open Source tools, to evaluate security technical features, administrative conditions etc. These methodologies are ideal for urgent, complex security assessments jobs irrespective of the size and density of the system. The objectives of these methodologies are to provide a clear guidance regardless of the type or the number of tests deployed [125].

Several distinguished penetration test methodologies exist such as Kevin Orrey's Penetration Testing Framework, Information Systems Security Assessment Framework (ISSAF), NIST SP 800-115, the Technical Guide to Information Security Testing and Assessment, Open Source Security Testing Methodology Manual (OSSTMM), Open Web Application Security Project

(OWASP), Penetration Testing Execution Standard (PTES) and the Offensive (Web) Testing Framework (OWTF) [123].

Choosing the right methodology depends on the pentester and on which one that will provide the best objective accomplishment, either to target technical specifications and infrastructure, or security risk suspicions, production aims, etc [125].

Pentesting processes do not always follow the structured methodology that have been mentioned. They do not justify the reason for performing penetration testing or to specify the targeted information. Pentesters sometimes consider that following the methodology limits their creativity and the methodology does not always simulate real attack behaviour. Therefore, a penetration test framework was needed to reflect the attacker viewpoint and simulate the real attack behaviour such as a *Kill chain* [123].

The Kill Chain penetration test approach - Figure2.5 was introduced by Mike Cloppert [137] and is unlike the linear penetration test life- cycle [126] due to its parallel stages. This approach represents a multi-attack performance with overlapping phases, which is more suitable for the research described here than the Broad and Bindner model [126]. The Kill chain consists of four main phases [123]:

1. Reconnaissance, which can be active or passive. Reconnaissance takes up to 70% of the total load of the pentester effort in order to gather information about the target.
2. Delivery, where the pentester selects their penetration tools to perform an exploitation.
3. Exploitation, where the target is actually compromised.
4. Post-exploitation which could be a specific action of the target, called Action on the objective, or when keepong the target - attacker connection open for future plans is called Persistence .

It is not enough to investigate the vulnerability, but to have proof an attack is a valid threat. The aim is to enhance network security against any anticipated attack via testing the system using the same attack tools an attacker would utilize.

2.11.2 Examples in Academic Research

NICE is a research project [112] focussed on cloud network security which proposed a solution using a penetration testing methodology. NICE introduced an intrusion detection engine based on the XEN virtualization cloud server. The project uses an attack graph to evaluate the attack risk level, and then select the most appropriate mitigation for the detected attack. The attack graph is created and rebuilt depending on scanned live or offline vulnerabilities, where the live

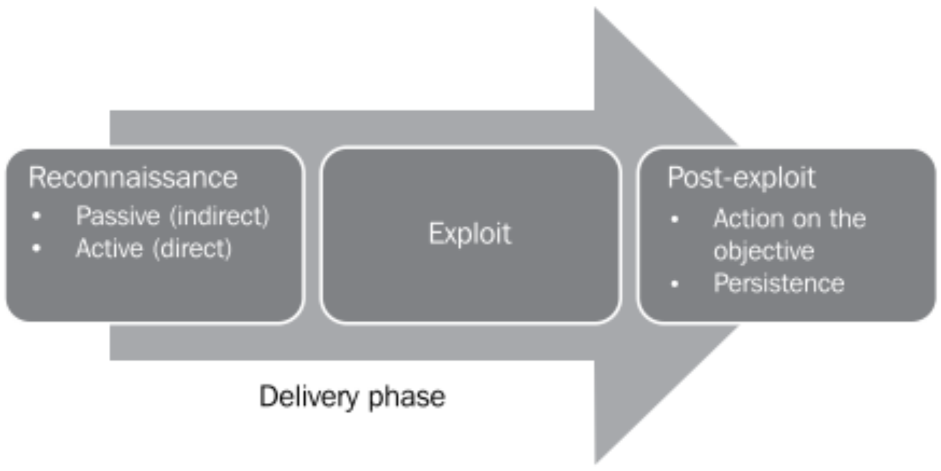


Figure 2.5: Kill Chain penetration test approach [137].

vulnerability has been scanned by the network controller. The offline scan is done using the penetration testing methodology databases such as OSVDB, CVE and NVD.

The penetration tests used in NICE are based on scripts involving the Metasploit framework and Armitage tools to emulate the attacker behaviour from different directions (externally and internally) and by firing multiple attacks taking advantage of existed VMs vulnerabilities [112].

As a security best practice, Mirantis [43] stated that before launching the cloud dashboard for public use, an external viewed IP must be scanned using vulnerability tools. They recommended using OWASP on the cloud dashboard as a penetration test methodology.

PCI DSS Virtualization Guidelines, recommended that any virtualized environment must be penetration tested, both externally and internally, on a yearly basis and whenever the infrastructure is updated or modified. Moreover, conventional system security skills do not reach the level required for virtualized environments, and hence a special training of penetration test is essential [6].

Penetration testing, Network scanning, vulnerability assessment and other security technique have proven to be more cost effective after delivering better results than any other expensive solution used by Testing the Security Architecture Companies [129].

Academic and commercial companies preferred significant objective threat modules for particular defined security evaluation scenario such as OWASP [31].

Labarge and McGuire [44] used network protocol and command line fuzzing, session hijacking and credential theft which are multiple penetration tests to test an OpenStack Essex Cloud Management Software. The outcomes of this research was a list of exploitable vulnerabilities in OpenStack (*which was of interest to our research*) and suggested several counter measures to strengthen the cloud from being attacked either by controlling the entire cloud platform or by attaining sensitive information. *This work indicated that OpenStack was a good choice as a test cloud for our data leakage experiments because of known vulnerabilities, even if we did not perform an exploitation phase.*

Discovery scanning is the process of identifying live hosts on a network. Tobergte and Curtis [130] listed the results of a discovery scamming, usually the target IP address, and several tools, which can be used to discover targets using protocols related to layers 2,3 and 4 of the OSI model.

2.11.3 Kali Linux

Penetration testing could have another stage of information gathering via tools such as those installed in **Kali Linux**. Example of useful information to collect is SDN, hostname, IP address, credentials, configuration, etc. Information collected can be classified into active or passive information. If the information is collected directly from the target via interacting with it through communication, then it is active data collection, If the collected information comes from other resources such as search engines, security Databases and repositories, then it is passive information collection. It is safer to collect information passively, but active information collection grants more detailed information gathering.

Kali Linux by Offensive Security, is a multipurpose operating system packed with the most significant security assessment and penetration testing tools. The previously mentioned methods commonly contain the following adjustable stages: Target scoping, Information gathering, Target discovery, Enumerating target, Vulnerability mapping, Social engineering, Target exploitation, Privilege escalation, Maintaining access and Documentation and reporting, all of which are in Kali. Kali Linux OS has an application menu that is organized based on the common penetration testing stages.

The main Kali tools used in this research are listed below along with some alternatives.

2.11.3.1 Kali Tools

Whois	used to attain information about the host from the domain name or an IP address.Whois can expose the routing path from pentester device to target and whether there is a firewall in place. Alternatives are traceroute, tcptrace.
Sparta	used to scan and enumerate a target, recusively processing from one tool to another embedded within it.
NMap	a classic tool used by pentesters either on the command line or through a GUI (ZenMap). It takes a single target, or full network range and scans at different intensity levels for ports, mail servers, load balancing, IP addresses etc. AMap can be used if only port information is needed to match against a registered database. Also, traceroute, tcptrace, ping, arping, fping, hping3, nping give information as to liveness of a target.
Nessus	is a vulnerability scanner which uses the classic CVE database and can link to other security tools. It uses a scripting language, the Nessus Attack Scripting Language (NASL) to test for individual threats and potential attacks. Nessus creates policies which are vulnerability tests applied to a target. Nessus can target different software versions and patches for updates, apply brute force testing and assess the target configurations for security threats. Vulnerabilities are labelled as critical, high etc. An alternative would be to use the CVE, OSVDB databases etc by checking operating system and software for vulnerabilities manually.

2.11.3.2 Pentesting Stages in Kali

After the **Information Gathering** phase - as mentioned in page 61-, the next step is to target devices in the network and determine their OSs and applications. At this point the best option is to identify the live targets and their OSs. These tools include ping, arping, fping, hping3, nping. All these tools depend on sending ICMP echoes to the targets; either a common ICMP echo like ping, or ARP requests such as arping, or multiple targets at one time such as fping and nping [125]. However, many of these tools are included in other bigger packages such as NMAP.

The best tool to identify live targets OS is NMAP with the –O option. The methods to find the target OS is called **Operating System (OS)** fingerprinting and is applied either passively or actively. By knowing the target OS, the tester (or attacker) can run more vulnerability tests based on the OS or select OS proprietary penetration test tools.

The next stage is **Enumerating Target**, and involves collecting much deeper information such as ports, services, etc.

Port scanning can be defined as a method used to determine the state of the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) ports on the target machines. An open port may mean that there is a network service listening on the port and the service is accessible, whereas a closed port means that there is no network service listening on that port.

Service enumeration is a method that is used to find the service version that is available on a particular port on the target system [125]. A Service version is a crucial piece of information for pentesters as it represents a valuable lead for vulnerabilities related to the service/software version. Example tools for detection are Network Scanners such as NMAP, ZenMAp and Amap.

A penetration test has to be concluded with a report to state the target status and list the security issues, the discovered vulnerabilities and suggestions as to how to mitigate them. The Kali Linux menu categorizes tools to match the penetration test life cycle phases and further penetration test tasks such as Information gathering, Vulnerability assessment, Web applications, Exploitation tools, Sniffing and spoofing, Reporting tools and System services, etc. Each category lists a number of popular penetration tools to serve the purpose of its category. Moreover, there are extra tool sections to provide specific penetration test tasks such as Wireless attacks, Reverse engineering, Stress testing, Hardware hacking and Forensics [125].

Consequently, we followed the Kali life-cycle which was similar to both Broad and Bindner [126] and the Kill Chain life-cycle [137], and we used tools embedded within Kali as used by real world pentesters.

2.12 Conclusions

From our research into the field of Cloud Virtual Networks, we decided to use penetration testing in testbed cloud network systems to test Isolation mechanisms. We wanted to discover any design flaws or mis-configurations that would compromise both the network isolation and multi-tenancy. A goal was to leak information about cloud network infrastructure and collect evidence about the tenant VN components and configurations. The desired output was a map of the infrastructure and a listing of other tenant resources to demonstrate security weaknesses in Cloud Virtual Network Isolation.

Consequently a methodology was chosen to allow a process of investigation via multiple tools to detect vulnerabilities in Data Leakage in cloud virtual networks and their setup. Given that Data Leakage was noted by several authors and can occur at different points in a network path, it was decided to determine what was visible or leaked internally (Internal Pentest) and externally (External Pentest) using the common tools a hacker or penetration tester would use within the

Kali package.

We used the tools and techniques outlined in this chapter to focus on information leakage from virtual networks, that is, we tested for Virtual Network Isolation. Isolation was described by multiple authors as a major research area and this work is one of the first to perform experimental analysis.

Chapter 3 outlines the first testbed built and tested, the Open Stack test cloud, to determine if our research into Data Leakage information gathering from penetration testing tools was valid.

CHAPTER THREE

OPENSTACK TESTBED AND INFRASTRUCTURE DEEP DIVE

Before building the cloud testbeds we revisit the objectives as laid out in §1.6 on page 11. The first research objective, discovering the research cloud platforms has been answered and Chapters 3 and 4 will cover the work on OpenStack and CloudStack. The next two research objectives were:

Q2: What are the required resources to evaluate CVNI?

Q3: what is the best methodology to identify the construction of the cloud network infrastructure, VN components and Isolation mechanism?

This chapter provides a detailed discussion of setting up an OpenStack - Advanced Network-Testbed to answer Q2. To answer Q3, this is followed by exposing the OpenStack Cloud Network Infrastructure through the Deep Dive Methodology. Finally, we listed the components of the Cloud tenants Virtual Network to be targeted via Penetration test in Chapter 5.

3.1 Overview

One of the reasons for the popularity of OpenStack is its simplicity to create, manage and facilitate the tenant requirements with Virtual Machines (VMs), routers, subnets, Firewalls, VPN, etc through a series of GUI options. A Software Defined Network (SDN) is fully supported by Neutron and its special feature is to add plugins from a large range of SDN vendors and technology. Hence, communication can pass through all components created by Neutron such as "routers, basic/advanced networks and security services"[138] [139].

The comparison in Sefraoui et al.[48] encouraged cloud provider administrators and researchers to deploy OpenStack especially if the intention was to use existing resources or to use the stack system for experimentation. Therefore we used OpenStack as our Testbed because of its community usage.

A cloud Tenant may represent “a customer, account, organization or project” based on the service intended to be provided [85]. Cloud tenant’s [1] “purchase” cloud resources such as compute, storage and network from cloud providers [18]. Tenants then utilize the rented cloud resources to launch VMs configured to provide services to their “clients”[17].

Despite the relative ease of creating VN components, it is important to first plan what tenants need to create and for what purpose. Neutron, the networking module responsible for network coordination, provides for a straightforward setup, build and management through the Horizon Dashboard, by supporting L3 virtual network abilities and logically defining the tenants virtual network [138]. Figures 3.1 and 3.2 show how the research Testbed scenarios for the virtual networks were planned for each tenant.

Despite the relative ease of creating VN components, it is recommended to first define what tenants need to create and for what purpose. Neutron provides straightforward setup, build, and management by OpenStack Dashboard (Horizon).

There are two tenants, Test with two VMs and Tenant1 with two users, each with two VMs. Figure 3.1 shows the Test Tenant scenario plan in the OpenStack Cloud Testbed. Test is a tenant, which has two VMs (my_first_instance and KaliLight) communicating through a network, Network1. The VMs in the network request and receive their IPs from the DHCP server and an outward facing gateway/ router (Router1) at 10.190.0.1.

Figure 3.2 represents the Tenant1 scenario plan in the OpenStack Testbed. Tenant1 consists of two users; User1 and User2. Each user created their own VNs (T1user1net for User1 and T1user2net for User2) and routers (T1u1R for User1 and T1u2R for User2). In addition, each user creates one VM (T1u1Vm for user1 and T1u2VM for User2) where the VMs are connected to the Internet through their network router.

The concept of the scenarios shown in Figures 3.1 and 3.2 are similar to the one described by

[1]Cloud Tenants, customers and users are alternative names used by documentation and published resources. Tenants are the most common used name by published resources, as we will use in this thesis. OpenStack documentation uses similar terminologies differently and such terminologies will be defined later in tenants and users section of this chapter.

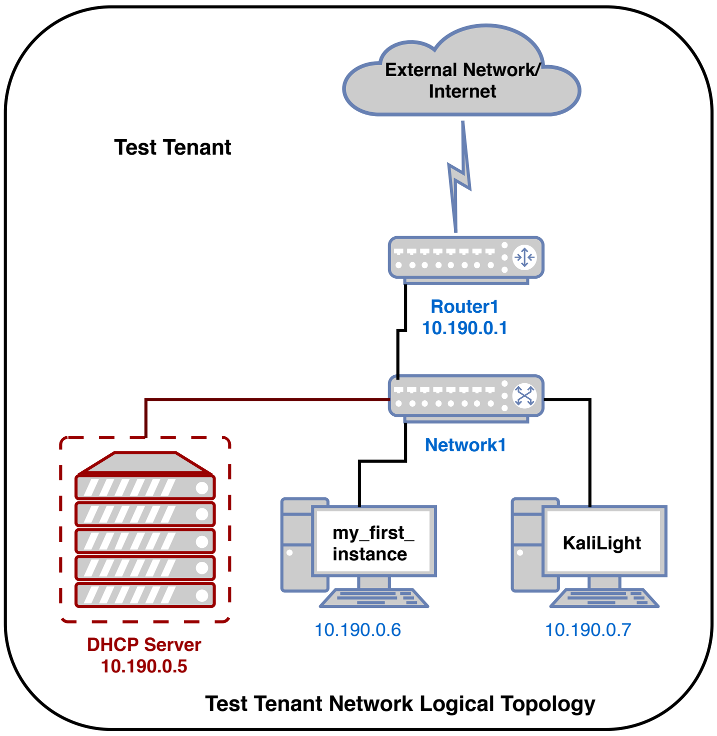


Figure 3.1: Infrastructure Plan of the Test Tenant.

Kashin [140] and are designed to:

- Create Virtual Networks (VN): use isolated networks for tenants and users and discover how these networks function within the cloud.
- Launch VMs: connect the VMs to the created network and observe how the communication operates within the cloud and the exit path.
- Request DHCP and Router services: router and DHCP services are controlled by the OpenStack Neutron. Our interest is to find out the role of these services and how they respond to tenant requests and communications.

Both scenarios represented in Figures 3.1 and 3.2 were implemented in the OpenStack cloud Testbed. The Testbed was implemented based on a multi-node architecture with OpenStack Networking (Neutron) using the OpenStack installation guide for Ubuntu 14.04 Mitaka[141] [142]. The Testbed shown in Figure 3.3 consists of two nodes; the Controller Node and the Compute Node. The internal infrastructure of the two Nodes is the focal point in this research as all the VN Isolation components and methods are implemented in both nodes. This will be explained in the following sections.

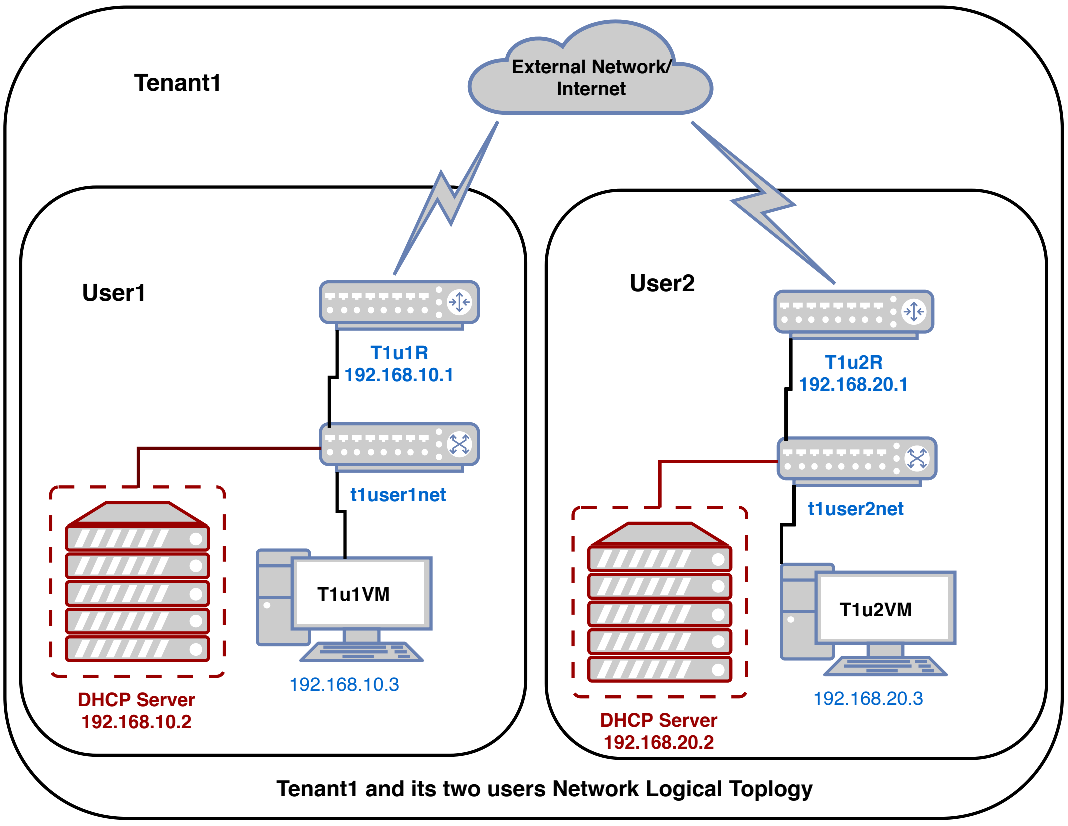


Figure 3.2: Infrastructure Plan of Tenant1 and Its Users.

Figure 3.3 represents the experimental Testbedused to implement the two scenarios. OpenStack was used as the cloud environment with Neutron to provide the VN implementation of Network as a Service. At this point there is no indication of how the cloud system will allocate and configure the provisioned resources of tenants and therefore how Figures 3.1 and 3.2 map into Figure 3.3.

3.2 Testbed Building

The following sections describe the process of how the Testbed was built and provide a discussion of the options and the configurations used to support this research.

3.2.1 Cloud Environment Preparation

First the cloud was built on two nodes, Controller and Compute, running Ubuntu OS. Following the online guidelines [141][142], the two nodes were connected to the management network (Tables 3.1 and 3.2), which manages the cloud communication between the nodes. The switch

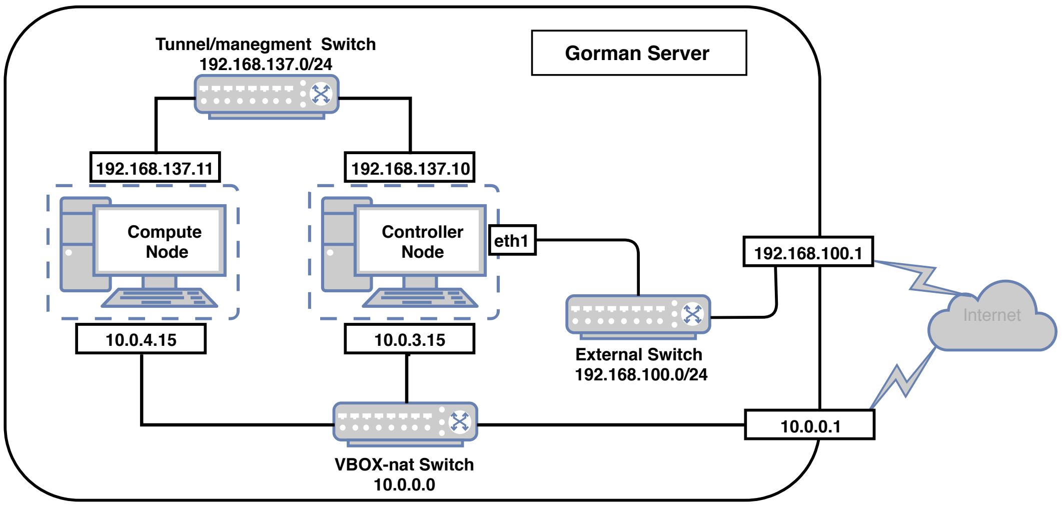


Figure 3.3: OpenStack Mitaka Cloud 2-Nodes.

ext-net was specified to provide instances (VMs) with an external connection to the Internet and the NAT network was specified for cloud packages installation and update.

Cloud Network:

Switch Name	Network	Gateway
Management	192.168.137.0/24	192.168.137.1
ext-net	192.168.100.0/24	192.168.100.1
NAT	10.0.0.0/24	10.0.0.2

Table 3.1: OpenStack Cloud Networking Configuration

Cloud Node:

Node	Controller	Compute
RAM	4GB	8GB
CPU	2 cores	6 cores
Hard Disk	300 GB	200 GB
Network	eth0:Static(Management)=192.168.137.10 eth1:Manual(Ext-net) eth2:Internet access(NAT)=10.0.4.15	eth0:Static(Management)=192.168.137.11 eth1:Internet access(NAT)=10.0.3.15
OS	Ubuntu-14.04.2-server-amd64.iso LTS	Ubuntu-14.04.2-server-amd64.iso LTS

Table 3.2: OpenStack Nodes Configuration Details.

Table 3.2 shows that the Compute node has a higher RAM and CPU as is required to support the hypervisor to create tenant instances. However, the Controller Node has an extra connection in eth1 that is set to manual as the Controller implements the Neutron service, which provides the external connections to the instances.

3.2.2 OpenStack Environment

In Appendix A, Table A.1 demonstrates the number of necessary packages to run the Mitaka version of CloudStack and shows the building process.

3.2.3 Cloud Network Configuration

In Appendix A, Table A.2 lists the most vital cloud configuration files that define the cloud network infrastructure. Due to the presence of Neutron as a network service in the controller node, the amount of files to be configured is significantly more than the Compute files. Moreover, Open vSwitch (OVS) plugins are extensively used as the configuration files indicate.

In the controller node, Neutron runs three main agents[143]:

- L3 agent: when a tenant creates a router, this agent employs the "Linux network namespace" to build the tenant vRouter using the configuration in the file l3_agent.ini.

- DHCP agent: another "Linux network namespace" is used to build up the DHCP server of the private tenant network based on the DHCP_agent.ini configuration file.

- L2 (OVS) agent: openvswitch_agent.ini provides all the configuration necessary to facilitate OVS within the cloud networking environment.

Namespaces and OVS are defined later in the next section. The explanations as to why and where they occur in our cloud is also presented.

All the noted options above, except for the networks and instances created by users other than the Admin, are controlled by the cloud provider and not the cloud users or tenants. In other words, all the configurations that define the VN structures are controlled by the configuration set by the cloud provider and the tenants are not aware of them.

3.2.4 External connection

Tables 3.1 and 3.2 list ext-net as an external network used by Neutron to provide the instance external connection. However, OpenStack designed the interconnection between the inner side of the cloud to the external side of the cloud through an OVS bridge called br-ex and it is configured as shown in the commands below. The command *add-br* adds the br-ex bridge to the cloud infrastructure. The command *add-port* links the controller node physical network eth1 -which is the exit path to the Internet – to the br-ex, which is the last edge of the cloud inner network connection.

The commands used were:

```
sudo ovs-vsctl add-br br-ex
sudo ovs-vsctl add-port br-ex eth1
```

3.2.5 Preparing the Testing Scenario

The following section outlines the steps taken to build the OpenStack testbed scenarios.

3.2.5.1 Tenants and Users

Cloud Consumers, users, roles are confusing terminology especially when it is referenced within the cloud systems. Openstack used similar terminologies for its implementation.

OpenStack uses the terminologies projects, roles and users. A project is described as an isolated resource container and is the new name of the Tenant where users are grouped within to share the resources and quotas. A user is the digital identity that can login to access the is OpenStack cloud services, is assigned to a specific project (tenant) and is then given specific privileges based on the role defined with (admin, standard, etc) [85] [142].

In this research we refer to the project as a Tenant and the user assigned to a project is a user. All users are given a standard role except for Admin, which had administrative privileges.

The OpenStack Mitaka cloud operates on two Ubuntu nodes (the Controller and Compute Nodes) as seen in Figure 3.3 earlier. Other than the Admin and the Test tenants which were created per the cloud installation guide [141][142], and which contained an Admin and Test user respectively, another tenant (Tenant1) was created as shown in Table 3.3. A private Virtual Network with a private router, see Table 3.4, was connected to a public router to allow instances to be connected from outside using SSH. Tenant 1 has two users (T1user1 & T1user2) each with a VM (instance), see Table A.3 in Appendix A. Tenant users provide credentials to access their instance and find more details about their assets in the cloud through the O enStack Horizon webpage, sometimes called the Horizon dashboard, (http://192.168.137.10/horizon).

Project	**Users**
Admin	Admin
Test	Test
Tenant1	T1user1 T1user2

Table 3.3: Projects (Tenants) Listed With Their Corresponding Users.

3.2.5.2 Network and Routers

The two networks– see Table 3.4- under the Admin are created by the provider, while the Test user did not create any. Users T1user1 and T1user2 create their own networks. Unlike the Admin

Network Details	**Admin**		**Test**	**T1user1**	**T1user2**
Network Name	ext-net	Network1	-	t1user1net	t1user2net
Router	external	Router1	-	t1u1R	t1u2R
Router Networks	-	Subnet1 ext-net	-	t1u1subnet1 ext-net	t1u2subnet ext-net
Physical network	br-ex	-	-	-	-
Network type	flat	vxlan	-	-	-
shared	Yes	Yes	-	-	-
Subnet Name	ext-subnet	Subnet1	-	t1u1subnet1	t1u2subnet
subnet IP	192.168.100.0/24	10.190.0.0/24	-	192.168.10.0/24	192.168.20.0/24
Subnet range	start=192.168.100.10 , end=192.168.100.200	start=10.190.0.5, end=10.190.0.254	-	-	-
DHCP	Disable	Enabled	-	Enabled	Enabled
Gateway	192.168.100.1	10.190.0.1	-	192.168.10.1	192.168.20.1

Table 3.4: Tenant Network Components and Configurations.

user, the two users did not provide the details of network configuration. However, they are assigned default values by the system based on the original configurations from the cloud setup files, as defined earlier in Table A.2. For example, the networks created by tenants are not shared by default and the network type is vxlan by default as configured in the file ml2_conf.ini above.

3.2.5.3 Instances

Table A.3 in Appendix A shows instances and configurations. My_first_instance is the only instance created by command line to test the cloud functionality. However, all the remaining instances were created from the dashboard using the Admin account. It is worth mentioning that each tenant can view only the information about elements they have created except for the Admin, which can view all elements. However, it was discovered that T1user1 and T1user2 could view all the elements from both users, which is of concern. Allowing tenants to create users if there is no privacy between them. In addition, all users can use Network1 not because it is created by the Admin but because the Admin set it to be shared. Therefore we decided that a test tenant should use Network1 to connect to its instances, and planned to make the Test tenant the cloud attacker.

All the above information and terminologies are essential and play significant roles in the VN created by the clouds. Relevant concepts and terminologies are defined in the following sections.

3.3 Technologies and Terminologies

This section contains vital knowledge to build up the understanding of the up-coming Cloud DeepDive Methodology section. The OpenStack Cloud technical components depend heavily on the terminologies listed in this section.

3.3.1 OpenStack Networking

Neutron runs in the Control node and is the sole module in the OpenStack infrastructure that is accountable for the management and coordination of VNs within the cloud environment. The main responsibility of Neutron is to manage subnets, routers, DHCP, etc, to ensure ultimate connectivity on the instances both locally and externally [140].

Starting from the Havana release, **GRE** and **VXLAN** are the overlay network technologies supported. The author of [143] pointed out that VXLAN is graded higher due to its higher performance capabilities. With respect to Table A.2 (Appendix A), our Testbed is configured with VXLAN as the OpenStack guides[141][142] recommended.

3.3.2 Virtual Network and Isolation Mechanisms

OpenStack provides two types of network setup; the Nova-network and the Neutron network. The Nova-network is not as flexible as the Neutron network, as the latter supports open source or vendor plugins to build up, govern and define the SDN hardware, or plugins such as built-in Linux facilities such as **OVS** and **Linux bridges**[139].

Note that VLAN, VXLAN and GRE tunnels are the isolation mechanisms OpenStack uses to isolate several networks created in the cloud. The isolation mechanisms are configured in the cloud configuration files listed in Table A.2, where our setup uses VXLAN as the isolation mechanism.

Neutron assigns the isolation mechanism configured, a VLAN tag or code, VXALN called vni, selected from the configuration, and allocated to a recently created network. Different tags are assigned for every network created and when traffic of any network is initialized the tag will be attached and all traffic from different networks with different tags will share one physical connection without interfering with other traffic from other networks. Therefore, Neutron has the ability to create multiple isolated networks, manage and track their tagging mechanism without burdening users with low level operations to ensure their privacy.

In addition, Neutron configured the namespace to be aware of the network tagging as each namespace will be associated with a created network, hence, the namespace must ensure the correct tag is applied to the network traffic passing through it[144].

3.3.3 Instance

An instance is created when a **VM** is created, connected to a network and launched in the Compute node which manages all the instances through the KVM or Quick Emulator (QEMU) hypervisor. When an instance is launched, it forms a connected sequence of cloud components, such as network, tagging, etc. Moreover, the tag is changed as the VM traffic is transferred within the cloud network components.

VMs can be created either through command line, as we did in the first_instance, or by using the OpenStack dashboard webpage, the GUI “Horizon”, which we used for T1u1VM. To list the VMs in the cloud the command *"# nova list"* is used[144].

Instances send their traffic thorough their virtual network card (vNIC) on the Compute node. On the inner side of the instance the vNIC is referred to as eth0 and its configuration details can be found in a file within the Compute node (/etc/libvirt/qemu/instance-xxxxxxxx.xml). On the cloud side, the XML file refers to the vNIC, eth0, as a **Test Access Point (TAP).**

The TAP device ID was named from the first 11 characters of the port ID, created by Neutron. Finally the TAP device of an instance is connected to an OVS bridge called br-int, which passes the instance traffic to the external connection of another cloud node[139].

3.3.4 Linux Bridges

The packets exiting eth0 from the instance are sent through the TAP device connected to the VM, as explained in the § 3.3.3. This TAP device is a part of a Linux bridge with a name starting with **qbr** and the port id as defined in the tap device attached to it. Although the logical idea is to connect the instance to the br-in (**OVS**) bridge directly without the qbr Linux bridge, however, such a solution is not applicable with to how OpenStack implements a cloud. An instance has to be defined within a security group for launching, and this involves the implementation of iptable rules on the TAP devices. However, iptables are not compatible with Open vSwitch. Therefore a direct connection between the instance and the OVS switch is not feasible [145][140][144][146].

As a solution, a Linux bridge is used to connect the VMs to the OVS bridge, through a set of ports called **Virtual Ethernet (Veth) pairs**. Veth pairs are a representation of a virtual wire and used heavily by OpenStack network structures. The Veth pair consists of two linked ports in the Linux bridge (starting with a **qvb** identifier) and the other port is attached to the br-int (OVS) bridge with the name starting with **qvo**[145][146].

3.3.5 Open vSwitch (OVS)

OpenSwitch is an open sourced apache 2.0 licensed technology, which introduced the multilayer vSwitch[139].

The OpenStack User Survey of 2016 categorized OVS as the most preferable network driver implemented within OpenStack Cloud provider[139].

OVS is powered with two protocols[140]:

- OVSDB: to configure and manage bridges, ports, VLAN, etc.
- OpenFlow: to register and manage the flow rules of traffic exchanges. OpenFlow and flow rules details are expanded in §3.3.5.4.

The architecture uses OVS to provide a connection between the Cloud VMs and the physical port of the cloud nodes[145].

For example, the controller node connects the eth1 physical port to the OVS bridge br-ex to allow an instance to have an external connection, as explained in the previous section.

The command *"# ovs-vsctl show"* lists the OVS bridges and the ports linked to it [145]. OpenStack with the basic guideline documentation installation [141] uses two OVS bridges in the Compute node; br-int and br-tun. Meanwhile, the Controller node consists of three OVS bridge; br-int, br-ex and br-tun as explained in § 3.4.5.4 & 3.4.6.2.

Any two OVS bridges are connected with a pair of Veth interfaces as virtual wire links between the two bridges. Moreover, whenever a VM creates a set of connected ports both the Linux Bridge and the br-int bridge share the same code, an 11-character port ID created by Neutron. The Open vSwitch agent handles the flow entries in the OVS bridges without any interference from the users [145].

3.3.5.1 Integration Bridge (br-int)

The Integrated bridge (**br-int**) is an Open vSwitch that has two tasks. First, it provides connectivity to all the VMs and network service instances, such as routers and DHCP in the cloud. The second task is to isolate the tenant's subnets using VLANs, where VLAN IDs are used by the br-int and are not circulated outside the cloud host, as they are only known locally [140][143].

The br-int bridge uses different techniques for connections, for example in the Controller they connect to the router and DHCP using tap and qr interfaces while they use qvo to connect to the Linux bridge in the Compute node. Meanwhile a patch interface is used to connect to other OVS bridges, such as br-tun and br-ex in both cloud nodes[146].

3.3.5.2 Tunnel Bridge (br-tun)

The Tunnelling bridge (**br-tun**) is another OVS bridge that exists in both cloud nodes. As in br-int, br-tun has a patch interface to connect to the br-int bridge; however, the main interface that defines the purpose of having br-tun is the tunneling interface (GRE or VXLAN) [143].

The duty of br-tun is to change the VLAN tagged traffic originating from br-int, to untag it and to replace it with a GRE/VXLAN tunneling ID before it puts the traffic into the physical network. The process is reversed if the traffic received from the physical links to be sent to br-int. The br-int link depends on the OpenFlow rule configuration in the Open vSwich translating between VLAN ID and the tunneling ID[140][143][146].

3.3.5.3 External Bridge (br-ex)

The Openv Switch External Bridge passes the VM traffic to the external network through the physical interface attached to it[140]. Any instance traffic meant to be forwarded externally goes through **External Bridge (br-ex)** using the namespace router interface (**qg-**) and is sent out through the node physical interface, Controller eth2 in our Testbed. Naturally, the physical interface attached to br-ex belongs to the same public network that the cloud external traffic is directed to[140][146].

3.3.5.4 OpenFlow Rule

OpenFlow rules are called forwarding entries. These entries are stored and organized within tables, where the packets always check table 0. Each table stores a list of entries numbered in descending priority. Entries with a higher priority are to be executed first. Other than the priority number, entries hold extra information such as port receiving packet, port sending packets, VLAN ID, tunneling ID, MAC address and action. Entries are executed through its action, which is categorized as[143][140][146]:

- Resubmit (X): packet to be transferred to table X.
- Output Y: use port Y to send exiting packet.
- Normal: normal packet flow without using OpenFlow configuration.
- Learn (table = z): a new entry to be added to table Z.

3.3.6 Namespaces

Network **namespaces** is a Linux kernel component characterized as an "isolated container". OpenStack depends comprehensively on namespaces to run its network services such as routers and DHCPs and manage them using its locally configured agents, such as the L3 and DHCP agent. Due to their isolation feature, namespaces encapsulate the entire network configuration, keeping functionality from being visible to the outside world. They allow network IP ranges to overlap. OpenStack uses namespaces to reduce its network complexity, and to isolate tenant networks[139][145].

The command *"# ip netns list"* is used to list all the namespaces presented in the cloud node. To access the network tenant's network configuration the command to be used must start with *"ip netns exec"* followed by the namespace name and any Linux network command, such as: *ip a, ifconfig, ip route, ping*, etc [146][145][140].

If the *IP netns* command is executed in a correctly functioning OpenStack Neutron cloud, the output will list two types of namespaces and they are [139],[145][146]:

- Router: is an L3 agent name such as **qrouter<router_id>**, where the router id can be found with the command *neutron router-list*
- DHCP service: the DHCP agent is named as **qdhcp<network_id>**, and the command *neutron net-list* will list all the tenants' networks including their IDs.

OpenStack integrates both namespaces and OVS bridges to form the cloud network infrastructure [145].

Interfaces created by OVS bridges are linked into the namespaces to accomplish overall connectivity without losing the isolation between the tenants. For example, a Linux network namespace router receives the traffic from the tenant private network coming from br-int and transfers it to the external connection through the external network gateway in br-ex, to be forwarded outside the cloud[140].

There are various ways to connect the namespace to the outside world. This work focuses on how this is achieved in OpenStack. OpenStack uses a combination of Open vSwitch and network namespaces. OVS defines the interfaces, which later can be added to designated namespaces [144].

3.3.6.1 Routers

A router is a Linux network namespace that gets implemented once an interface gets linked to it. This interface is defined as a router gateway[143].

The command *ip netns exec qrouter<router-id>* with the Linux network command displays inner network configuration and functionality, such as routing tables, iptables, interfaces and their IP address. A router namespace is connected to a **qg-port-ID** in the br-ex as its external network gateway and connected to Integration Bridge (br-int) bridge with the port **qr-port-id** to represent a tenant 's private network gateway[139][146].

3.3.6.2 DHCP

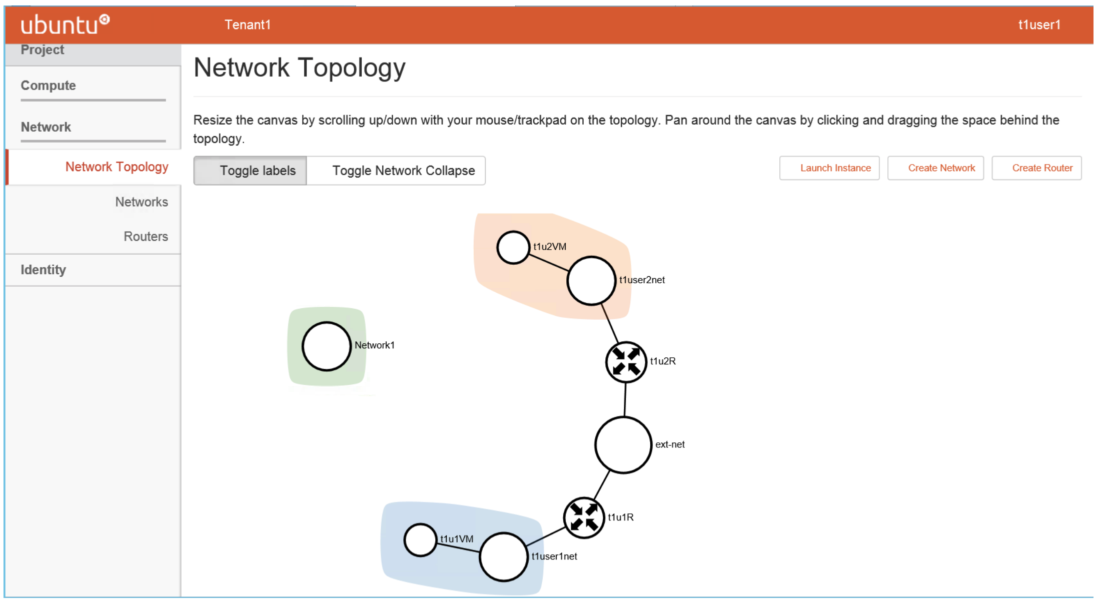
The moment the tenant's private network is created with a DHCP-enabled configuration, OpenStack establishes a namespace which runs the **dnsmasq** as a DHCP server. Similarly to the router namespace, a DHCP namespace can be listed using ip netns, while its contents are displayed using the *ip netns exec* command[146].

Dnsmasq is a lightweight Linux tool that provides a DNS and DHCP service for the network. It responds to the VMs DHCP request within the tenant private network [144].

3.4 The OpenStack Cloud Network Infrastructure DeepDive

Once the OpenStack testbed cloud was built we were challenged as to how to determine the mechanisms used to implement VN and associated isolation configurations. This was our Research Objective three; to find the best methodology to identify the cloud network infrastructure components. The DeepDive methodology was used to overcome these technical concerns.

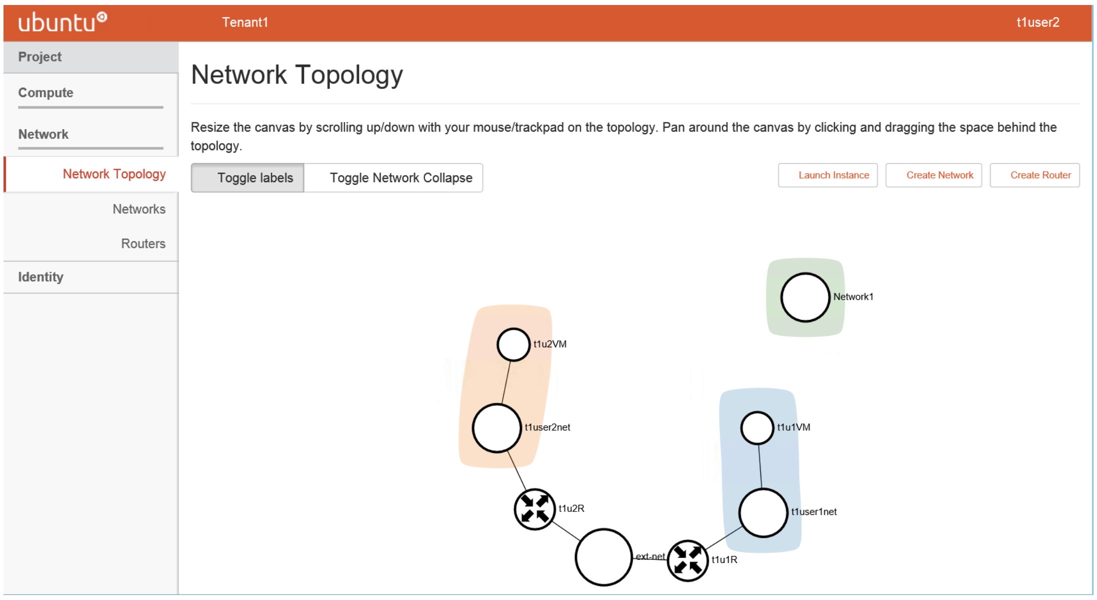
Before applying the DeepDive Methodology,Tenants were only able to view the network topology based on their privileges; the two users of Tenant1 can view each other's topologies and VMs, despite the fact that each user's VM is in a different network (Figure 3.4 and 3.5). The Test tenant can only view their own topology (Figure 3.6) and can see no sign of any elements from the Tenant1 network topology except for Network1 and external Network (ext-net). These are created by the Admin, and configured with shared features, which means all tenants can see and use that network.



Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.

Figure 3.4: Dashboard View of Tenant1 User 1 Network Topology

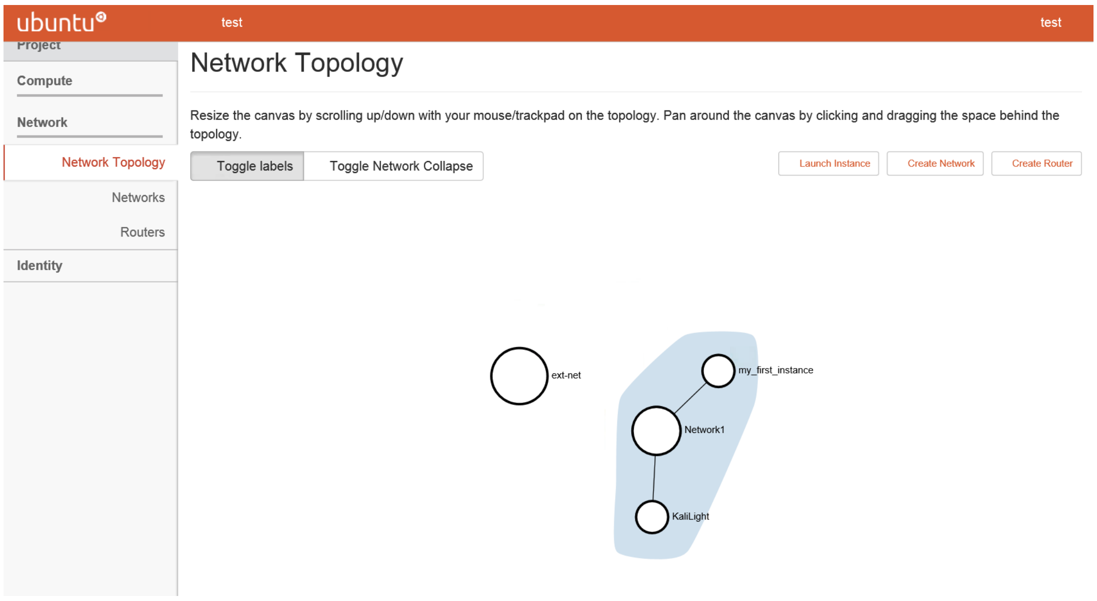
However, such information was not sufficiant to achieve the aim of the DeepDive methodology to collect as much information as possible from the cloud testbeds using different resources such as the testbeds setup, the scripting files (for white box testing) and the cloud web interface (dashboards). The collected information then had to be analyzed in order to represent the cloud network infrastructure as diagrams for each testbed.



Network Topology

Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.

Figure 3.5: Dashboard View of Tenant1 User 2 Network Topology



Network Topology

Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.

Figure 3.6: Dashboard View of Test Network Topology

3.4.1 Sources to Collect Information

As the Dashboard view of the tenants does not show any information related to other tenants and there is no indication of the network isolation mechanisms used, a deeper analysis was required to expose the true cloud network infrastructure and draw out the relationship between the tenant network elements and how they are implemented at the level of the cloud network. These goals can be satisfied using the deep diving methodology, and we used three sources of information collection:

- Information used from the cloud setup and Testbed building.
- Projects/Tenants accounts from the cloud dashboard. We used the Admin account, as all the information of all tenants is visible in it.
- White box testing in a form of scripts files executed in each cloud node with Admin privilege.

To clarify, in this chapter we do not aim to find the VN Isolation (VNI) vulnerabilities. However, we try to define all the elements and technologies used to build up the VN and its Isolation mechanism. Hence, we require to define the VNI targets that we will apply our hypotheses on.

Our OpenStack cloud network deep diving was executed using six script files. Compute.sh is the only script executed in the Compute node, the remaining five scripts are executed in the Controller and are:

- Controller.sh
- Compute.sh
- Admin.sh
- Test.sh
- T1user1.sh
- T1user2.sh

As we are the creators of the cloud we possess the credentials of the Admin user.

Compute and controller scripts focus more on collecting the physical components, configurations and connections of the cloud. The remaining scripts were designed to collect the information of the cloud components from the tenants or projects perspectives. Therefore, we started the scripts with an openrc file of each tenant/project with their credentials. The scripts used to collect the cloud component information are similar to the information collected from the Testbed and the Dashboard with the Admin account, but there are more details such as OVS and namespaces.

3.4.2 The DeepDive Diagram of the Cloud

A DeepDive is an analysus of a system to determine the identity and configuration of components such as routers, ports, tunnels and nodes. The DeepDive diagram is a visual representation of the OpenStack cloud inner infrastructure. The process of producing the diagram held multiple challenges. First, it required a long process of large data analysis. Starting from the information used in building the Testbed, we started building the outer layer of the diagram as the nodes (Controller and Compute) along with their interfaces and the switches connected to them: management, External and (VirtualBox (VBOX)) NAT.

Figure 3.3 on page 73 represented the outer infrastructure layer of OpenStack Testbed as built. Revisiting Figure 3.3, we can now show that the Controller node connects to a VBox NAT switch using eth2 on the network 10.0.0.0/24. The Management Network with IP address 192.168.137.0/24 connects hosts to the Management switch where a controller node connects with eth0. The Tenant external network with IP address 192.168.100.0/24 uses the manually configured eth1. Remaining ports such as br-ex represent the OVS bridges interfaces, which will be shown in the OVS section under the Controller node to follow.

Our research requires deeper analysis than what Figure 3.3. represents. Hence, after using the DeepDive methodology , the data must be reviewed in order to reveal the components and configuration of the cloud Network infrastructure. The outcome led to a contribution as we developed the DeepDive methodology and we thus transformed the Testbed representation from an outer layer representation (Figure 3.3) into the OpenStack Cloud Network infrastructure DeepDive representation as shown in Figure 3.7 below.

As stated above, the Admin account was used to gather information from the cloud Dashboard as Admin provides the system information about all the tenants. The reason behind running commands that give the same information as the Dashboard was to ensure that we are not missing any valuable information. We discovered that the commands in the scripts are more powerful than the Dashboard.

Focusing on the project scripts data output files (Admin, Test, T1user1, T1user2) the first step was to discover the networks created in the cloud. The Admin output file was the best option as it had the required privileges to list all the information of all the projects in the cloud.

Moreover, we wish to refer to the OpenStack project (tenant) data displayed using their ID to build the knowledge diagram that the DeepDive produces. The easiest way to match data is by running the commands under the Admin privilege-: *OpenStack project list* and *OpenStack user list*. We will utilize this information to define the ownership and visibility of the cloud network infrastructure components, as shown in Table 3.5.

Using the command *neutron net-list* shows the network ID, Name, associated Subnet Id and Network IP value. For more details, we used the Network ID and used it in another command,

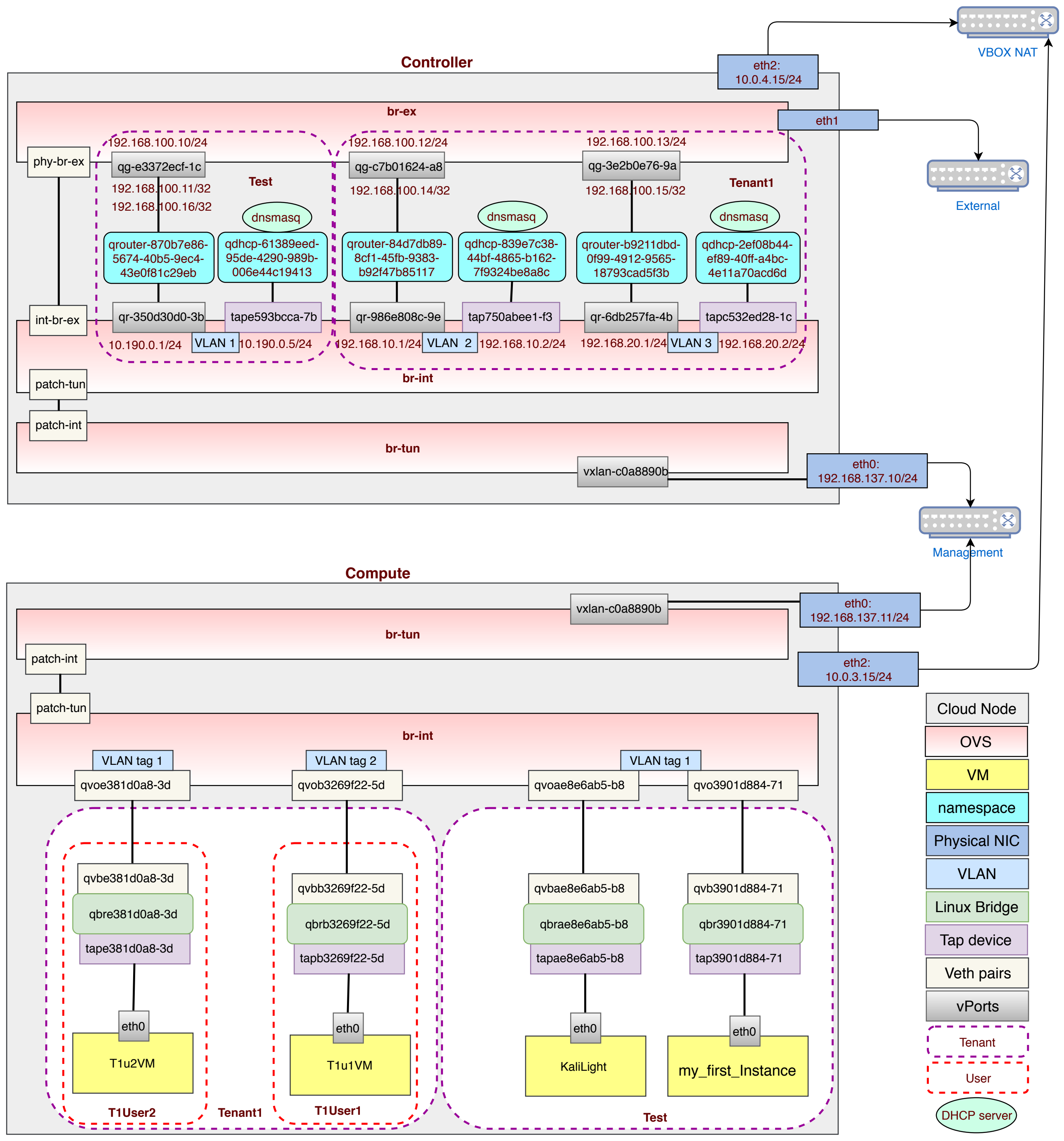


Figure 3.7: OpenStack Cloud Network Infrastructure DeepDive Diagram.

neutron net-show $net_id. Moreover, the *neutron subnet-list* command is used to list the subnet info, which displays the subnet ID, name, Network IP with CIDR and the IP address ranges. To obtain further details the *neutron subnet-show $subnet_id* was used. These commands were used in the Admin file and the other projects scripts files to show what information is visible for each

S.Q	**Type**	**ID**	**Name**
1	Project	0689c18aa8324c83add7472625d74e5c	tenant1
2	Project	59bde68f1a7a40829c8b67e098c295bf	test
3	Project	cc6f4dd57bfd4aba996934dfc51d3bc1	admin
4	User	0e291d5fff7746c1afa4c4369e6ad8c8	t1user1
5	User	1c750f9f80ec41daad80b0dae98707a6	admin
6	User	bfac39ffd3b343f393193240d432a24e	t1user2
7	User	d548a1c68d5a43368644128a468a029a	test

Table 3.5: List of the Projects Used in the OpenStack Testbed Cloud.

project (tenant). The command: *neutron net-list* showed four networks as is shown in Table 3.6.

Network Name	**ext-net**	**t1user2net**	**t1user1net**	**Network1**
ID	cbce2f55-dace-4b5a-a2ed-b2eabb2b45b9	2ef08b44-ef89-40ff-a4bc-4e11a70acd6d	839e7c38-44bf-4865-b162-7f9324be8a8c	61389eed-95de-4290-989b-006e44c19413
Type	flat	vxlan	vxlan	vxlan
Shared	True	False	False	True
router:external	True	False	False	False
Visible by	Admin, t1user1, t1user2, Test	Admin, t1user1, t1user2	Admin, t1user1, t1user2	Admin, t1user1, t1user2, Test
Owned by	cc6f4dd57bfd4aba9969 34dfc51d3bc1(Admin)	0689c18aa8324c83add7 472625d74e5c(Tenant1)	0689c18aa8324c83add7 472625d74e5c(Tenant1)	cc6f4dd57bfd4aba9969 34dfc51d3bc1(Admin)
Subnets				
Subnet Name	ext-subnet	t1u2subnet	t1u1subnet1	Subnet1
Subnet ID	05015b1b-ae84-4a79-be75-126bf2fc21c7	8e594217-2561-4165-b9cb-40de1a981edc	27554221-332b-4488-ac03-1e380d101a3d	e53b0a6f-325b-42ee-9339-e811e78cbe5b
Network IP/CIDR	192.168.100.0/24	192.168.20.0/24	192.168.10.0/24	10.190.0.0/24
IP Range	"192.168.100.10", - "192.168.100.200"	"192.168.20.2", - "192.168.20.254"	"192.168.10.2"- "192.168.10.254"	"10.190.0.5", - "10.190.0.254"
DHCP	False	True	True	True
gateway_ip	192.168.100.1	192.168.20.1	192.168.10.1	10.190.0.1

Table 3.6: Network Details.

Table 3.6 lists the network details and subnet details derived from these commands. The Table consists of all the networks in the cloud as designed in the Testbed where all the information is equally provided by the scripts and Dashboard. However, the information from the Testbed was limited compared to the other resources, which provided extra information as detailed in Table 3.6. Moreover, the privileges associated with tenants or projects, specify what information is visible. For example in the provider network only Admin can view the network type, physical network and Segment ID.

Despite the fact that Table 3.6 contains important information about the network, at this point in the Deep Dive, there was no indication on how this information mapped in Figure 3.7, and how each piece of information was located within the infrastructure above. More investigation and information gathering was required to complete the DeepDive diagram, shown in Figure 3.7. These steps are described in the next section.

3.4.3 Cloud Ports and vRouter Commands Used

Each IP from a subnet is related to a port, which contributes to building up the tenant networks and is linked to the underlying network, such as routers, DHCP servers, and VM interfaces. Several commands were used to collect specific information as shown in Table 3.7:

The commands in Table 3.7 produced a large amount of information and it would be extremely confusing for the reader to display everything here. However, we will explain the findings from the data collected and point out the important information that has been discovered in the DeepDive diagram above.

3.4.4 Initial Important Findings

The Admin tenant has higher privileges and therefore all the information about the networks is fully visible. T1user1, T1user2 can see the entire network because t1user2net & t1user1net are created by the same tenant that the two users belong. Ext-net & Network1 are shared networks which means every user in the cloud can see them and connect to them. The difference between Admin and other users are the provider information -only visible by Admin- network-type, physical network and segmentation.

Only IP addresses attached to the VMs are linked with Security groups. However, IP addresses assigned to ports in the router, DHCP or belong to an external network (floating IP) are not connected to a Security group.

The command *neutron router-port-list $router_id* lists all ports connected to specific router, and five commands were necessary to get the full picture:

- Command *neutron port-list* gave the port details,
- Command *neutron port-show* $port_id gave the router id that connected to,
- Command *neutron router-list* & *neutron router-show $router_id* gave the router id and name and ports from external network, not the private network,

Command	**Gathered Information**	**Description**
neutron port-list	port ID, MAC, fixed IP(subnet id, IP)	List all the ports in each network created by neutron.
neutron port-show $port_id	ID, host, vif_typ, device ID, device owner, fixed IP (subnet id, IP), MAC, network ID, tenant.	Using the ID of each port from the previous command to show detailed information about each port.
neutron router-list	ID, name, net id, subnet id, IP	List all the routers in the cloud infrastructure
neutron router-show $router_id	ID, name, net id, subnet id, IP, tenant ID	Using the router ID from the previous command to show all the details of each router
neutron router-port-list $router_id	ID, MAC, subnet ID, IP	Using router ID as in the above command to list all the ports of each router
neutron floatingip-list	ID, Fixed_ip, floating IP, port ID (fixed ip port)	List the floating IP provided by neutron
neutron floatingip-show $floatip_id	ID, fix IP, float ip, float net ID, pot id (fixed), router id, tenant ID	Expand the details of each floating IP using the Floating IP ID from the previous command.

Table 3.7: OpenStack Command to Display the Cloud Network Devices and Ports Details.

- Command neutron router-port-list $router_id listed router id and port ids linked to (both external and internal).

By matching the router IDs and port IDs we obtained information about which port connected to which router. In other words, no direct command was available to obtain the router name matched with the port IP, hence we used several commands to achieve this.

The *neutron floatingip-list* command lists only the floating IP address that is associated with VMs IPs, which matches the information in Table A.3 in Appendix A. The floating ID listed before under the command *neutron port-show $port_id* with Device ID for VM ports. The *neutron floatingip shows $floatip_id* listed as the floating ID, and floating IP as the associated VM IP. The port ID of the VM IP is the router ID that contains the floating IP and the router Tenant ID. This means that although the floating IP is associated with VM IP, which is physically located in the Compute Node, the floating IP is located in a router owned by a tenant, which in our case is the same owner of the VM. Logically fixed and floating IPs assigned to the same VM, but not in the same physical location as the router, means keeping the floating IP contained in its namespace in the Network/Controller Node for NAT processing of the external connectivity.

At this point we understood where each port was physically resident. The command *neutron port-show $port_id* provided information called binding:host_id which indicated which Node controller | computer the port existed in.

Moreover, the command *neutron port-show $port_id* provided another information called binding:vif_type. If the result is OVS that means this port belongs to the OVS switch but we do not know which OVS switch it belongs to.

Another useful piece of information is device_owner and its meaning is as follows:

- network:router_interface: network means usually located in the Controller Node, where the NaaS runs, and it is an interface of the router namespace. Most likely this port belongs to a private network within the cloud.

- compute:nova: this in the Compute Node and belongs to an instance with a private ip address on the internal network the instance is connected to.

- network:router_gateway : this in the Controller Node and contains an ip address from the external network which is a router (namespace) gateway the instance uses to connect to the outside.

- network:floatingip: although this port was assigned to the network (Controller Node), the binding host was not specified and it is a floating ip address associated with an instance.

- network:DHCP: this port is in the Controller Node and belongs to a namespace called DHCP. It holds a private network ip address.

One more useful piece of information; as mentioned earlier in §**3.3.3**, is that the port ID has been used to define the actual port within the infrastructure. As shown in Figure 3.7, each port in the diagram starts with qr-,qg-,tap, qbr,qvo,qvb followed by 11 characters from the port ID.

We gathered the following information about the ports including:

- Which node they are located in,
- Which network and subnet they belong to,
- What is their purpose? e.g. Instance interface, router interface, gateway, instance associated floating ip address, DHCP interface.
- Were they created by OVS or not?

However, we do not have information about where they are located in each node and how the node is connected. There is more information gathered that will help us decipher all these tangled concepts to clarify the construction of the DeepDive diagram in the next sections.

3.4.5 Compute Node

The Compute Node is responsible for configuring the hypervisor, creating the VMs and linking them to Linux and OVS bridges.

3.4.5.1 Physical ports

The following Table 3.8 shows the interfaces visible at the the Compute Node: number, Name, MAC, IP, Master port are all from using the command: *$ ip a*

Important points derived (from the script file alongside the limited information from the Testbed configuration):

1. The information in this part is primarily gathered from the script file alongside the limited information from the Testbed configuration.
2. LO is a loopback interface, to be ignored, as OpenStack does not use it.
3. eth0, eth1 are physical ports of the compute node and their IPs tell us that eth0 belongs to the management network and eth1 belongs to the VBOX NAT network (internet).

# list	Name	MAC	IP	Master
1	Lo	00:00:00:00:00:00	127.0.0.1/8	
2	eth0	08:00:27:1b:ac:9e	192.168.137.11/24	
3	eth1	08:00:27:6e:29:45	10.0.3.15/24	
4	virbr0	52:54:00:50:87:52	192.168.122.1/24	
5	virbr0-nic	52:54:00:50:87:52		
6	ovs-system	a6:c4:43:1d:0c:48		
7	br-int:	72:03:0e:65:06:45		
8	br-tun	22:81:62:cd:2c:48		
9	qbr3901d884-71	8e:f4:a8:db:29:7f		
10	qvo3901d884-71	22:09:99:ea:5b:71		ovs-system
11	qvb3901d884-71	8e:f4:a8:db:29:7f		qbr3901d884-71
12	tap3901d884-71	fe:16:3e:62:a6:c3		qbr3901d884-71
13	qbrb3269f22-5d	82:9a:91:0f:ea:72		
14	qvob3269f22-5d	96:4a:d1:f9:55:96		ovs-system
15	qvbb3269f22-5d	82:9a:91:0f:ea:72		qbrb3269f22-5d
16	tapb3269f22-5d	fe:16:3e:aa:8b:b8		qbrb3269f22-5d
17	qbre381d0a8-3d	06:0e:92:0c:78:e2		
18	qvoe381d0a8-3d	86:18:01:f8:bb:6f		ovs-system
19	qvbe381d0a8-3d	06:0e:92:0c:78:e2		qbre381d0a8-3d
20	tape381d0a8-3d	fe:16:3e:05:a6:8e		qbre381d0a8-3d
21	qbrae8e6ab5-b8	ba:8f:6b:74:81:66		
22	qvoae8e6ab5-b8	a2:2c:8e:19:1a:28		ovs-system
23	qvbae8e6ab5-b8	ba:8f:6b:74:81:66		qbrae8e6ab5-b8
24	tapae8e6ab5-b8	fe:16:3e:00:ca:d3		qbrae8e6ab5-b8

Table 3.8: Compute Node Ports List.

4. virbro0 and virbr0-nic belong to libvirt, the core of the KVM hypervisor, and according to [139] they are to be ignored, as the Cloud does not use them.

5. Br-int and Tunnel Bridge (br-tun) are local interfaces of the br-int and br-tun OVS bridges

6. Group of ports starting with qbr,qvo,qvb,tap share the same hex number (11 characters including the -). The sequence of ports with the same numbering belongs to the same instance to form the communication chain from the tenant instance starting with the tap and ending up with qvo in the br-int OVS bridges. The numbering associated with ports is produced by the commands in Table 3.7.

7. The port tap is the direct port that is attached to the instance. Meanwhile, qbr represents a Linux Bridge, which is used solely to manage the instance to be defined with a security group.

8. The ports qvb and qvo are veth pair connections between OVS and Linux bridges, where the qvb is located in the qbr Linux bridge and the qvo are located in the OVS br-int bridge.

9. Although all the potential ports included in the network infrastructure are listed, the way they are linked to the system is not clear yet.

3.4.5.2 Instance

The command *nova list* displays the list of VM instances created along with their IP addresses. This indicates how VMs are connected to the cloud network[144].

This part of information collection was done through the script files and our source of information is the output files T1user1-output.txt, T1user2-output.txt and Test-output.txt.

Table 3.9 lists result found by the commands *Nova list, nova show $VM_ID*. Such information are the VMs, IDs, Names, Network IPs, owners Tenant, user that are visible.

Name	**T1u1VM**	**T1u2VM**	**KaliLight**	**My_first_instance**
ID	4ce1c9d4-deab-4945-b5e1-1cb913929dda	09d6ce76-2d84-469c-b51e-9122a067d92d	7a582f9a-799c-41a6-8ee6-f67ee2f115d8	6114319c-936b-41b3-baba-3d9dbbd4e9d8
Network	t1user1net	t1user2net	Network1	Network1
IPs	192.168.10.3, 192.168.100.14	192.168.20.3, 192.168.100.15	10.190.0.7, 192.168.100.16	10.190.0.6, 192.168.100.11
Image	CirrOS 0.3.4	CirrOS 0.3.4	KaliLight2	CirrOS 0.3.4
Owner: tenant	0689c18aa8324c83 add7472625d74e5c	0689c18aa8324c83 add7472625d74e5c	59bde68f1a7a4082 9c8b67e098c295bf	59bde68f1a7a4082 9c8b67e098c295bf
Owner: user	0e291d5fff7746c1 afa4c4369e6ad8c8	bfac39ffd3b343f3 93193240d432a24e	d548a1c68d5a4336 8644128a468a029a	d548a1c68d5a4336 8644128a468a029a
Visible to	T1user1, T1user2	T1user1, T1user2	Test	Test

Table 3.9: Output of the Commands Nova List and Nova Show for the Two Tenants: Test and Tenant1

Important points derived (from Table 3.12) include:

1. Table 3.9 shows the instance (name and ID) and the related private network (name and IP), the floating IP addresses used by the tenant to connect to the instance externally, and the ownership of the instance at the level of the user and tenant.

2. Both users t1user and t1user2 belong to Tenant 1, but each user has their own network and instances, yet users for the same tenant can see each other's instances and content details.

3. The Test tenant has only one user called Test and two created instances called KaliLight and my_first_instance. Both instances are visible only by the Test tenant and user.

4. At this level, we can draw the instances and their ownership (tenant/user), but we do not have enough information to connect each instance to the correct Linux Bridge and tap port. This step can be achieved with the information from Table 3.10.

Kofman [144] provided a clear explanation on tracking the connection of an instance to its network, through using the instance "definition file". It is an XML file "libvirt.xml" created by the hypervisor to define the instance configuration and this file is saved in the compute file under the path "/var/lib/nova/instances/<instance-id>/".

The network configuration of the xml file for the instance "my_first_instnce" is shown below. Similarly, all other instance information is gathered in Table 3.10:

```
<interface type="bridge">
     <mac address="fa:16:3e:62:a6:c3"/>
     <model type="virtio"/>
     <driver name="qemu"/>
     <source bridge="qbr3901d884-71"/>
     <target dev="tap3901d884-71"/>
   </interface>
```

All instance XML files have been gathered using a "compute.sh" script and the output stored in the output file "Compute-output-mar17.txt". Using the VM ID from the command *nova list* and using the values of VM_ID , from Table 3.9, generates the content (cat) of the instance XML file (libvirt.xml) existing under /var/lib/nova/VM_ID/.

Table 3.10 lists information from the XML file relating to the VM such as: ID, instance name, nova name, user ID & Name, project, interface type, mac, source bridge and target dev.

Nova Name	T1u1VM	T1u2VM	KaliLight	my_first_instance
ID	4ce1c9d4-deab-4945-b5e1-1cb913929dda	09d6ce76-2d84-469c-b51e-9122a067d92d	7a582f9a-799c-41a6-8ee6-f67ee2f115d8	6114319c-936b-41b3-baba-3d9dbbd4e9d8
Instance Name	instance-00000002	instance-00000003	instance-00000004	instance-00000001
User ID	0e291d5fff7746c1 afa4c4369e6ad8c8	bfac39ffd3b343f3 93193240d432a24e	d548a1c68d5a4336 8644128a468a029a	d548a1c68d5a4336 8644128a468a029a
User Name	t1user1	t1user2	test	test
Project ID	0689c18aa8324c83 add7472625d74e5c	0689c18aa8324c83 add7472625d74e5c	59bde68f1a7a4082 9c8b67e098c295bf	59bde68f1a7a4082 9c8b67e098c295bf
Project Name	Tenant1	Tenant1	test	test
interface type	bridge	bridge	bridge	bridge
Mac	fa:16:3e:aa:8b:b8	fa:16:3e:05:a6:8e	fa:16:3e:00:ca:d3	fa:16:3e:62:a6:c3
Source bridge	qbrb3269f22-5d	qbre381d0a8-3d	qbrae8e6ab5-b8	qbr3901d884-71
Target dev	tapb3269f22-5d	tape381d0a8-3d	tapae8e6ab5-b8	tap3901d884-71

Table 3.10: Instances Definition File Significant Information.

Important points derived:

- The extra information added from Table 3.9 to Table 3.10 includes the instance name (the local name associated with the creation files), user name and ID (easy to distinguish which user it belongs to), project Name and ID (easy to distinguish which tenant it belongs to), MAC address of instance eth0 to capture any traffic in the future and the name of the bridge and the tap port which indicate which Linux bridge the instance is connected to.

The sequence of ports connected to the instances are shown in Figure 3.7.

3.4.5.3 Linux Bridges

Besides the information in §3.3.4 and Table 3.10 further information was required to understand how Linux bridges facilitate the connection. Therefore, using the Compute.sh script, the linux bridge command lines: *brctl show* and *sudo brctl showmacs $linux_br*. This section's output was organized and analysed in Table 3.11. The Linux Bridge details are only internal details that can not be found from the Dashboard or the Testbed configuration.

Table3.11 lists the linux bridge, id, interfaces names, interfaces # and MAC address

Bridge Name	Bridge ID	Interfaces	Port No	MAC
qbr3901d884-71	8000.8ef4a8db297f	qvb3901d884-71	1	8e:f4:a8:db:29:7f
		tap3901d884-71	2	fe:16:3e:62:a6:c3
qbrae8e6ab5-b8	8000.ba8f6b748166	qvbae8e6ab5-b8	1	ba:8f:6b:74:81:66
		tapae8e6ab5-b8	2	fe:16:3e:00:ca:d3
qbrb3269f22-5d	8000.829a910fea72	qvbb3269f22-5d	1	82:9a:91:0f:ea:72
		tapb3269f22-5d	2	fe:16:3e:aa:8b:b8
qbre381d0a8-3d	8000.060e920c78e2	qvbe381d0a8-3d	1	06:0e:92:0c:78:e2
		tape381d0a8-3d	2	fe:16:3e:05:a6:8e
virbr0	8000.52540050875	virbr0-nic	1	52:54:00:50:87:52

Table 3.11: Linux Bridge Lists and Details.

The command *sudo brctl showmacs $linux_br*, only list the port number and the MAC without the interface names. The MAC and interface numbers had been matched using the Table 3.8 under the computer physical interface.

Important points derived:

1. Table 3.11 shows the Linux bridges (qbr) and the ports belonging to it. Every instance created by the tenant creates a Linux bridge with a tab and qvb ports and a qvo port in the br-int

OVS Bridge. All the ports share the same named 11 characters derived from the instance port ID as explained in §3.3.3.

2. From §3.3.4 we know that the qvo port in br-int is listed to represent the connection from br-int to the instance. Due to the current technology of OpenStack and the use of security groups when creating the instance (iptables) the instance cannot be connected to the OVS bridge directly, therefore a Linux bridge is the intermediate solution to connect the two ends.

3. The qvo port is the veth pair to link the OVS and Linux Bridge through the port qvb. We can draw the connection by matching the numbers associated with each of the three ports (tap, qvb,qvo).

3.4.5.4 OVS

Proceeding with the connection of the infrastructure, we need to track the connection from the Linux bridge to the OVS bridge br-int. To do so, we configured the Compute.sh script with several OVS commands listed below and recorded the result in Compute-output-mar17.txt, as the most significant information is displayed in Table 3.12. Similar to the Linux bridge, the OVS information sources gathered via the script files only.

OVS details were collected using the commands:

- *sudo ovs-vsctl list-br*: lists all OVS bridges created in the Node.
- *sudo ovs-vsctl list-ifaces br-int*: lists all the interfaces listed in br-int bridge.
- *sudo ovs-vsctl list-ifaces br-tun*: lists all the interfaces listed in br-tun bridge.
- *sudo ovs-vsctl show*: shows all the OVS bridges and their associated interfaces, Tags, patches and extra information.
- *sudo ovs-ofctl show br-int*: lists br-int interfaces, listed #, MAC
- *sudo ovs-ofctl show br-tun*: lists br-tun interfaces, listed #, MAC

Table 3.12 combines the result from the above commands to represent the following information:

Important points derived:

1. Br-int bridge contains several ports: the Veth link port with br-tun (patch-tun) and the br-int local port, the qvo represent the connection to the instance created tenants. qvo is tagged by a VLAN number assigned locally by OpenFlow. In addition, qvo is a Veth link between

Bridge			br-int			
	Port	**Tag**	**Type**	**option**	**Listed #**	**MAC**
	patch-tun		patch	peer=patch-int	1	ea:f4:eb:12:f1:d8
	qvo3901d884-71	1			2	22:09:99:ea:5b:71
	qvoae8e6ab5-b8	1			5	a2:2c:8e:19:1a:28
	qvob3269f22-5d	2			3	96:4a:d1:f9:55:96
	qvoe381d0a8-3d	3			4	86:18:01:f8:bb:6f
	br-int		internal		LOCAL	72:03:0e:65:06:45
Bridge			br-tun			
	Port	**Tag**	**Type**	**option**	**Listed #**	**MAC**
	patch-int		patch	peer=patch-tun	1	3e:66:11:e0:b7:8a
	br-tun		internal		LOCAL	22:81:62:cd:2c:48
	vxlan-c0a8890b		vxlan	local_ip="192.168.137.11 remote_ip="192.168.137.10"	2	22:db:bd:64:0e:6d

Table 3.12: Compute Node OVS Details.

the OVS and the Linux bridge, where both ports (OVS qvo and Linux bridge qvb) were identified with the same sequence of numbers that matches the first 11 characters of the instance port ID.

2. In Table 3.12 we have 4 qvo ports, two of them have the same VLAN tag, which means that those ports belong to two different instances within the same network. The other two qvo ports are each assigned different VLAN tags that means those ports are related to instances in different networks.

3. Patch-int is a patch port from the veth link connection between br-tun and br-int with its Veth port patch-tun.

4. The VXLAN interface is the tenant VN network type defined in the Neutron configuration, linked to the node marked local IP 192.168.137.11. The traffic destination would be to the other node with IP 192.167.137.10. This means that the VXLAN tunnel is used by all tenant traffic isolated by VXLAN code for each tenant. VXLAN code can be found in the flow rules that will be listed using the command *sudo ovs-ofctl show <bridge-name>*.

5. In the br-tun bridge, the port vxlan-c0a8890b represents the tunnelling port that br-tun uses to transfer the traffic from the OVS environment to the physical connection. The option column illustrates the connection between the two nodes the local IP represents the eth0 interface of the compute Node, and the remote IP represents the eth0 controller IP.

Figure 3.7 illustrates the full connection sequence in the Compute node. Thus, after having covered half of Figure 3.7, the subsequent sections describe the Controller node to complete the full picture.

3.4.6 Controller Node

The Controller Node contains the majority of the infrastructure for the Cloud Networking such as the Horizon Dashboard and the database. The Compute node connects through the VXLAN tunnel to the Controller. Similarly to the Compute node, the controller.sh script file was configured to produce the results reported in the subsections below. All the outputs were saved in the controller-output-mar17.txt.

3.4.6.1 Physical ports

The following Table 3.13 shows the interfaces visible at the Controller Node level: port number, Name, MAC, IP, which can be displayed by using the command: *$ ip a*.

# in the list	Name	MAC	IP
1	Lo	00:00:00:00:00:00	127.0.0.1/8
2	eth0	08:00:27:62:08:29	192.168.137.10/24
3	eth1	08:00:27:5f:9d:7b	
4	eth2	08:00:27:82:cc:17	10.0.4.15/24
5	ovs-system	8e:84:89:13:9a:02	
6	br-int	6e:ee:e6:50:5f:41	
7	br-ex	08:00:27:5f:9d:7b	
8	br-tun	56:54:ec:de:f8:4	

Table 3.13: Controller Node ports list.

Other than the main physical interfaces of the controller (eth0, eth1 and eth2), there are three additional interfaces: Br-int, br-ex and br-tun, which are local interfaces of the OVS bridges. This indicates that similarly to the Compute node, this node uses OVS from the port ovs-system and the three interfaces that represent their three ovs bridges. As discussed in §3.2.4, br-ex is a key requirement in the controller of the cloud setup.

3.4.6.2 OVS

Table 3.14 lists the OVS commands used in the Controller Node, which outnumber the commands used in the Compute node. As the Controller node creates an extra OVS bridge, as described in §3.2.4, an additional list of OVS commands is used to collect the details of that bridge. Table 3.15 displays the OVS output of the commands listed in Table 3.14.

Command	Gathered information	Description
sudo ovs-vsctl list-br:.	OVS bridges	lists all OVS bridges created in the Node
sudo ovs-vsctl list-ifaces br-int:	Interfaces	lists all the interfaces listed in br-int bridge.
sudo ovs-vsctl list-ifaces br-tun:	Interfaces	lists all the interfaces listed in br-tun bridge.
sudo ovs-vsctl list-ifaces br-ex:	Interfaces	lists all the interfaces listed in br-ex bridge
sudo ovs-vsctl show:	OVS bridges, Interfaces, VLAN Tag, patched interfaces	shows all the OVS bridges and their associated interfaces, Tags, patches and extra information.
sudo ovs-ofctl show br-int:	Interfaces #, Name, MAC	lists br-int interfaces, listed #, MAC
sudo ovs-ofctl show br-ex:	Interfaces #, Name, MAC	lists br-ex interfaces, listed #, MAC
sudo ovs-ofctl show br-tun:	Interfaces #, Name, MAC	lists br-tun interfaces, listed #, MAC

Table 3.14: Controller Node OVS Command List.

Table 3.15 combines the result from the above commands to represent the following information about bridges and interfaces as follows.

Important points derived:

1. Picking up the connection tracking where we stopped in the Compute node, the traffic transferred from the tunneling connection from the Compute node is received by the tunneling bridge (br-tun) in the Controller node.

2. The name of thetunneling interface vxlan-c0a8890b in br-tun matches the VXLAN interface in the Compute Node br-tun bridge as shown in Table 3.12. This means that vxlan-c0a8890b is one tunnel connecting the two nodes using the same tunnel code for each tenant traffic.

3. As in the Compute Node, br-tun and br-int are connected through the veth link using the option peer. If the option column listed peer=interface-name info, then it is a veth linked (peered) with another veth from another bridge

4. Similarly, the bridges br-int and br-ex are connected through veth link in the same way phy-br-ex in br-ex bridge is linked to int-br-ex in br-int.

5. Br-int consists of another type of interface; some start with "tap" and others start with "qr". The numbering system in the Controller node is different form that of the Compute node, however they can be grouped using their tag number.

Bridge				Br-ex		
	Port	**Tag**	**Type**	**option**	**Listed #**	**MAC**
	eth1				1	08:00:27:5f:9d:7b
	br-ex		internal		LOCAL	08:00:27:5f:9d:7b
	phy-br-ex		patch	peer=int-br-ex	2	fe:31:88:a5:85:a4
	qg-3e2b0e76-9a		internal		5	
	qg-c7b01624-a8		internal		4	
	qg-e3372ecf-1c		internal		3	
Bridge				br-int		
	Port	**Tag**	**Type**	**option**	**Listed #**	**MAC**
	int-br-ex		patch	peer=phy-br-ex	1	12:bd:ad:92:77:5d
	patch-tun		patch	peer=patch-int	2	a2:88:a4:e4:78:fd
	br-int		internal		LOCAL	6e:ee:e6:50:5f:41
	qr-350d30d0-3b	1	internal		4	
	qr-6db257fa-4b	3	internal		8	
	qr-986e808c-9e	2	internal		6	
	tap750abee1-f3	2	internal		5	
	tapc532ed28-1c	3	internal		7	
	tape593bcca-7b	1	internal		3	
Bridge				br-tun		
	Port	**Tag**	**Type**	**option**	**Listed #**	**MAC**
	br-tun		internal		LOCAL	56:54:ec:de:f8:4c
	patch-int			peer=patch-tun	1	ee:92:70:fe:ba:fb
	vxlan-c0a8890b		vxlan	local_ip="192.168.137.10", remote_ip="192.168.137.11"	2	f2:c3:da:b7:43:10

Table 3.15: Controller Node OVS Bridges and Interface Details.

6. br-int interfaces are tagged with numbers (VLAN Tag). Figure 3.7 lists all ports with the same tag next to each other as they share the same VLAN since they belong to the same network.

7. At this point we have listed the ports and their tags in each bridge as shown in Figure 3.7 lists ports and their tags in each bridge without any additional information, such as IPs or where they are connected to cloud network infrastructure.

3.4.6.3 Namespaces

This was one of the hardest to find points in the DeepDive experiment as this information was not visible either through the Testbed or the Dashboard. Namespaces exist only in the Controller Node and their details can only be retrieved using the commands in the controller.sh script.

Table 3.16 & 3.17contains namespace details collected using the following commands, where $x is qdhcp_ns or qrouter_ns:

- *ip netns*: lists namespaces (qdhcp, qrouter).
- *sudo ip netns exec $x ip a*: port list, port name, MAC, IP, Broadcast IP (Bcast), glopal scope.

- *sudo ip netns exec $x ifconfig*: port name, MAC, IP, Bcast
- *sudo ip netns exec $x ip link*: port list, port name
- *sudo ip netns exec $x route -n*: Destination, gateway, use iface.
- *sudo ip netns exec $x ip route*: default IP, GW dev, network, dev, scope link src.

NameSpace		qdhcp-2ef08b44-ef89-40ff-a4bc-4e11a70acd6d			
		Port info			
Port list	**Port name**	**IP**	**MAC**	**Bcast**	**Scope Global**
15	tapc532ed28-1c	192.168.20.2/24	fa:16:3e:91:60:23	192.168.20.255	tapc532ed28-1c
Routing info (GW)		**Routing info (Network)**			
Gateway/default IP	**Use Iface/dev**	**Net range**	**Use Iface/dev**	**Scope link src**	
192.168.20.1	tapc532ed28-1c	192.168.20.0	tapc532ed28-1c	192.168.20.2	
NameSpace		qdhcp-839e7c38-44bf-4865-b162-7f9324be8a8c			
		Port info			
Port list	**Port name**	**IP**	**MAC**	**Bcast**	**Scope Global**
12	tap750abee1-f3	192.168.10.2/24	fa:16:3e:ea:1b:e1	192.168.10.255	tap750abee1-f3
Routing info (GW)		**Routing info (Network)**			
Gateway/default IP	**Use Iface/dev**	**Net range**	**Use Iface/dev**	**Scope link src**	
192.168.10.1	tap750abee1-f3	192.168.10.0/24	tap750abee1-f3	192.168.10.2	
NameSpace		qdhcp-61389eed-95de-4290-989b-006e44c19413			
		Port info			
Port list	**Port name**	**IP**	**MAC**	**Bcast**	**Scope Global**
9	tape593bcca-7b	10.190.0.5/24	fa:16:3e:f5:47:a5	10.190.0.255	tape593bcca-7b
Routing info (GW)		**Routing info (Network)**			
Gateway/default IP	**Use Iface/dev**	**Net range**	**Use Iface/dev**	**Scope link src**	
10.190.0.1	tape593bcca-7b	10.190.0.0 /24	tape593bcca-7b	10.190.0.5	

Table 3.16: Controller Node DHCP Namespaces Details.

Important points derived from Table 3.16 & 3.17:

1. Using the information from Table 3.16 & 3.17, the name spaces are linked to an interface that belongs to the OVS bridges as shown in Table 3.15. Therefore the namespaces are located between br-int and br-ex (see Figure 3.7) where the namespaces are linked to interfaces from both bridges.

2. qdhcp namespace is a DHCP server associated with each VN created by a tenant, which is linked to a tap interface in br-int where the ip address of the tap is the ip address to represents the DHCP server.

3. qrouter is a virtual router that the tenant creates and is linked to two interfaces; (1) the gateway (qr-# from br-int bridge) of the private VN the tenant created earlier and (2) an

NameSpace	qrouter-b9211dbd-0f99-4912-9565-18793cad5f3b				
Port info					
Port list	Port name	IP	MAC	Bcast	Scope Global
16	qr-6db257fa-4b	192.168.20.1/24	fa:16:3e:11:18:87	192.168.20.255	qr-6db257fa-4b
17	qg-3e2b0e76-9a	192.168.100.13/24 192.168.100.15/32	fa:16:3e:61:6b:f7	192.168.100.255 192.168.100.15	qg-3e2b0e76-9a
Routing info (GW)		Routing info (Network)			
Gateway/default IP	Use Iface/dev	Net range	Use Iface/dev	Scope link src	
192.168.100.1	qg-3e2b0e76-9a	192.168.20.0/24 192.168.100.0/24	qr-6db257fa-4b qg-3e2b0e76-9a	192.168.20.1 192.168.100.13	
NameSpace	qrouter-84d7db89-8cf1-45fb-9383-b92f47b85117				
Port info					
Port list	Port name	IP	MAC	Bcast	Scope Global
13	qr-986e808c-9e	192.168.10.1/24	fa:16:3e:66:3a:58	192.168.10.255	qr-986e808c-9e
14	qg-c7b01624-a8	192.168.100.12/24 192.168.100.14/32	fa:16:3e:5a:08:14	192.168.100.255 192.168.100.14	qg-c7b01624-a8 qg-c7b01624-a8
Routing info (GW)		Routing info (Network)			
Gateway/default IP	Use Iface/dev	Net range	Use Iface/dev	Scope link src	
192.168.100.1	qg-c7b01624-a8	192.168.10.0/24 192.168.100.0/24	qr-986e808c-9e qg-c7b01624-a8	192.168.10.1 192.168.100.12	
NameSpace	qrouter-870b7e86-5674-40b5-9ec4-43e0f81c29eb				
Port info					
Port list	Port name	IP	MAC	Bcast	Scope Global
10	qr-350d30d0-3b	10.190.0.1/24	fa:16:3e:21:f6:3e	10.190.0.255	qr-350d30d0-3b
11	qg-e3372ecf-1c	192.168.100.10/24 192.168.100.11/32 192.168.100.16/32	fa:16:3e:d0:0e:a9	192.168.100.255 192.168.100.11 192.168.100.16	qg-e3372ecf-1c
Routing info (GW)		Routing info (Network)			
Gateway/default IP	Use Iface/dev	Net range	Use Iface/dev	Scope link src	
192.168.100.1	qg-e3372ecf-1c	10.190.0.0/24 192.168.100.0/24	qr-350d30d0-3b qg-e3372ecf-1c	10.190.0.1 192.168.100.10	

Table 3.17: Controller Node vRouter Namespaces Details.

interface (qg-# from be-ex) with the external network IP and qrouter. To provide the instances external access, NAT rules are used between the private and public subnets.

4. qg-# ports are associated with a list of IPs from the external network. The first IP from the list is usually with /24 is the IP for the port. The remaining list, usually with /32, are the floating IPs associated with the tenant instances(VMs) with private IPs belongs to the private network that the qr-# belong to.

5. A question remains of how to identify which elements belong to which tenant?

   a) Networks and IP address:

      i. Every network shown in Table 3.6 is associated with a subnet. Admin owns 2 networks, ext-net and Network. Tenant 1 owns T1user1net and t1user2net.

   ii. IPs from ext-net are shown only on the side of the br-ex bridge.

   iii. As Network1 is created by Admin as a shared network, all the tenants can see Network1 and the Test tenant uses this network as its private network IPs for its instances.

   iv. As shown in Table 3.9, VMs owned by a specific tenant and usera are assigned private IP addresses from the same range as in tap-# and qr-#. Moreover, the instances associated with floating IP addresses are listed within qg-# in qrouter-#.

b) Owners of namespaces and ports:

   i. The DHCP namespace is defined as qdhcp-network ID. The network ID attached to qdhcp can be matched with network ID in Table 3.6 shows ownership.

   ii. The router namespace is defined as qrouter-router ID, hence, the data analysis of Table 3.6 prove the ownership of the router. Although Router1 is owned by Admin, and it is used by Test tenant, the router and the router floating gateway is not visible by the Test tenant. However,data obtained from the scripts output files indicates that Router 1 is connected to Network1, which is used by Test.

   iii. As mentioned in §3.4.4 ports are defined as qr-/tap-/qg- first 10 hex numbers of port ID as has been explained before in .

   iv. Ports defined for floating IP addresses are not presented with interfaces.Instead, they are associated with instances shown in Table 3.9. Ports assigned local IPs in VMs and associated with floating IP are listed in qrouter namespaces under qg-10 hex of port ID in Table 3.17.

3.5 Technical Implications

3.5.1 OpenStack Testbed Construction

The following points indicate some of the major technical obstacles of building an OpenStack TestBed:

1. It took six trials to build a single-node Demo cloud between Hyper-v and VirtualBox environment for the cloud Nodes. Different Ubuntu versions were examined (Ubuntu 12.02 and 14.04) and different cloud repositories tried (icehouse and Juno), but the cloud builds repeatedly failed.

   a) The Demo Cloud failed to install with DevStack, which was not flexible enough to support our planned testbed. The successful demo single-node was built with VirtualBox, Ubuntu Server 14.0 and devstack /OpenStack-Juno. However, the physical infrastructure server (Gorman) was supported with a Hyper-v virtualization environment. This Demo cloud was a good base reference for the intended testbed.

2. The planned testbed was a multi-node (3 node) OpenStack cloud, which was not successful with DevStack and hence the cloud was built manually using OpenStack Installation Guide for Ubuntu 14.04 – Kilo[147] with GRE. Trial 7 was the first successful multi-node OpenStack Cloud.

   a) The cloud operation was healthy except that the External Network connection failed and Instances were not connecting with outside the cloud Nodes boundaries.

   b) After a long period and countless trials of solutions to resolve the cloud external network issue, we found that Hyper-v External vSwitch connecting the instance to the outside was incompatible with the external OVS switch in the Network Node and the connection was blocked in between. The Hyper-V switch connectivity issue solution was unsolvable, even with expert help.

   c) As a solution we decided to change the virtualization environment to VBox as was the original Cloud Demo. However, at that time OpenStack-Mitaka was released and the installation guide set the cloud structure to be built on two nodes rather than three nodes.

      i. A 2-node OpenStack-Mitaka cloud on Hyper-V (trial 8) had the same external network connection issue. Hence the Hyper-V virtual environment was disabled and replaced with a VBox in the Gorman Server.

      ii. OpenStack–Mitaka (2-Node) cloud with VXLAN was successfully deployed and the External Network connection was successful (trial 9).

      iii. Tenant scenarios, the DeepDive methodology and the Penetration Testing methodology were all repeated from the 3-nodes OpenStack-Kilo Cloud testbed to a 2-node OpenStack-Mitaka cloud testbed.

3.5.2 OpenStack DeepDive Application

The following are the technical obstacles of the DeepDive method:

1. Although the Demo single-node OpenStack cloud was built on one server, the planning of the DeepDive took longer than expected to study the infrastructure as the documentation

was not as clear and available as we had expected. We found the proper commands and identifying the cloud's significant information was not easy.

2. Moreover, due to our initial limited knowledge in cloud systems, their infrastructure and associated information, coming up with the best way to analyse the collected information manually in the first attempt took significantly longer than expected.

3. A redesign of the DeepDive method to move from a single node cloud to a 3-node cloud was necessary as the infrastructure differed in both clouds, the amount of collected data was even larger and the manual data analysis took longer than for demo cloud.

4. Upon changing the OpenStack Cloud from a 3-node Kilo to a 2-node Mitaka, the cloud infrastructure changed and the DeepDive method had to be adjusted.

5. Chapter 4 contains the last DeepDive method procedure, however, the manual data analysis is still appropriate but the time spent remained significantly long.

3.6 Conclusions

This Chapter started by achieving the second research objective of discovering the required resources to evaluate Cloud Virtual Network Isolation by building the OpenStack testbed. This was followed by achieving the third research objective of discovering the best methodology to acquire infrastructure information and resulted in a contribution of developing the DeepDive methodology. The DeepDive was applied on the testbeds to reveal the components and configuration of the cloud network infrastructure.

Due to the nature of this exploratory research, we only managed to ground the hypothesis after achieving the third objective as it clarified for us what to be tested.

The final outcome of this chapter is a complete OpenStack Network infrastructure diagram (Figure 3.7 on page 87) showing the tenant elements and components such as VMs, Networks, Routers and DHCP servers. The diagram clarifies and identifies the isolation mechanisms after the process of gathering information, analysis and visualisation. The VN isolation mechanisms used by this cloud testbed setup were:

1. VLAN: locally defined by Openflow within the OVS switches to isolate a tenant's traffic. Each network is assigned to a specific VLAN. Note that Neutron dashboard is not aware of these isolation mechanisms as it is handled directly by OpenFlow controllers.

2. VXLAN: The tenant network type is defined by Neutron. Each VXLAN tunnel is created between two cloud nodes where all the tenants share the VXLAN tunnel and traffic isolated by the VXLAN code is tagged in the tenant traffic packets header. Therefore tagging is used as an isolation mechanism.

3. Namespaces: A Linux isolated container containing the tenant's network configurations. Our Testbed setup shows two types of namespaces. DHCP namespace, which represents a DHCP server created by Neutron when a tenant creates a network. Router namespace, which represents the local router created by the tenant. This links the inner side of the cloud (tenant private network) with the outside (External network) world to provide the instances external connection. Therefore namespaces are used to isolate tenant network configurations.

The DeepDive was the most involved experiment performed in this research. The aim of the experiment was to discover whether the identities of the tenant and user connections in a commercially used open-source system could be derived. Theoretically router and bridge identifications should not be accessible by users. However this DeepDive experiment has demonstrated that with the use of multiple commands throughout the cloud architecture, any user can find private information, which can compromise tenants and users. This experiment provides a methodology for cloud network isolation analysis.

DeepDive did not only provide a graphical representation of the cloud infrastructure but it identified the cloud network infrastructure, VN components and configurations. This information was used as our cloud infrastructure and isolation knowledge guide.

The work described above is followed by experiments discussed in Chapter 5 and 6, wherein a publicly available penetration testing tool is placed inside a Cloud tenant and outwith a cloud to simulate attackers both internally and externally.

These experiments were aimed at determining if an attacker can duplicate the extensive Deep Dive, that is, finding Cloud Virtual Network Isolation information or tenant information from different connection vectors. In other words, the experiments demonstrate whether Virtual Network Isolation mechanisms can be learned by other tenants or external users users as stated in the Research Aim of detecting cloud virtual network isolation information.

CHAPTER FOUR

CLOUDSTACK IAAS TESTBED AND INFRASTRUCTURE DEEPDIVE

This chapter follows a similar structure to that of Chapter 3 as it aims to answer two of the research questions which are;

What are the required resources to evaluate CVNI? and

What is the best methodology to identify the construction of the cloud network infrastructure, VN components and Isolation mechanism?

To answer the first of these research questions, this chapter illustrates extensively the process of setting up the CloudStack - Advance Network- Testbed, followed by the process of exposing the CloudStack Cloud Network Infrastructure. This is done through the DeepDive Methodology, which answered the second of these research questions. Finally, we listed the components of the Cloud tenants Virtual Network to be targeted via the following Penetration test as described in Chapter 5.

4.1 CloudStack Testbed

According to Sefraoui et al. [48], a preferred choice of cloud platform for developers was OpenStack and CloudStack was the second most preferred cloud platform. Despite successfully building and performing the OpenStack experiments, the results did not provide a clear conclusion about security on all clouds as the OpenStack tests were only one case study. Therefore, a second cloud was built, and the same testing procedures were applied to determine whether the IaaS Virtual Network Isolation security issues were generic or if they were applicable only to our original case study.

Sabharwal and Shankar [148] state that CloudStack is a solution or a platform for IT Infrastructure as a Service that allows a pooling of computing resources to build public, private and hybrid IaaS cloud services. These can be used to provide an IT infrastructure, such as Compute nodes (hosts), networks, and storage as a service to end users on demand.

CloudStack uses a similar range of technologies as OpenStack, however it adopts them differently. In the subsequent sections the process of CloudStack testbed construction is described. Additionally, the differences between building the CloudStack and OpenStack testbeds are highlighted. The CloudStack testbed was implemented based on the OpenStack testbed construction and scenarios. To justify that decision, we seek to compare the two testbed infrastructures in terms of Isolation mechanisms and vulnerabilities due to poor design and implementation. The two testbeds were built with as near as possible equivalent configurations of the testbed host OS, cloud network technologies and infrastructure, isolation mechanism, cloud nodes, tenants numbers and components, etc.

4.1.1 Requirements

As mentioned before, it was vital to create the CloudStack testbed as similar as possible to the OpenStack testbed, and hence the two testbeds have similar setups with some structural differences that were necessary for the CloudStack testbed. Those differences will be pointed out as we go through the testbed setup.

One of the major differences between the two testbeds was that although both provisioned the tenant with their own on-demand VN, each provided the networking service differently. OpenStack implements advanced networking using a designated module called Neutron to provide and manage the networking service. However, CloudStack uses the term “advanced zone” [148] to represent advanced networking configuration and provisioning.

As the aim was to test the cloud tenant VNs in both testbeds, a logical decision was to adopt similar tenant scenarios as in OpenStack. Figure 4.1 shows the implementation plan of the Test Tenant resources in CloudStack. As with OpenStack, the Test tenant is a legitimate, but rogue, tenant in the cloud, who attempts to compromise the other tenants VN isolation by launching the attack from a “KaliLight” VM. In this case, the Test tenant uses a private network “TestNT”

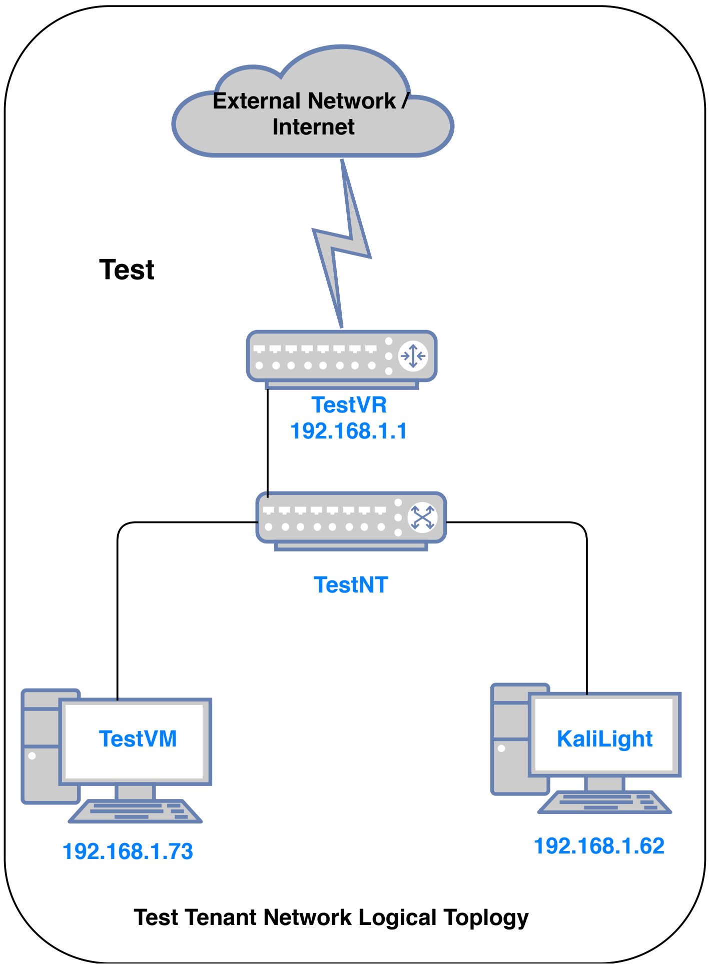


Figure 4.1: Infrastructure Plan of Test Tenant.

with the range 192.168.1.0/24 and the vRouter "TestVR" gateway IP address is 192.168.1.1. Moreover, the DHCP service for this network is configured within the vRouter itself. This is one of the differences in the scenarios between OpenStack and CloudStack. Once the vRouter in OpenStack is created with a DHCP option, a (hidden to user) component acts as a DHCP server is created in the infrastructure. Due to the nature of the CloudStack vRouter component implementation, it provides further networking functionality besides routing. The details of a vRouter in CloudStack is illustrated in detail later in §4.4.

The other tenant is Tenant1, which consists of two users each with their own private network, vRouter and VMs. The scenario, seen in Figure4.2, was designed to be equivalent to the Tenant1 in OpenStack. However, the CloudStack Tenant1 DHCP configuration is different from the OpenStack, and is similar to the CloudStack Test vRouter, which serves to route DHCP, DNS, etc. The next section provides detailed information about the implementation of the scenarios in

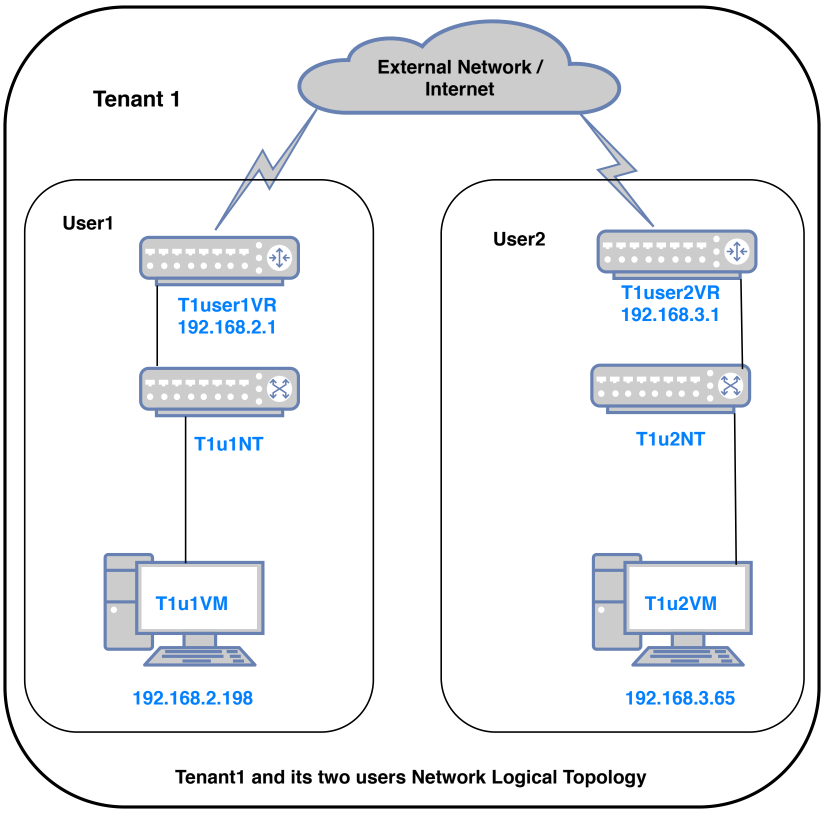


Figure 4.2: Infrastructure Plan for Tenant1 and its Users.

CloudStack.

The CloudStack Testbed setup took a similar approach to OpenStack, with a multi-node cloud architecture. As shown in Figure 4.3, the CloudStack testbed was built and customized based on several mixed resources following the CloudStack guideline [149] as well as other technical CloudStack community blogs and websites [150] [151] [152] [153] [154] [155]. The IaaS CloudStack testbed consists of two nodes; the Management Node and the KVM Host Node. As in OpenStack, we are interested in the two-nodes CloudStack inner infrastructure to uncover the cloud network infrastructure and exploit any VN vulnerabilities.

We built CloudStack in the same server but used a different virtualisation platform (VMware) and used the same two-node cloud Structure. We can still compare the testbed nodes as the CloudStack Management Node is equivalent to the OpenStack Controller Node and the CloudStack KVM Host Node is equivalent to the OpenStack Compute Node.

The next section describes the CloudStack testbed setup.

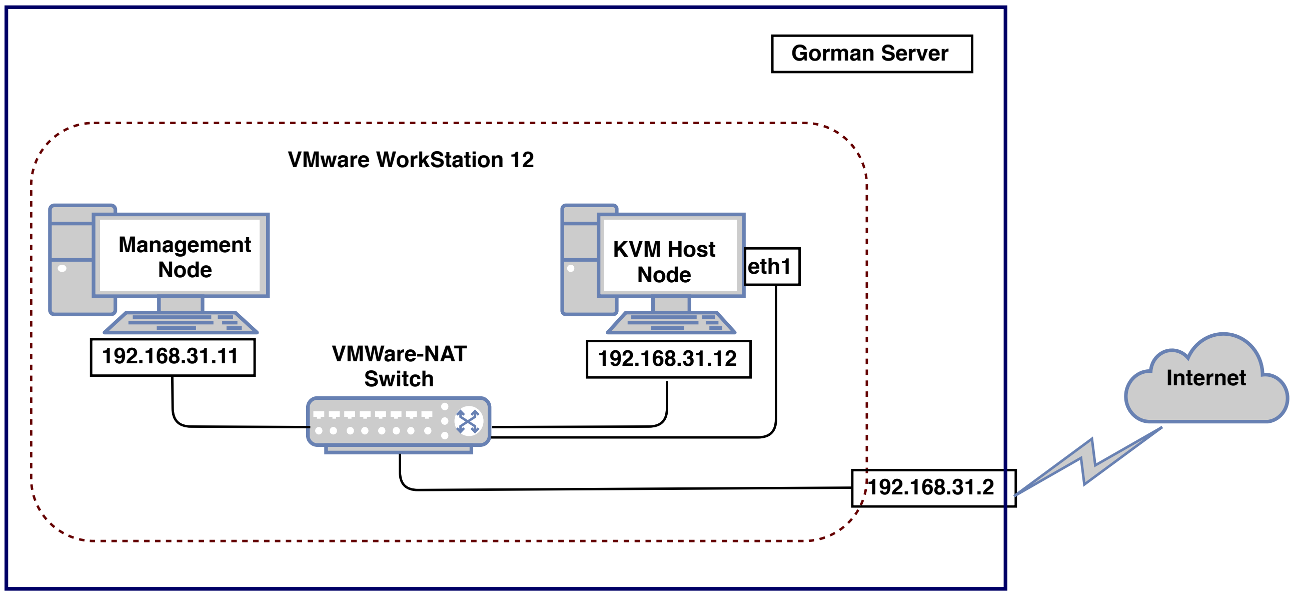


Figure 4.3: CloudStack Cloud 2-Nodes.

4.1.2 Testbed Setup

Based on lessons learnt from building a private IaaS cloud structure in Openstack, it took less, specifically four, attempts to build the CloudStack cloud. As with OpenStack, in order to study and investigate the CloudStack cloud we built up a pre-structure, which was an all-in-one basic node network cloud using the online resources [150] [151]. Some of the technical implications of this process are listed in §4.5.2.1.

4.1.2.1 The Preparation of the Cloud Environment

At the time of carrying out the experiments, most of the available CloudStack setup examples and guidelines focused on the use of one range of Network and partitioned IP addresses to fit the cloud structure components. However, one document [149] mentioned the possibilities of using different IP ranges for the Management connection and the Guest connection. The cloud platform setup in CloudStack presented even more challenges than in case of OpenStack. This resulted in performing a series of troubleshooting exercises, some of which are listed in §4.5. These led to multiple attempts to build a cloud and eventually to use one IP range for the whole cloud as instructed by the guidelines. Hence, the CloudStack Testbed differs from the OpenStack Testbed in the nodes network outer connectivity. We believe that this difference is not a great concern to our research and will not dramatically affect our security testing, as the aim is to investigate the Tenant VNs security that are located within the inner side of the cloud.

4.1.2.2 Cloud Network

As mentioned above, the CloudStack testbed was built on VMware using a single Network range. The testbed was configured with NAT for Internet connectivity, which was crucial for all CloudStack network types (Management, Public and Guest), as, in contrast with OpenStack, they failed to operate without it. The CloudStack VMWare-NAT range for all the Network ranges was 192.168.31.0/24. The different CloudStack Networks will be explained later in §4.2

4.1.2.3 CloudStack Nodes

Node	**Management**	**KVM Host**
RAM	4GB	10GB
CPU	2 cores	8 cores
Hard Disk	100 GB	100 GB
NIC/IP	eth0 (Management-NAT) = 192.168.31.11	eth0 (Management & Public-NAT)= 192.168.31.12 eth1(guest-NAT)=Manual
OS	Ubuntu-14.04.2-server-amd64.iso LTS	Ubuntu-14.04.2-server-amd64.iso LTS

Table 4.1: CloudStack Nodes Configuration Details.

Although our approach was to match the OpenStack Testbed, as mentioned before, several differences in the CloudStack cloud two-node structure were inevitable. We configured two nodes - Table 4.1- (Management and KVM host nodes) with Ubuntu 14.04 LTS and provided the hypervisor Node (KVM Host) with higher hardware resources (RAM and CPU). However, unlike in OpenStack, the CloudStack Network connection services are predominantly based in the KVM Host Node compared to the Management Node.

4.1.2.4 CloudStack Environment

Table A.4 in Appendix A shows the necessary packages configured and run in both CloudStack nodes. It is noted that the number of packages in CloudStack is fewer than those installed in OpenStack.

4.1.2.5 Network Configuration

Table A.5 in Appendix A shows the vital cloud Network configuration and it is clear the largest configuration is in the KVM Host, which means the Cloud Network infrastructure configuration focus is not in Management as we witnessed in the OpenStack Testbed.

We made sure to include OVS in the structure and as with the collection of resources, we implemented two OVS bridges: cloudbr0 and cloudbr1. Cloudbr0 was implemented for the

Management and Public connections and linked to eth0, while cloudbr1 was for the Guest network linked to the physical eth1. Additionally, the cloud infrastructure agent.properties file was configured to use OpenvSwitch(OVS) in its network infrastructure to access virtual switches, as well as the hypervisor to link any virtual interfaces for the newly created instance to the designated OVS switches.

At this point the main and preliminary blocks and necessary software for the cloud infrastructure have been setup. Yet, this is not a complete cloud infrastructure, as there are remaining structures such as Zones, PODs, Clusters, Hosts, Storages, Cloud Networking, tenants' accounts and tenants resources provisioning are yet to be configured in the cloud Management server console via the Admin account.

This is a major difference from OpenStack as the Management server console is tenant focused and all the cloud Infrastructure was configured completely beforehand.

CloudStack Advance Zone setup: CloudStack adopts a hierarchical design to its structure that consists of a pool of physical resources and is controlled from "a single management interface". The CloudStack Networking style and functionality is defined once the cloud administrator has configured the cloud Zone to be Basic or Advanced[148]. Table 4.2 presents the detailed configurations of our CloudStack Testbed deployment module.

4.2 CloudStack Terminologies

The CloudStack platform is designed to manage the physical infrastructure by dividing the resources logically into different layers. These layered resources are labeled Zones, Pods and Clusters. Each layer consists of even further resources that are vital to cloud operation such as Hosts, Primary and Secondary Storages. Figure 4.4 shows how CloudStack applies its " logical partitions to form the cloud deployment model [148].

Zone: As the CloudStack deployment model logically partitions the infrastructure in a hierarchical fashion, a Zone is considered to be the " highest level" of the cloud hierarchy as shown in Figure4.4. Depending on the cloud deployment a Zone can represent a single Data Centre or can be "geographically distributed" [148]. Our design follows the former option. The main components of a Zone are a minimum of one POD (see later) and secondary storage.

Our testbed selected the option of Advanced Zone, to match the level of the OpenStack testbed. The Advanced Zone provided the cloud network with a layer 3 subnet, using VLAN, STT and GRE for "security and isolation" and other network services like NAT and VPN [148].

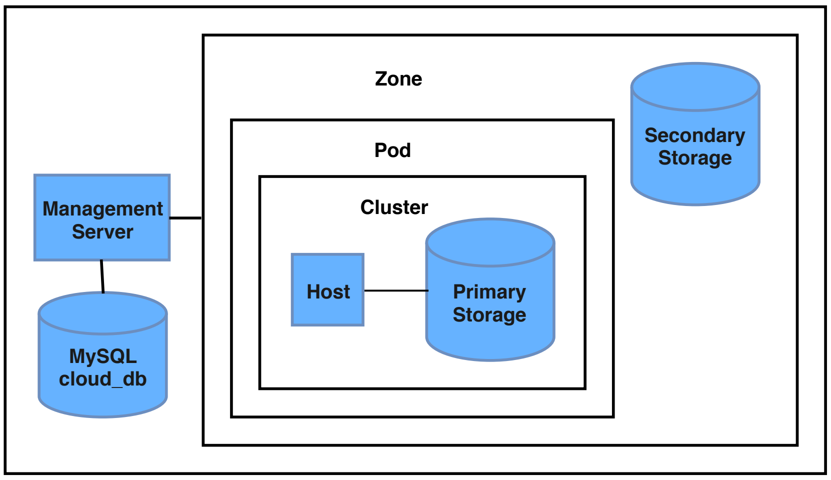


Figure 4.4: CloudStack Deployment Model [148].

Table 4.2 on page 118 displays the configuration details used to build our testbed, such as internal and external DNS value(s) and the Hypervisor used.

CloudStack Networking: As mentioned above selecting the Zone type provides different configuration parameters, which offers different levels of networking provisions, and therefore the Advanced Zone provided advanced network features. As shown in Table 4.2, the Advanced Zone network configuration consists of two physical networks located in the Host Node, and each is related to defined network traffic and connected to a specific OVS bridge built in the Host Node.

The Advanced Zone is the most suitable option for a complicated network structure that supports "multiple physical networks", guest networks with VLAN isolation and advanced "network services such as Firewall, VPN and load balancers".It also supports four types of network traffic: Guest, Management, Storage and Public[148]. As shown in Table 4.2, which matches the configuration in Table A.5 for the first physical network, Host Node eth0 is connected to the Cloudbr0 bridge and handles management and public network traffic. Meanwhile, the other physical network, Host Node eth1, is connected to cloudbr1 and is reserved for the guest network traffic. Moreover, extra configuration is required for the public traffic network such as the IP address range to be used, gateway, netmask and guest traffic VLAN ID range (see Table4.2).We also had to change the configuration of the tenant network isolation type from VXLAN, as we did in OpenStack, into GRE. This was because the OVS driver of the hypervisor VXLAN was not supported and hence the nearest alternative was to user GRE. The details of this issue are discussed in §4.5.2.2.

POD: A Zone is divided logically into smaller portions called PODs, which the Cloud community describes as a physical rack in the Data Centre. The main aim of using pods is to host the hypervisor hosts and the management subnet. The CloudStack Management node uses them to communicate with special cloud components called “system VMs”. Additionally, PODs host hypervisor hosts contained within clusters [148], which is the next hierarchical level in the CloudStack model, as previously illustrated in Figure 4.4.

Cluster: A POD contains a minimum of one cluster, which collects a set of the same hypervisor hosts [148]. This Testbed has one cluster and hosts one KVM hypervisor host.

Host: CloudStack supports different hypervisors such as XenServer, VMware and KVM [148]. One hypervisor server is a host. Table 4.2 shows that we defined a KVM host by its IP address and gave it the credentials of the Zone for operation authentication.

Primary Storage: Primary storage is built at the level of a Cluster, a minimum one primary storage per cluster. Primary storage is a vital component to host the tenant’s instance (VMs) and its virtual disk specification and allocation, hence the primary storage must be compatible with the hypervisor implemented in that cluster. CloudStack supports the primary storage through technologies such as iSCSI, NFS, VMFS or EXT file systems [148].In our testbed we configured the primary storage using NFS, which is hypervisor independent, and it is managed and provisioned by the CloudStack management, however it is used as the mount point by a KVM hypervisor [148]. Table 4.2 shows the configuration details of the Testbed Primary Storage.

Secondary Storage: Secondary storage, contrary to primary storage, has only NFS as a sole choice of configuration. This is mainly used to store ISO images or templates to create tenant instances (VMs). This is built at the Zone level, therefore, PODs share this component and make it accessible to all hosts [148]. As shown in Table 4.2 both primary and secondary storage are physically implemented in the Management Node.

4.3 Preparing the Testing Scenario

We used the OpenStack testbed as a reference in order to create a test scenario for CloudStack, and applied the same testing methodologies (DeepDive and Penetration testing). The scenario consists of tenants, each with their VN, VN components and configuration VMs. The scenarios are configured differently in both testbeds, as described later, but their outcomes are similar.

Tenants and Users:

As our aim is to achieve a scenario as close to OpenStack as possible, we created the same tenants and users in the CloudStack testbed as shown in Table 4.3.

Cloud Level	**Parameters**	**Configuration Value**
Zone	Advance Zone	Name: zone1 IPv4 DNS1 & Internal DNS1: 192.168.31.2, Hypervisor: KVM
Setup Network: KVM Host	Physical network 1	Management: KVM Trafic lable= cloudbr0, public: KVM Trafic lable= cloudbr0, Isolation method=VLAN
	Physical network 2	guest: KVM Trafic lable= cloudbr1, Isolation method=GRE

	Public Traffic	Gateway= 192.168.31.2,
		Netmask= 255.255.255.0,
		Start:192.168.31.100,
		End: 192.168.31.120

	Guest Traffic:	Physical Network 2: VLAN/VNI Rang: 10 - 20
POD:	Pod name=pod1	Reserved system gateway= 192.168.31.2, Reserved system netmask= 255.255.255.0, Start Reserved system=192.168.31.121, End Reserved system = 192.168.31.140
Cluster:	Cluster name=cluster1	hypervisor=KVM,
Host:	Host Name= 192.168.31.12,	usernamr=root, password=cloudstack

Storage	Primarry Storage:	Name=primary ,
		Scope=Zone-Wide ,
		Protocole=nfs ,
		Server=192.168.31.11 ,
		Path=/export/primary

	Secondry Storage:	Provider=NFS,
		Name=Secondary ,
		Server=192.168.31.11 ,
		Path= /export/secondary

Table 4.2: CloudStack Advance Zone Structure Setup.

Accounts	**Users**
Admin	Admin
Test	Test
Tenant1	T1user1 T1user2

Table 4.3: CloudStack Tenants(accounts) and Users.

Apache CloudStack has its own in-house style to define tenants and users. CloudStack manages three levels and they are: Accounts, Users and Domains. An account defines the cloud customer (Tenant), which is isolated from other accounts and can contain multiple users. Domains are used to group cloud accounts logically. Users are sometime called “aliases”. All users in one account are not isolated, however each user is assigned credentials to utilize the cloud resources and service [156].

In this research we use the term Tenant to refer to the CloudStack account, while the term user is used to identify the CloudStack user.

4.3.1 Network and Routers

This section describes the implementation of the scenarios shown in Figures 4.1 and 4.2. After creating the tenants, the VN and vRouters are configured as described below.

CloudStack Network:

As mentioned in §4.1.2.5, the CloudStack Admin offers two network methodologies, Basic and Advanced, which can be selected during the Zone creation. Additionally, tenants can select from various types of network services, based on the networking offering configured by the cloud Admin[148].

Maintaining multi-tenancy in the cloud solution via isolation is a crucial requirement, hence, CloudStack allows the Admin user to define different levels of network topologies to serve the cloud tenant’s complicated application demands. Therefore tenant networks are logically contracted divisions of the physical network that CloudStack uses to store their configurations and settings, activated on launching the first VM, and isolated logically through VLANs [148].

The construction of the network in CloudStack is comprised of different resources, configuration and services, such as network providers, network service and network offers. Network providers consist of network elements, hardware and virtual appliances and they assist in creating the network services. A collection of network services provides the network offers that tenants use to create their own virtual networks [148]. Table 4.4 contains our experimental tenants and their network configuration scenarios including the network offering. We selected one of the default provided offerings to create the Virtual network.

CloudStack vRouter:

CloudStack created their on-demand virtual network using a CloudStack virtual router, which delivers multiple network services, such as DHCP, DNS, and NAT amongst others. A tenant VM

Network Details	**Test**	**T1user1**	**T1user2**
Network Name	TestNT	T1u1NT	T1u2NT
Router	TestVR	T1user1VR	T1user2VR
Network type	Guest Network (isolated)	Guest Network (isolated)	Guest Network (isolated)
Guest Network IP	192.168.1.0/24	192.168.2.0/24	192.168.3.0/24
Provider	VpcVirtualRouter	VpcVirtualRouter	VpcVirtualRouter
VPC Offering	Default VPC offering	Default VPC offering	Default VPC offering
Network offering	Default Isolated Network Offering For Vpc Netowrks	Default Isolated Network Offering For Vpc Netowrks	Default Isolated Network Offering For Vpc Netowrks
Gateway	192.168.1.1	192.168.2.1	192.168.3.1
Netmask	255.255.255.0	255.255.255.0	255.255.255.0
ACL	default-allow	default-allow	default-allow

Table 4.4: CloudStack Tenants Network Components and Configuration.

connected to a private guest network can reach the outside world via the virtual router assigned to that guest network [148]. Table 4.4 shows the Virtual Router Configuration (VPC) and the VPC offering defined in §4.4.2.2 for our scenarios.

Differences between OpenStack and CloudStack configuration scenarios:

- OpenStack security group polices are defined at VM level, while CloudStack defines them at the router level.

- The OpenStack DHCP option should be defined as an option when creating the network, while CloudStack uses a different approach where both network and vRouter are called offerings. CloudStack provisions the network scheme and network elements as resources. The offering choices selected in the table include the DHCP option. Therefore, while OpenStack creates a network and uses the DHCP option selected to create a network element separate from a vRouter to represent a DHCP server, this is hidden from the tenant.

- We accepted the differences in the technologies implementation as essential and inevitable implementation differences. However we focused on using similar networking technology options. Although we anticipated the differences between the two cloud infrastructures designs, we settled with fully functional cloud systems that provide similar services.

4.3.2 Instances

In order to create tenant instances, server parameters must be available, as listed in Table 4.5; the Compute offering, Template Featured, Disk offering and the guest network to connect the VM.

Instance Details		**Test**	**T1user1**	**T1user2**
Name	KaliLightVM	TestVM1	t1u1vm	t1u2vm
Compute Offering	Medium Instance	Small Instance	Small Instance	Small Instance
Template Featured	KaliLight2v1	CentOS 5.5(64-bit) no GUI(KVM)	CentOS 5.5(64-bit) no GUI(KVM)	CentOS 5.5(64-bit) no GUI(KVM)
Disk Offering		Small [5 GB]	Small [5 GB]	Small [5 GB]
net-id	TestNT	TestNT	T1u1NT	T1u2NT
IP address	192.168.1.62	192.168.1.73	192.168.2. 198	192.168.3.65

Table 4.5: CloudStack Tenants Instances and Their Configuration Details.

The Compute offering is defined by the Admin and allocates resource capacity for the created VM. Resources are the CPU processing speed, memory size and network rate, etc [148]. Table 4.5 shows two types of Compute offerings that were selected for the VM scenarios, which are (1) a Small instance, and (2) a Medium Instance.

A disk offering represents storage resources from primary storage that the cloud provisions to the tenant VMs as per their request[148]. We selected one option of the disk offering and that is the Small [5GB] option, as seen in Table 4.5.

Out of the two types of Template features offered by CloudStack, we used a pre-existing OS image template[148] to install the VM operating systems, see Table 4.5. The installed operating systems include CentOS, which is a default image provided by the cloud platform for testing, and KaliLight2v1 which we created for the Data Leakage penetration testing.

The Differences between CloudStack and OpenStack configuration scenarios are as follows:

- The CloudStack Compute offering & Disc Offering is called a flavor in OpenStack.
- The CloudStack Template Featured is called an image in OpenStack.

4.4 CloudStack Infrastructure DeepDive

A DeepDive is a method to reveal the underlying Cloud infrastructure through a white box testing technique and results in an infrastructure map. The collected information from this experiment is analysed and mapped in Figure4.7.

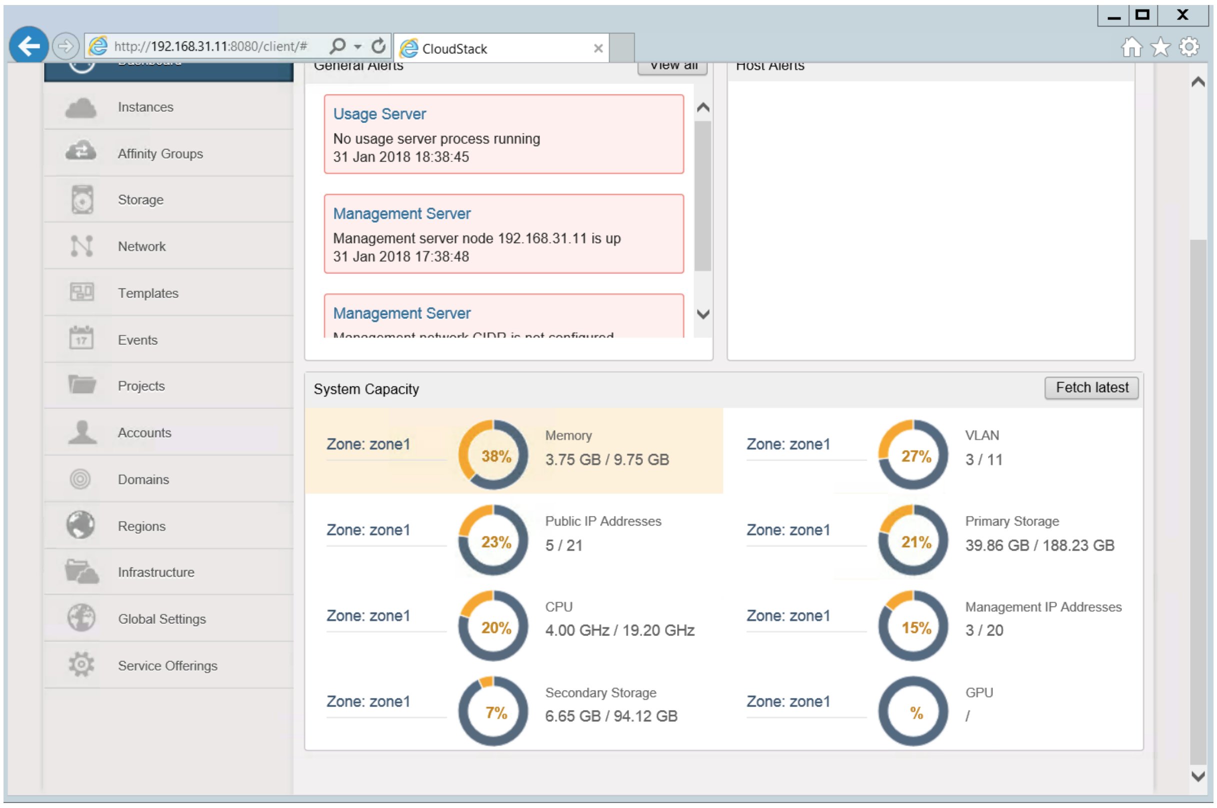
Sources of Information:

We used the same DeepDive approach as in the case of the OpenStack Testbed. Additionally, we planned to use both the sources and procedure(s) to collect the information from the Management Server Console and scripts. Although the CloudStack platform used an all-in-one pre-testbed, we found that unlike OpenStack, CloudStack does not have a common command-line interface (CLI) to its APIs. Therefore, we relied on OpenStack commands to build the scripts to collect the information directly from the OpenStack database. Viewing, modifying, deleting and adding cloud structure configuration was done from the Management server console. Hence, as an option, we installed an external CLI tool built in python for CloudStack, called cloudmonkey [1]. However, unlike OpenStack, cloudmonkey did not give us any additional information that we obtained from the Management server console. Hence, we limited our sources to the Management server console and reduced the number of scripts. After dropping the option to collect from the cloud database, we used only two script files for the cloud nodes. Again we used the DeepDive methodology to map the tenant network components and how and where they fit together in the cloud network infrastructure. We explored the isolation mechanism used and defined our target to test our hypothesis on isolation security and data leakage through analysing the information collected from:

- The information used from the cloud setup and testbed building.
- The Management Server Console: we collected the information from within the Infrastructure tab as well as the Instance tab, the Network tab and the account table for the tenants and users as shown in Figure 4.5.
- The two script files: the Management Node file (cldtkmgmtscript.sh) and the KVM Host Node file (cldstkkvmscriptV2.sh). Both scripts contained command lines that produce outputs of the physical (e.g NICs) and logical (e.g OVS) components, configuration and connections.

Terminologies between the two clouds may differ but the functionality is the same, for example in OpenStack a tenant is represented with the terminology project while CloudStack uses the term account. Moreover, the privacy and multi-tenancy is applied on the tenant level (OpenStack project, CloudStack account) and yet any cloud user belonging to the same tenants has no privacy, as is shown by Network tab in Figure 4.6.

[1]https://cwiki.apache.org/confluence/display/CLOUDSTACK/CloudStack+cloudmonkey+CLI#CloudStackcloudmonkeyCLI-Whatiscloudmonkey



Usage Server
No usage server process running
31 Jan 2018 18:38:45

Management Server
Management server node 192.168.31.11 is up
31 Jan 2018 17:38:48

Management Server

System Capacity

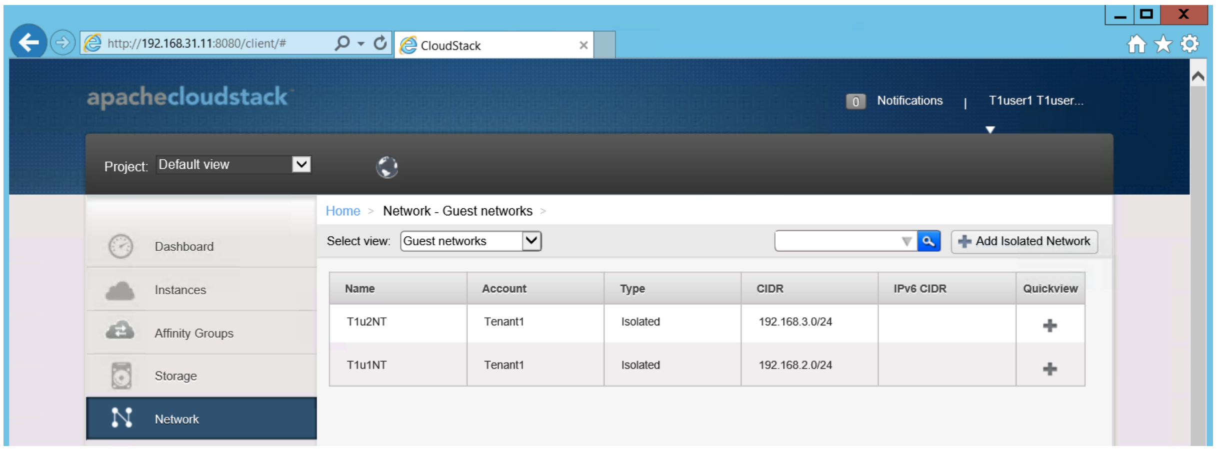
Zone: zone1	38%	Memory 3.75 GB / 9.75 GB	Zone: zone1	27%	VLAN 3 / 11
Zone: zone1	23%	Public IP Addresses 5 / 21	Zone: zone1	21%	Primary Storage 39.86 GB / 188.23 GB
Zone: zone1	20%	CPU 4.00 GHz / 19.20 GHz	Zone: zone1	15%	Management IP Addresses 3 / 20
Zone: zone1	7%	Secondary Storage 6.65 GB / 94.12 GB	Zone: zone1	%	GPU /

Figure 4.5: CloudStack Management Server Console.

4.4.1 Cloud Network Infrastructure Map

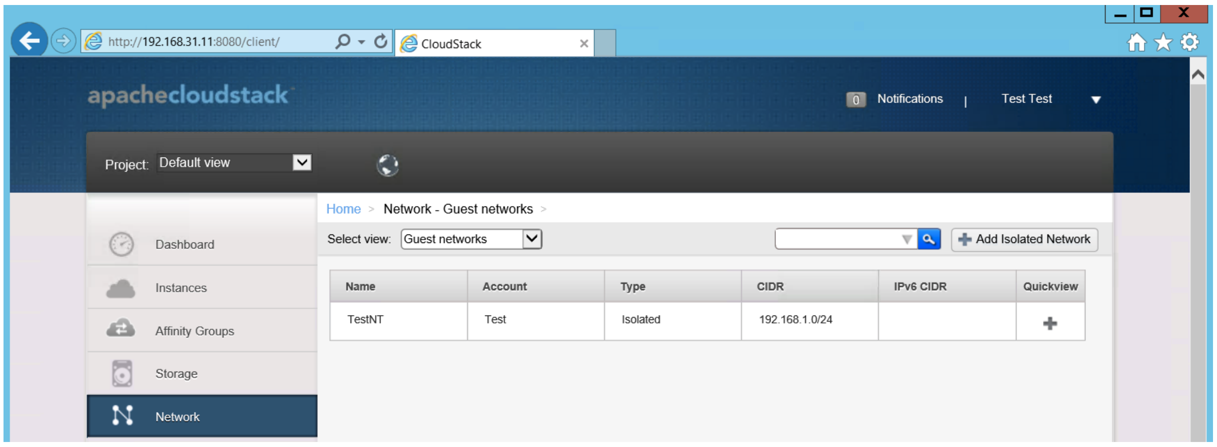
ThebDeepDive methodology has been used in industry in system work to troubleshoot a non-standard processes. We have used a DeepDive and the outcome is a visual representation of the cloud inner infrastructure. In this DeepDive version we created our own process for a different purpose; after studying the cloud platforms we devised our own white box testing version to collect platform information, analyse collected data and map them in a diagram for better understanding and identification of the cloud VN components and isolation mechanisms. This information can also be used as a reference guide to evaluate the level of leaked data in the penetration testing methodology discussed in Chapter5

Applying the DeepDive methodology was proven to be much easier in CloudStack as we had gained the advantage of our experience from applying it in OpenStack. However, analysis and mapping components and labels into a diagram required a significant amount of effort due to a the large data analysis process. The starting point of the DeepDive was from expanding Figure 4.3 and zooming inside it to place each cloud infrastructure component into its correct place and drawing the connections as per the analysed data. Therefore we used the outer layer of Figure 4.3 and enlarged it to include the tenants, users, VMs and VN components such as OVS



Name	Account	Type	CIDR	IPv6 CIDR	Quickview
T1u2NT	Tenant1	Isolated	192.168.3.0/24		+
T1u1NT	Tenant1	Isolated	192.168.2.0/24		+

(a) Tenant1 User1 and User2 Networks



Name	Account	Type	CIDR	IPv6 CIDR	Quickview
TestNT	Test	Isolated	192.168.1.0/24		+

(b) Test Tenant Network

Figure 4.6: Tenants and Users Network Privacy.

bridges, ports, connections, IPs and VLANs as shown in Figure 4.7. The next sections describe the analysis approach that was carried out to match each component with the DeepDive diagram.

4.4.2 DeepDive Analysis

While the approach taken in OpenStack was to use script files, in CloudStack we depended heavily on the Management server console accessed with the Admin account to collect the structure information. The remaining information collection was determined via scripts. The scripts showed that the Management Node provides no further information, besides the Node interface and its IP address, that could be used for the infrastructure. Therefore, we concluded that the entire Cloud Networking infrastructure is resident in the KVM Host Node. We analysed the collected information from the Management server console and the script files into tables for better understanding as displayed below.

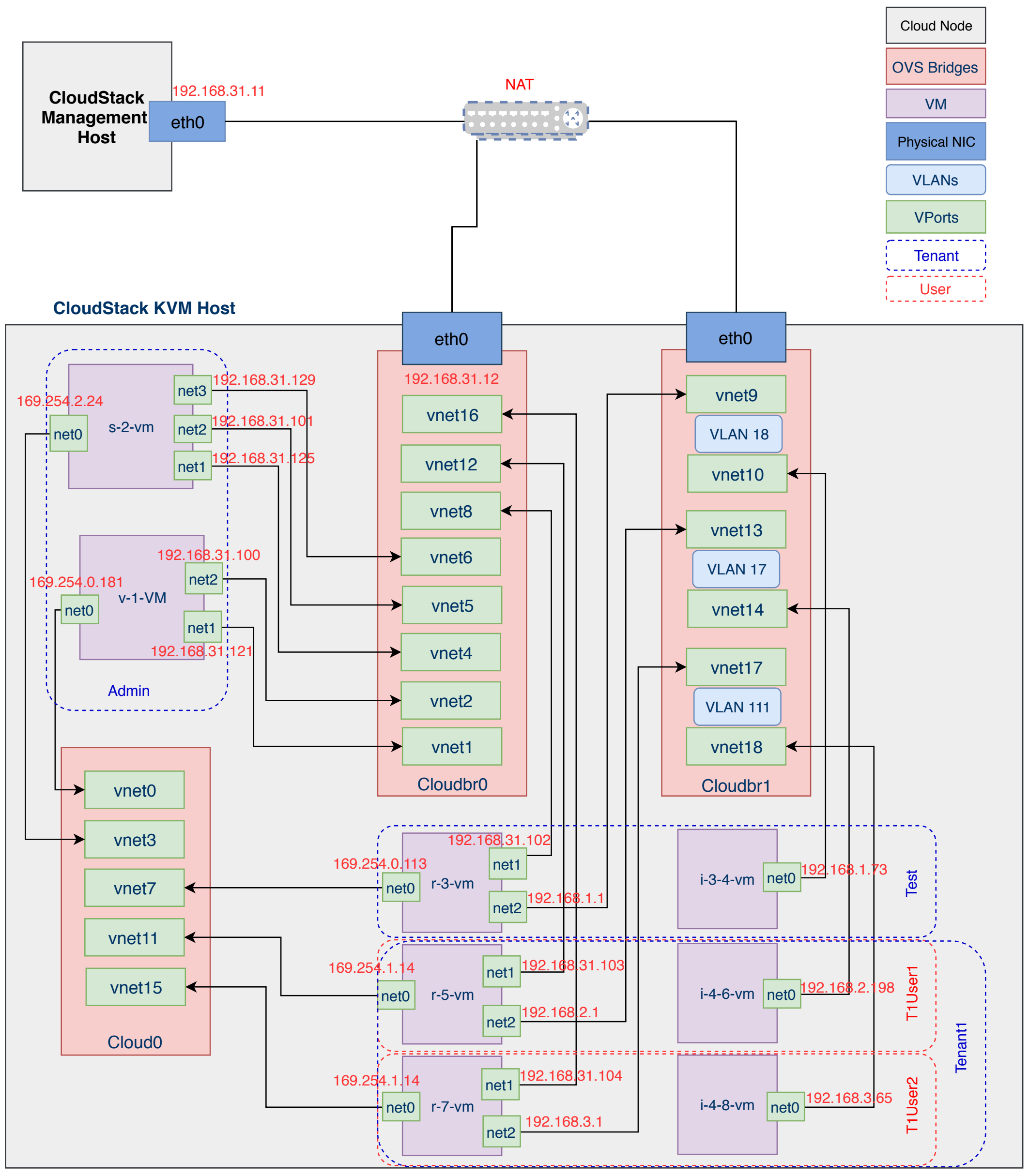


Figure 4.7: CloudStack Network Infrastructure DeepDive .

This section presents the analysis of CloudStack Testbed collected information 81and highlights the findings to match them with the DeepDive diagram illustrated in Figure 4.7. Table4.5 lists the Kali VM, however, it is excluded from Figure 4.7 and from the DeepDive analysis, as the DeepDive methodology was executed prior to creating the KaliLight instance.

We did not need the table of tenants and their identities as CloudStack uses the tenant name rather than the IDs. We then used Table 4.5 as a reference. In CloudStack the ownership is represented with by the tenant name, listed originally in Table 4.3 as will be shown in the following tables by

4.11, 4.12, 4.13 and 4.15.

4.4.2.1 KVM Host Node

KVM Host Physical ports

From the cldstkkvmOutput file that is produced by the cldstkkvmscriptV2.sh script file the result of the two commands: *ifconfig* and *ip a* was a list of 27 ports, their physical addresses and IP addresses of some ports in the KVM host node. This is shown in Table A.6 in Appendix A. The list contains:

- The physical node interfaces: eth0 and eth1,
- The OVS-system which indicate that the system uses OVS technology,
- The OVS bridges: cloudbr0, cloudbr1 and cloud0. We created cloudbr0 and cloudbr1 while cloud0 was automatically created by the cloud system,
- The libvirt bridge (virbr0) is automatically created by the hypervisor and had nothing to do with the cloud connections.
- The 19 virtual network interfaces listed from vnet0 up to vnet 18.

Up to this point we had collected a great deal of information of the cloud infrastructure, however, we did have enough information to locate and place each piece in the infrastructure.

OVS:

There is no trace of OVS bridges or ports in the Management Node as all are resident in the Kernel-based Virtual Machine (KVM) Host Node.

The result from the script file matches the configuration in the Table A.5 where the OVS bridges are created, and physical ports are added. However, an additional bridge (cloud0) was not part of the configuration, which means that this bridge was created by the cloud platform after the Zone was created.

This section consists of an OVS command list table, OVS bridges and the interface details table as well as the placing of the information into Figure 4.7. Table 4.6 consists of the OVS commands used in the KVM Host Node, the type of information collected and the command output description. Meanwhile Table 4.7 contains the output after running the commands listed in Table 4.6.

Command	Gathered information	Description
sudo ovs-vsctl list-br:.	OVS bridges	lists all OVS bridges created in the Node
sudo ovs-vsctl list-ifaces cloudbr0:	Interfaces	lists all the interfaces listed in cloudbr0bridge.
sudo ovs-vsctl list-ifaces cloudbr1:	Interfaces	lists all the interfaces listed in cloudbr1 bridge.
sudo ovs-vsctl list-ifaces cloud0:	Interfaces	lists all the interfaces listed in cloud0bridge
sudo ovs-vsctl show:	OVS bridges, Interfaces, VLAN Tag, patched interfaces	shows all the OVS bridges and their associated interfaces, Tags, patches and extra information.
sudo ovs-ofctl show cloudbr0:	Interfaces #, Name, MAC	lists cloudbr0 interfaces, listed #, MAC
sudo ovs-ofctl show cloudbr1:	Interfaces #, Name, MAC	lists cloudbr1 interfaces, listed #, MAC
sudo ovs-ofctl show cloud0:	Interfaces #, Name, MAC	lists cloud0 interfaces, listed #, MAC

Table 4.6: KVM Host Node OVS Command List.

Both Tables 4.6 and 4.7 list specific OVS bridges; cloudbr0, cloudbr1 and cloud0. The same bridges are also shown in Table 4.4 for the Advanced Zone network setup where cloubbr0 was specified for management and public traffic while cloudbr1 was specified for guest traffic. At this point we have no knowledge of the Cloud0 bridge and what it is used for.

Cloud0 is a host only bridge, where it gets created as soon as a hypervisor host is added to the cloud zone. Cloudbr0 is sometimes called an internal bridge, which has no configuration as the CloudStack manages it [154]. The system bridge (cloud0) is created by the CloudStack-agent package, which configured it, in this case, with “local-link network 169.254.0.0/16”. It was assigned to handle the management traffic of this network untagged VLAN [157].

Table 4.7shows the interfaces of the three bridges, their listing numbers and MAC addresses. The following points are a summary of our observations based on the information collected in Table 4.7:

- Each bridge has a different number of ports: cloudbr0 consists of 10 ports, cloudbr1 contains 8 ports and cloud0 has only 6 ports.
- Only two bridges are connected to the physical ports, cloudbr0 has eth0 and cloudbr1 has eth1.
- Cloudbr1 is the only bridge with a tagged port, which means this bridge contains the values of the VLANs and there are three VLAN numbers, 11,17 and 18.
- Mapping this information to Figure 4.7 applies to the OVS boxes and their ports as well as

Bridge			Cloudbr0		
	Port	**Tag**	**Type**	**Listed #**	**MAC**
	eth0			1	00:0c:29:72:d6:cf
	vnet1			2	fe:3d:d6:00:00:01
	vnet12			8	fe:2f:60:00:00:18
	vnet16			9	fe:59:68:00:00:19
	vnet2			3	fe:e9:a6:00:00:15
	vnet4			4	fe:d4:3a:00:00:05
	vnet5			5	fe:ee:34:00:00:16
	vnet6			6	fe:88:18:00:00:09
	vnet8			7	fe:00:ca:00:00:17
	cloudbr0		Internal	Local	b6:00:07:e2:7f:a7
Bridge			cloudbr1		
	Port	**Tag**	**Type**	**Listed #**	**MAC**
	eth1			1	00:0c:29:72:d6:d9
	vnet10	18		3	fe:00:1c:0f:00:01
	vnet13	17		4	fe:00:46:c8:00:02
	vnet14	17		5	fe:00:12:bc:00:01
	vnet17	11		6	fe:00:63:79:00:02
	vnet18	11		7	fe:00:7f:e8:00:01
	vnet9	18		2	fe:00:52:86:00:02
	cloudbr1		internal	Local	00:0c:29:72:d6:d9
Bridge			cloud0		
	Port	**Tag**	**Type**	**Listed #**	**MAC**
	vnet0			1	fe:00:a9:fe:00:b5
	vnet11			4	fe:00:a9:fe:01:0e
	vnet15			5	fe:00:a9:fe:01:e9
	vnet3			2	fe:00:a9:fe:02:18
	vnet7			3	00:a9:fe:00:71
	cloud0		internal	Local	6e:19:aa:d7:57:4c

Table 4.7: KVM Host Node OVS Details.

the VLAN numbers in cloudbr1 attached to their associated ports.

- At this point, other than the physical ports (eth0 and eth1) we do not have any knowledge about which port, in each bridge, is connected to or belonged to which tenant elements.

One of our observations is that there is no trace of the GRE information from the collected data either from the general OVS commands or the OVS table flow rules commands, despite the fact that we configured it in Table 4.2 and expected it to be assigned in the cloudbr1 bridge. However, after further investigation and comparing it with OpenStack, we found that GRE is only used to connect the tenant components distributed between the cloud Nodes. In other words, because OpenStack distributed the tenant network components between the Compute and Controller nodes the connection was isolated using VXLAN. However, as we used a similar number of nodes in CloudStack (one management node and one hypervisor node) and all the tenants network components were located in the host node, there was no reason to configure the GRE in the designated bridge. However, we are confident that such a configuration would be applied once another host is added to the CloudStack testbed.

Libvirt XML files

When we tried to extract the XML files that contain the VM configuration from the libvirt path as we did in OpenStack, we discovered they were nowhere to be found. We used the virsh tool, which is installed by default with libvirt within the CloudStack-agent package. The technical details of this issue are listed in §4.5.2.2.

In order to achieve the information collection of the instances we used different commands from the scripts files of the OpenStack DeepDive :

- Firstly, we used *virsh list –all* and the output exceeded our expectation as we found eight VMs listed when we expected only three VMs.

- Secondly we used *Virsh dumpxml <VMName> > vmname.xml* to collect the listed VM XML files content. A closer look at each XML file gave us a hint about three different VM types created in the cloud from the VM name and the number of connection interfaces. Therefore we analysed each type into separate tables as shown in Tables 4.8, 4.9 and 4.10.

Compared to OpenStack, we found that allocating and connecting the VMs into the infrastructure is much easier as we only have three OVS bridges and all the VM interfaces are directly connected to them. Moreover, this version of the XML files contains information about the VNC service; graphical type information, designated ports and such information as is useful. However, the XML files gave no indication of the owner of the VM, opposite to OpenStack, and therefore we still have no information as to which tenant each VM belongs to.

Instance Name	*v-1-VM*			*s-2-VM*			
ID	2			3			
Interface Type	bridge			bridge			
MAC	0e:00:a9:fe:00:b5	06:3d:d6:00:00:01	06:e9:a6:00:00:15	0e:00:a9:fe:02:18	06:d4:3a:00:00:05	06:ee:34:00:00:16	06:88:18:00:00:09
Alias Name	Net0	Net1	Net2	Net0	Net1	Net2	Net3
Source Bridge	cloud0	cloudbr0	cloudbr0	cloud0	cloudbr0	cloudbr0	cloudbr0
Target Dev	vnet0	vnet1	vnet2	vnet3	vnet4	vnet5	vnet6
Graphics Type	VNC, port='5900', listen='192.168.31.12'			VNC, port='5901', listen='192.168.31.12'			

Table 4.8: v-1-VM and s-2-VM XML Information Details.

Table 4.8 shows that v-1-VM consists of three interfaces, one connected to a port in Cloud0 OVS bridge and the other two ports connected to interfaces belonging to the cloudbr0 OVS bridge. The information about the bridges and their associated ports "Target dev" matched the information from Table 4.8. At this point we mapped those VMs to Figure 4.7 without the IP address of their owner. Similarly, the same strategy was applied to s-2-VM and the only difference is that it consists of four interfaces, three of which are connected to cloudbr0.

Instance Name	*r-3-VM*			*r-5-VM*			*r-7-VM*		
ID	4			6			8		
Interface Type	bridge			bridge			bridge		
MAC	0e:00:a9:fe:00:71	06:00:ca:00:00:17	02:00:52:86:00:02	0e:00:a9:fe:01:0e	06:2f:60:00:00:18	02:00:46:c8:00:02	0e:00:a9:fe:01:e9	06:59:68:00:00:19	02:00:63:79:00:02
Alias Name	Net0	Net1	Net2	Net0	Net1	Net2	Net0	Net1	Net2
Source Bridge	cloud0	cloudbr0	cloudbr1	cloud0	cloudbr0	cloudbr1	cloud0	cloudbr0	cloudbr1
VLAN tag			18			17			11
Target Dev	Vnet7	Vnet8	Vnet9	Vnet11	Vnet12	Vnet13	Vnet15	Vnet16	Vnet17
Graphics Type	VNC, port='5902', listen='192.168.31.12'			VNC, port='5904', listen='192.168.31.12'			VNC, port='5906', listen='192.168.31.12'		

Table 4.9: r-3-VM, r-5-VM and r-7-VM XML Information Details.

Table 4.9 represents a similar type of information and we used a similar approach to translate that information into Figure 4.7. The difference is that each VM consists of three interfaces, with each one connected to a port in one of the three OVS bridges. Another difference is that the interfaces connected to the Cloudbr1 OVS bridge are tagged with a VLAN number which matched one represented in Table 4.7 above. This makes sense, as we already know from the cloud setup that cloudbr1 is reserved for guest traffic, which means it is used to connect VLAN isolated tenant networks.

Instance Name	*i-3-4-VM*	*i-4-6-VM*	*i-4-8-VM*
ID	5	7	9
Interface Type	bridge	bridge	bridge
MAC	02:00:1c:0f:00:01	02:00:12:bc:00:01	02:00:7f:e8:00:01
Source Bridge	Cloudbr1	Cloudbr1	Cloudbr1
VLAN tag	18	17	11
Target Dev	Vnet10	Vnet14	Vnet18
Graphics Type	VNC, port='5903', listen='192.168.31.12'	VNC, port='5905', listen='192.168.31.12'	VNC, port='5907', listen='192.168.31.12'

Table 4.10: i-3-4-VM, i-4-6-VM and i-4-8-VM XML Information Details.

Table 4.10 is the last table with VM information, as each of the three VMs contain only one interface and each is connected to one port in cloudbr1 with a matching VLAN tag. Based on the given information in Table 4.10 we assumed that the VMs in this table represent the tenant instances as each contains only one VLAN isolated interface and is connected to the OVS bridge that handles the guest private networks.

VMs from the three Tables 4.8, 4.9 and 4.10 are plotted in Figure 4.7 with their names, interface alias name and connection to the OVS bridges.

4.4.2.2 Management Server Console Collected Information

After investigating the management server console, we found that the Admin account was the one that collected the infrastructure and tenant information and their resources. We logged in to the CloudStack website using the URL http://192.168.31.11:8080/client and the Admin credentials. Note that the user interface is called a Dashboard in OpenStack but in CloudStack the interface is a console with several tabs as shown in Figure 4.8.

In this section we collected and analysed information from the console including the network tab, the instance tab and the infrastructure tab, the latter being exclusively used by the Admin account.

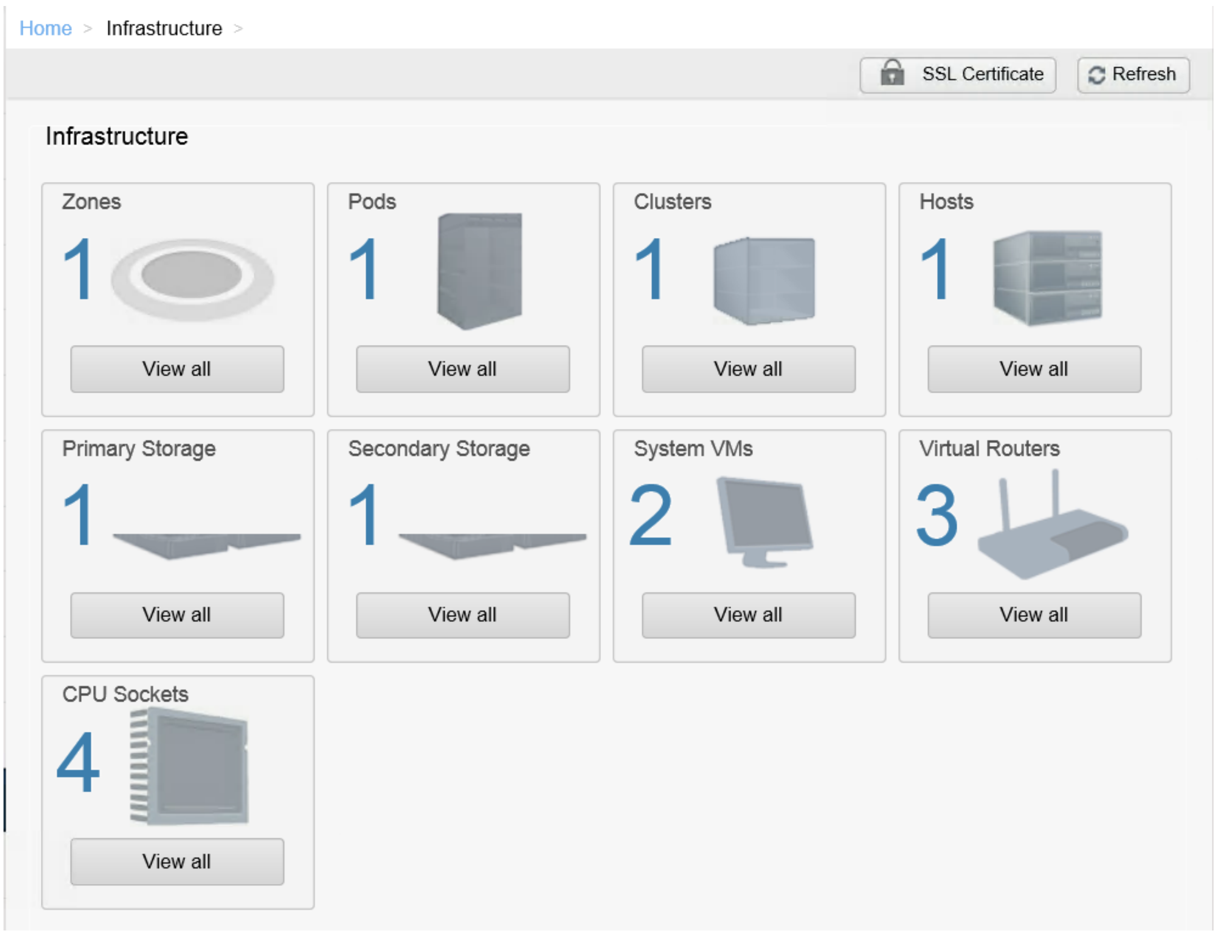


Figure 4.8: CloudStack Management Server Console – Infrastructure.

Network Tab: The Network tab is where the Guest Network and Network VPC (vRouter) are created and managed.

Guest Networks: Table 4.4 contained the configuration details of the tenant networks and vRouters when we planned the tenant scenarios. However, after those configuration options were applied, the cloud expanded the network information with extra details that are listed in Table 4.11:

Table 4.11 contains three tenant networks. One belongs to the Test tenant as the account field indicates and the other two belong to Tenant1, one for each user. Moreover, this table contains the Network CIDR and the vRouter Virtual Private Cloud (VPC) connected to it. However, the most important detail that links network information to the cloud infrastructure details presented above is the VLAN ID. This table clearly states the VLAN ID reserved for each tenant/user network and the VLAN ID here matched the VLAN ID shown in Tables 4.7,4.9 and 4.10. The DeepDive information has now discovered which resource belongs to which tenant.

Name	**TestNT**	**T1u1NT**	**T1u2NT**
ID	ff012e06-3de8-439b-aceb-9ed64cb7dad2	37548a93-1e62-4f72-9fc3-84ce383b397d	50a955f3-1e01-4021-82e0-7f040602811a
Type	Isolated	Isolated	Isolated
VPC ID	66606d26-0964-4e8b-af53-2768d2dd6597	013e96cb-d580-42ae-a85f-6cd522765e13	55708efb-4af3-4502-9b7f-e737a9cf17a0
VLAN/VNI ID	18	17	11
Broadcast uri	vlan://18	vlan://17	vlan://11
Network Offering	Offering for Isolated Vpc networks with Source Nat service enabled	Offering for Isolated Vpc networks with Source Nat service enabled	Offering for Isolated Vpc networks with Source Nat service enabled
CIDR	192.168.1.0/24	192.168.2.0/24	192.168.3.0/24
Network Domain	cs3cloud.internal	cs4cloud.internal	cs4cloud.internal
Account	Test	Tenant1	Tenant1

Table 4.11: Tenants Network Details from Management Server Console.

A CloudStack definition of a VPC Virtual Router is required here. A "CloudStack Virtual Private Cloud is a private, isolated part of CloudStack. A VPC can have its own virtual network topology that resembles a traditional physical network. A VPC acts as a container for multiple isolated networks that can communicate with each other via its virtual router [158]."

The Cloud Virtual Router is a form of System Virtual Machine offered by CloudStack, on tenant demand, and provides the network service. This vRouter (system VM) can only be logged into via the cloud Admin for troubleshooting and configuration. Tenants cannot login, and can only perform limited actions,such as port forwarding configuration and connecting their VMs through their private network. Tenants can create the virtual router after creating their private network through the system service offering, that is co-related with the network offering. Physically the router (system VM) is allocated in the Host node and contains several network interfaces each assigned an IP which has significant meaning. Each vRouter consists of three NICs and they perform the following[148]:

- One acts as a gateway for the guest network,

- Another contains the local link IP address to be connected to the CloudStack system for configuration purposes and
- One is assigned with a public IP address and configured with DNAT to facilitate the tenant VMs with an Internet connection.

Name	**TestVR**	**T1user1VR**	**T1user2VR**
ID	66606d26-0964-4e8b-af53-2768d2dd6597	013e96cb-d580-42ae-a85f-6cd522765e13	55708efb-4af3-4502-9b7f-e737a9cf17a0
Account	Test	Tenant1	Tenant1
CIDR	192.168.1.0/24	192.168.2.0/24	192.168.3.0/24
VPC Router Details			
Name	r-3-VM	r-5-VM	r-7-VM
DNS	192.168.31.2	192.168.31.2	192.168.31.2
Gateway	192.168.31.2	192.168.31.2	192.168.31.2
Public IP Address	192.168.31.102	192.168.31.103	192.168.31.104
Guest IP Address	192.168.1.1	192.168.2.1	192.168.3.1
Link Local IP Address	169.254.0.113	169.254.1.14	169.254.1.233
Service Offering	System Offering For Software Router	System Offering For Software Router	System Offering For Software Router
Account	Test	Tenant1	Tenant1

Table 4.12: VPC Virtual Router Details.

As explained above, vRouters are often VMs located in the Host Node, which explains the extra VMs found by the virsh tool. The VM name in Table 4.9 matches the name presented in Table 4.12, which gave the information of the owner of the VMs found in Table 4.9. Moreover, following the explanation above, each router contains 3 NICs and this matches the details in Table 4.9. However, Table 4.9 showed the interface name as Net# and its MAC address while the Management console provided the IP addresses. The question now was how to match the IP addresses to their correct network in order to map them into Figure 4.7 The answer is to login to each router VM and find each NIC MAC then match it with the one in Table 4.9 as is extensively done in Table 4.13.

Virtual Routers System VMs: The router system VM can be accessed through the console under the Infrastructure tab and then choosing Virtual Routers.

Information in Table 4.13 was collected from Virtual Routers interface under the Infrastructure tab in the console. In order to map the information of the vRouters in Figure 4.7, information was linked from the following tables; the Guest Network in Table 4.11, the VPC Virtual Router information in Table 4.12, the Infrastructure Virtual Router in Table 4.13 and the Libvirt XML File in Table 4.9. We matched the information from each table to find the link and map it to Figure 4.7. The same procedure applied for the guest VMs; using the Instance Table, instance console and XML Files, and then completed by mapping the information into the DeepDive diagram.

System VMs from Infrastructure Tab

At the time of defining the POD in the Zone a management subnet must be defined for the system VMs. System VMs are considered as being internal cloud resources that uses management traffic to communicate to each other or with the cloud management. There are several system VMs in CloudStack such as a Console Proxy VM, Secondary Storage VM and Virtual Router. CloudStack is accountable for the creation and management of the system VMs, which are responsible for performing various tasks in the cloud. For example, the Secondary Storage VM, as seen in Table 4.14, created after adding the Secondary Storage in the Zone, takes major responsibility for any task related to the secondary storage such as managing the templates in the secondary storage, handling the snapshots for backup, and managing all communication through the secondary storage VM from any component in the cloud [148]. The Console Proxy VM also in Table 4.14 is responsible for facilitating the instance's console viewing via the CloudStack web interface for the tenants and the instance VNC port [159].

IP addresses were listed but we could not find what they are connected to until we accessed the VMs and applied some network commands such as *ip a*. The IP address displayed in the Management server console was matched with the Ethernet label, e.g eth0 and MAC address, which was matched with the record from the XML file of the VM. We observed that S-2-VM had an extra network interface (eth3) that is not displayed in the Management server console and yet it was listed in the VM OS and the XML file. The eth3 interface was mapped in Figure 4.7 with the IP address 192.168.31.129 as it matched information in Table 4.8 with a MAC address of 06:88:18:00:00:09.

Tenants Instances from Instances Tab

Tenants instances from the Instances Tab are shown in Table 4.15.

Using a similar approach to mapping the Virtual Router VMs to Figure 4.7, we found that the

Name	**r-3-VM**			**r-5-VM**			**r-7-VM**		
Network ID	ff012e06-3de8-439b-aceb-9ed64cb7dad2			37548a93-1e62-4f72-9fc3-84ce383b397d			50a955f3-1e01-4021-82e0-7f040602811a		
Public IP Address	192.168.31.102			192.168.31.103			192.168.31.104		
Guest IP Address	192.168.1.1			192.168.2.1			192.168.3.1		
Link Local IP Address	169.254.0.113			169.254.1.14			169.254.1.233		
Compute offering	System Offering For Software Router			System Offering For Software Router			System Offering For Software Router		
Account	Test			Tenant1			Tenant1		
VPC ID	66606d26-0964-4e8b-af53-2768d2dd6597			013e96cb-d580-42ae-a85f-6cd522765e13			55708efb-4af3-4502-9b7f-e737a9cf17a0		
NICs	**NIC1**	**NIC2**	**NIC3**	**NIC1**	**NIC2**	**NIC3**	**NIC1**	**NIC2**	**NIC3**
Type			Isolated			Isolated			Isolated
Traffic Type	Control	Public	Guest	Control	Public	Guest	Control	Public	Guest
Netmask	255.255.0.0	255.255.0.0	255.255.0.0	255.255.0.0	255.255.0.0	255.255.0.0	255.255.0.0	255.255.0.0	255.255.0.0
IP Address	169.254.0.113	192.168.31.102	192.168.1.1	169.254.1.14	192.168.31.103	192.168.2.1	169.254.1.233	192.168.31.104	192.168.3.1
Network ID	e9720fbd-8a24-4697-ae6b-1a857b7c2d36	6f6c94cb-7d4c-403b-bc35-f8fb979b5cd0	ff012e06-3de8-439b-aceb-9ed64cb7dad2	e9720fbd-8a24-4697-ae6b-1a857b7c2d36	6f6c94cb-7d4c-403b-bc35-f8fb979b5cd0	37548a93-1e62-4f72-9fc3-84ce383b397d	e9720fbd-8a24-4697-ae6b-1a857b7c2d36	6f6c94cb-7d4c-403b-bc35-f8fb979b5cd0	50a955f3-1e01-4021-82e0-7f040602811a
Isolation URI		vlan://untagged	vlan://18		vlan://untagged	vlan://17		vlan://untagged	vlan://11
Broadcast URI		vlan://untagged	vlan://18		vlan://untagged	vlan://17		vlan://untagged	vlan://11
MAC from Console:	(Eth0) 0e:00:a9:fe:00:71	(eth2) 06:00:ca:00:00:17	(eth2) 02:00:52:86:00:02	(eth0) 0e:00:a9:fe:01:0e	(eth1) 06:2f:60:00:00:18	(eth2) 02:00:46:c8:00:02	(eth0) 0e:00:a9:fe:01:e9	(eth1) 06:59:68:00:00:19	(eth2) 02:00:63:79:00:02

Table 4.13: Virtual Router System VMs Details.

Name	v-1-vm	s-2-VM
ID	e66ce844-f079-4bb7-a1c1-b40c516acd8d	1c0f33bf-6784-4681-aa8b-310b061211be
type	Console Proxy VM	Secondary Storage VM
Public IP Address	192.168.31.100 (eth2) MAC: 06:e9:a6:00:00:15	192.168.31.101(eth2) MAC: 06:ee:34:00:00:16
Private IP Address	192.168.31.121(eth1) MAC:06:3d:d6:00:00:01	192.168.31.125 (eth1) MAC:06:d4:3a:00:00:05
Link Local IP Address	169.254.0.181 (eth0) MAC:0e:00:a9:fe:00:b5	169.254.2.24 (eth0) MAC:0e:00:a9:fe:02:18
Gateway	192.168.31.2	192.168.31.2

Table 4.14: Console Proxy and Secondary Storage System VMs Details.

Name	T1U2VM	T1U1VM	TestVM
Template	CentOS 5.5(64-bit) no GUI (KVM)	CentOS 5.5(64-bit) no GUI (KVM)	CentOS 5.5(64-bit) no GUI (KVM)
Compute offering	Small Instance	Small Instance	Small Instance
# of CPU Cores	1	1	1
CPU (in MHz)	500	500	500
Memory (in MB)	512	512	512
Account	Tenant1	Tenant1	Test
Network			
Network Name	T1u2NT	T1u1NT	TestNT
Type	Isolated	Isolated	Isolated
IP Address	192.168.3.65	192.168.2.198	192.168.1.73
Gateway	192.168.3.1	192.168.2.1	192.168.1.73
Netmask	255.255.255.0	255.255.255.0	255.255.255.0
MAC from Console	02:00:7f:e8:00:01	02:00:12:bc:00:01	02:00:1c:0f:00:01

Table 4.15: Tenants VMs Details from Instance Tab.

tenant VMs in the Management server console do not show the VM label according to the XML file and this could only be found by logging into the VM. We then matched the IP and MAC of the VMs as listed in Table 4.15 with the MAC from the XML file in Table 4.10.

Thus the DeepDive was completed on CloudStack and Figure 4.7 was complete.

4.5 Technical implications

4.5.1 Testbed Constructing Obstacles

The following includes the technical obstacles that were faced during the CloudStack set up:

1. The initial all-in-one attempt (pre-structure) revealed several technical issues:
    a) As with OpenStack we used a powerful server (Gorman) and built the cloud on a virtualized layer. Selecting among the virtualized platforms was time consuming.
        i. The CloudStack setup failed after the cloud-agent was installed and KVM tested "kvm-ok". This implied that KVM acceleration could be used.
        ii. The CloudStack-agent persisted in using KVM as the host node and the use of VT-X/AMD-V was a requirement. As in OpenStack we used VirtualBox.
        iii. The virtualized environment was tested for virtualization extensions using the command "*sudo egrep -c '(vmx|svm)' /proc/cpuinfo*" and the result was 0, hence VT-X/AMD-V is not supported.
        iv. Our investigation revealed that "KVM requires VT-X/AMD-V, but Virtual Box does not pass VT-X/AMD-V to the guest operating system, Therefore, KVM can't run in Virtual Box"[160].
        v. Moreover, the choice to switch to QUEM as we did in OpenStack was out of question. As the resource [161] stated, the CloudStack agent.property "set the hypervisor type, values are: kvm, lxc" only.
        vi. We attempted to configure the host as a XEN server but none of the configurations were successful and this option drove us further away from the OpenStack cloud testbed configuration.
        vii. We switched the virtual environment to VMWare Workstation 12, and the result of "*sudo egrep -c '(vmx|svm)' /proc/cpuinfo*" was higher than 1, equal to the CPU

assigned for the host node (VM). This proves that we had a working environment through VMWare rather than through VBox which had failed.

2. It took us four attempts to reach the satisfactory two-node cloud structure with KVM host and OVS switching and the following are some of the issues behind such attempts:
    a) After installing NFS in management Node, the NFS file mount was denied,
        i. We changed the NFS version configuration from 3 to 4 and removed the export path from their designated separate partitions to the boot partition.
    b) After creating the cloud Advanced Zone setup from the Management server console, the Secondary storage was not recognized, templates were not available and both system VMs were running but their agent state was disconnected and hence could not create instances.
        i. The attempt to delete and recreate the System VMs failed.
        ii. The attempt to delete the Zone and recreate it failed, because the POD was undeletable and the primary storage is a non-destroyed volume even with a deletion attempt from the cloud management DB.
        iii. Attempting the System VM troubleshooting using [162], revealed that the IP address of NFS in the System VMs automatically selected the old IP address of the Management server than the statically configured IP address.
    c) Failed provision resources for some tenants:
        i. CloudStack made the cloud infrastructure load on the Host Node and hence it must be empowered with the highest resources.
        ii. We had to shutdown the whole cloud, increase the CPU and memory capacity for the KVM Host Node, and then repower the cloud.

4.5.2 The DeepDive Obstacles

The following are the technical obstacles that were faced during the CloudStack DeepDive :

1. Management server console:
    a) Network Provider OVS was disabled:
        i. During the DeepDive , information collection issues on VXLAN were encountered. Despite the cloud running correctly with virtual interfaces added normally to OVS

bridges, we found that the OVS Network Service Provider was disabled. However, there was no sign of VLAN tags or VXLANs in the OVS bridges.

ii. According to [163], theKVM OvS VifDriver does not support VXLAN and a solution was to drop the use of OVS and replace it with Linux bridges, and then configure the KVN with BridgeVifDriver.

iii. As our major aim was to create the CloudStack Testbed as close as possible to the OpenStack Testbed, we decided to solve this issue by keeping the OVS configuration and, alternatively, to change the isolation type from VXLAN to GRE as shown in the example[164].

iv. Another implication was that re-configuring the physical network was not possible unless we deleted the entire Zone and re-created it, and that required us to re-create the tenants and their scenarios.

v. Once the OVS network provider is enabled in CloudStack and the tenant VNs re-created, the interfaces were added to the OVS bridges along with their assigned VLAN specifically in Cloudbr1 bridge as is shown in Table 4.10.

2. Management host: Scripts provided very limited information. Due to the fact that none of the cloud network infrastructure was included in the Management Node, we had to cut down the commands running in the designated scripts.

3. KVM Host: Instance XML files were nowhere to be found in the KVM Host. Unlike OpenStack, which uses the default Libvirt path to store the VM configuration parameters XML file, we had to modify the script files for collecting the content of XML files from the source path and used the commands *virsh list –all, Virsh dumpxml <VMName> > vmname.xml* [165] to get the instance XML file contents.

4. The DeepDive analysis was similar to the OpenStack analysis. However, due to the infrastructure differences between OpenStack and CloudStack, the manual analysis had to be adjusted to fit the CloudStack information collected.

4.6 Conclusions

This chapter achieved two research objectives by firstly defining the research required resources through outlining the CloudStack setup. The identification of the construction of the cloud network infrastructure, VN components and Isolation mechanism was performed via the DeepDive methodology on the CloudStack Testbed.

A fully operating CloudStack testbed, functioning similar to the OpenStack testbed, was analysed and multiple tables of data were generated to allow a map of the cloud infrastructure to be created. From this, the outcome was a complete CloudStack Network Infrastructure diagram, which showed virtual network isolation mechanisms (e.g. VLAN) and network components (e.g. vRouters, OVS bridges). Further, connections within the cloud are typed and labelled as shown CloudStack Network Infrastructure DeepDive diagram in Figure 4.7 on page 125.

Comparably, the CloudStack Testbed Infrastructure DeepDive representation is less dense than that with OpenStack diagram represented in Chapter 3. The difference in technology resulted in different Cloud infrastructures, although matching technology options and scenarios were used. As an outcome, the third research objective, the identification of a Cloud Network Infrastructure methodology, provided a cloud infrastructure and isolation knowledge guide for OpenStack (Chapter 3) and CloudStack (Chapter 4) to be used in the remaining research objectives.

Consequently, the next chapter focuses on penetration tests on the two experimental testbed clouds to discover if an attacker can determine cloud virtual network isolation information as discovered by DeepDive methodology.

CHAPTER FIVE

PENETRATION TESTING FOR DATA LEAKAGE

Following on from the setup of the two testbeds and the mapping of their network infrastructures in Chapters 3 and 4, this chapter outlines the security tests for detecting any Cloud Network infrastructure components using Penetration Testing. The fourth research objective, stated in Chapter 1 (§1.6), concerned detecting and determining the characteristics of the leaked data, and this is achieved and described in this chapter. The aim was to collect leaked information from running bi-directional Penetration Tests (Internally and Externally) on the two testbeds and to then analyse this information. If components and configuration details of the tenant VNs within the Cloud Network infrastructure could be determined, then Virtual Network Isolation is weak and insecure.

5.1 Penetration testing and tools

The research problems in §1.6 stated that although VN Isolation is a core element of cloud security, the literature shows that very little experimental work has been conducted to test, discover and evaluate any VN isolation data leakage.

We recall our Null hypothesis that the security of the IaaS Cloud Virtual Network Isolation is robust against Data Leakage caused by current penetration tools. This chapter empirically tests this hypothesis.

As stated in the Problem Statement of §1.6, the aim of this research is to detect or evaluate information leakage from cloud virtually isolated networks. Therefore, questions arise as to how data leakage can be determined and what the characteristics can be leaked, such as IP addresses, OS type, VM types, open ports etc?

Chapters 3 and 4 answered two of the research objectives questions; What are the required resources to evaluate CVNI? and What is the best methodology to identify the construction of the cloud network infrastructure, VN components and Isolation mechanism? The first question was answered by selecting OpenStack and CloudStack as our base testbeds for this study. The second question was answered when the DeepDive Methodology was developed to identify the construction of the cloud network infrastructure, VN components and Isolation mechanisms.

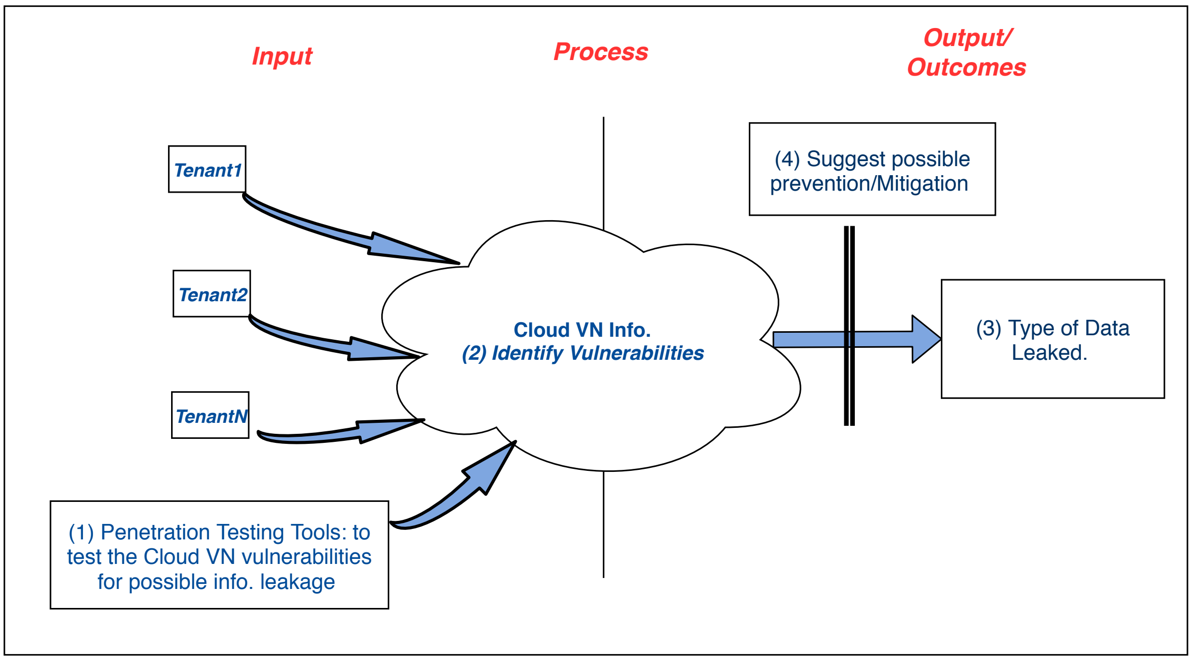

Input
Process
Output/ Outcomes
Tenant1
Tenant2
TenantN
(1) Penetration Testing Tools: to test the Cloud VN vulnerabilities for possible info. leakage
Cloud VN Info.
(2) Identify Vulnerabilities
(4) Suggest possible prevention/Mitigation
(3) Type of Data Leaked.

Figure 5.1: Research Map

We mapped the research as shown in Figure 5.1 to show the flow of the empirical work performed to answer the research questions. The research was divided into three main stages as follows:

1. Stage one, Setup:
   - Build the Advanced VN Cloud Testbeds with multi-tenancy scenarios.
   - Select one tenant to be a rogue tenant and use the initial information to perform bi-directional penetration testing attacks (External and Internal)
2. Stage two, Penetration Testing:
   - The rogue tenant launches the currently available penetration tools - the Kali Tool kits- to test the Cloud VN Vulnerabilities for possible information leakage.

- Vulnerabilities are identified and their relevance to VN, the impact on identifying the co-existing VN of the other tenants or the breaking the VN Isolation are determined. At this stage the decision was made to accept or reject the null hypothesis.

3. Stage three, Outcome:

   - An outcome is to find and categorize leaked data them as well as detect or determine their source and cause.

   - Finally, and based on the results above, we suggest possible prevention or mitigation guidelines.

This chapter outlines Stage 2 in the above list and introduces our Penetration Testing procedures. A Penetration Testing methodology is described as “a roadmap with practical ideas and proven practices that can be followed to assess the true security posture of a network, application, system, or any combination thereof “[125].

5.1.1 Tools Used

Following the application od Kali Linux, data was discovered using tools for information gathering, vulnerability analysis, web application analysis, database assessment, password attacks, reverse engineering, exploitation, sniffing & spoofing, post exploitation, reporting and forensics. To match the official penetration test cycle, the Kali categorizations of information gathering, tools for vulnerability analysis, exploitation, sniffing & spoofing were selected. Although the cloud systems provide a web dashboard, this process moved focus away from the infrastructure to another penetration test methodology. The information presented in the dashboard originally is stored in the cloud database, which required another penetration test process. Labarge and Maguire [44] did extensive work on cloud web dashboard penetration testing using an old version of an OpenStack cloud, and did not target VN information. Their password attacks and reverse engineering are out of the scope of this work, as the penetration testing performed was not on an enterprise cloud with employees and tasks. We do not aim to break system passwords as this is subject to the administrative setting of those passwords. To execute a full vulnerability assessment, according to the literature, it is not sufficient to find vulnerabilities without confirming exploitability. Hence, it was decided to focus on penetration testing rather than vulnerability assessment.

Nessus is mainly designed as a vulnerability assessment tool. However, it also performs validity checks of found vulnerabilities, and provides attack lists to exploit those vulnerabilities, which saves us from going through the list of vulnerabilities and examining them [134][133]. Finally, forensic analysis is not relevant to the current study; hence it is not included in the testing.

Due to the large number of potential tools, a pre-experiment test was performed using a demo cloud (devstack[166][167] on the Virtual Box virtualization environment), to select the most suitable tools to run the experimental penetration testing on the real testbed.

Given the limited information on the suitability of penetration tools for leaking cloud infrastructure information, and consequently for extracting VN relative information, over a hundred popularly recommend penetration tools were tested. These range between multi-purpose or single tools; web, GUI or CLI tools and locally executed or via Internet access tools. Most web and Internet accessible tools depend on their own defined databases of publicly connected systems, which do not apply to private, locally run cloud testbeds. Other tools use either a GUI or a Command Line Interface, locally executed, and provided more promising results. The tools that provided non-relevant information were removed. Further remaining tools in the Kali Tool Kit, with deeper examination, were found not to be as efficient or they gave the exact same information but were limited in comparison to other tools. For example, netdiscover[136] and arping[125] provided the same address information as Zenmap[125] and Sparta[127] did. However, Dmitry[125], unicornscan[1] and xprob2[2] gave more limited information about ports than Zenmap and Sparta. For vulnerability scanning, Nessus was used to discover and identify the vulnerabilities based on the information found by Zenmap and Sparta.

The tools that were selected for this empirical work include Zenmap, Sparta and Nessus, as discussed in Chapter2. The following sections detail the results of this work.

5.2 OpenStack Penetration Testing

Figure 5.2 shows an OpenStack-Mitaka cloud operating on two Ubuntu nodes (Controller and Compute nodes) built on a Virtual Box (VBox) virtualization environment within the local server, the Gorman Server. The cloud consists of two main networks; one for Management, 192.168.137.0/24, and the external network with address 192.168.100.0/24. The network address of 10.0.0.0 was used for connecting the node to the Internet for cloud package installation and has no role in the cloud operation itself.

Two tenants were created, as explained in Chapter3; the Test tenant contains one user with the same name and the other tenant is Tenant1 which contains two users: T1User1 and T2User2. Each user owns one VM (instance) except for Test, which has two VMs; a private Virtual

[1]https://tools.kali.org/information-gathering/unicornscan

[2]http://knoxd3.blogspot.com/2013/07/how-to-use-xprobe2-in-kali-linux.html

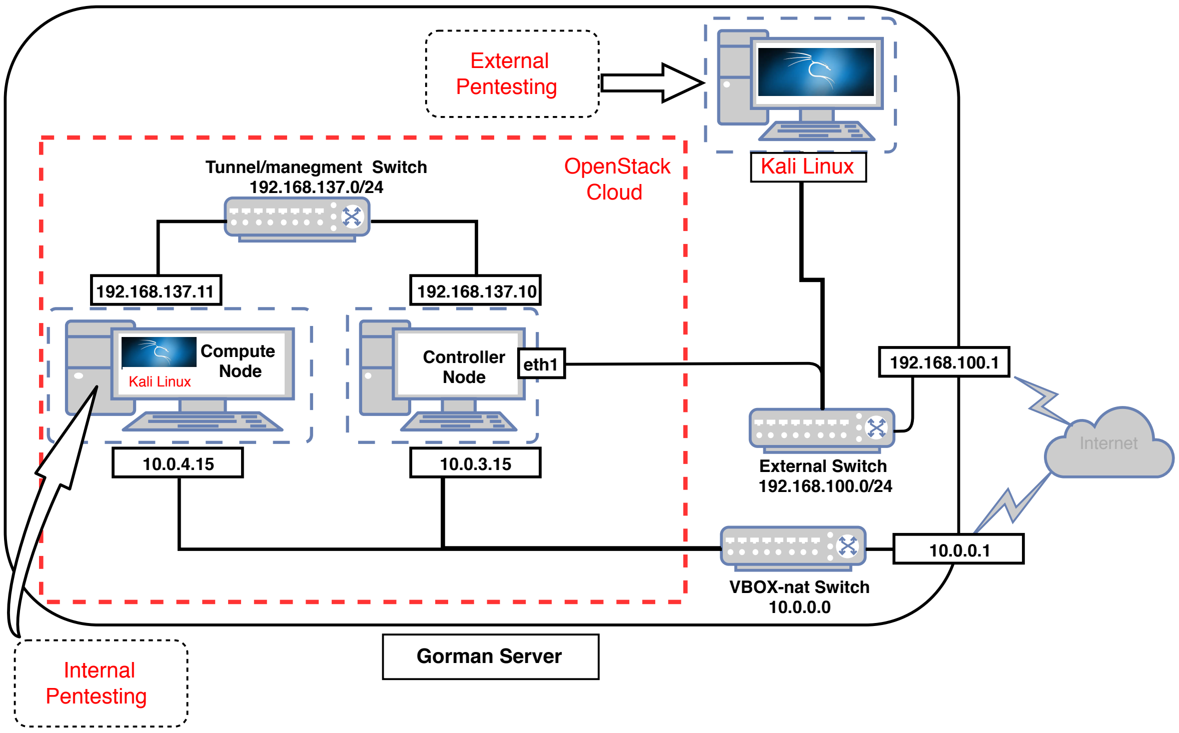


Figure 5.2: The OpenStack Bi-directional Penetration Testing Methodology

Network with a private router connected to a public router (through the external network) to allow an instance to be connected from the outside using SSH. Tenants/ users are provided the required credentials to access their instances and find more details about their assets in the cloud by the Openstack Horizon dashboard webpage. The preliminarily information is delivered to from the cloud provider to the tenants. As the Test tenant, we have performed a penetration test externally. Using the Kali Linux OS as the Tenant Terminal machine connected to the external network, we accessed the Openstack cloud and operated the tools installed in Kali to perform pentesting on the Horizon web-based application and the SSH connection. The Internal penetration test, the second Test VM is a specially modified light version of Kali Linux and performs the same penetration test procedure as in the external penetration test. Both the internal and external penetration testing are types of Black Box testing.

The main goal of this test is to answer the question, *Can the Test Tenant, using Pentesting, uncover the underlying layer of the cloud, discover the details of the Virtual Network and traffic of other tenants sharing the same Cloud?* If the pentesting tools expose the cloud environment the tenants share and the information about other tenants VN, then this is a serious vulnerability of VN isolation.

There may be issues of scale here which have not been addressed due to research limitations.

However, external and internal testing on minimal clouds with a small number of tenants and users, should demonstrate process viability and potential vulnerabilities.

5.2.1 External Pentesting

For experimentation, the Kali Linux installed externally is the "Kali Linux custom image" provided by Offensive Security [3] and the image version used was *"Linux kali 4.3.0-kali1-amd64 #1 SMP Debian 4.3.3-7kali2 (2016-01-27) x86_64 GNU/Linux"*.

In this scenario, the Test tenant is the attacker, who has prior information about the cloud URL (http://192.168.137.10/horizon) and the floating IP addresses – the External network - of the Test two VM instances in the cloud: 192.168.100.11 and 192.168.100.16

The OpenStack External Penetration testing was the first attempt in using the pentesting methodology and therefore several commands were executed as an initial test drive; nmap, TCPdump and amap, Fping. After comparing the results with what we found using Zenmap and Sparta the test-drive tools were very limited. However, the test drive provided a good view of the live host on the pre given network range other than what the Test tenant already knew. For example, the Test tenant knew that the IP address 192.168.37.10 had been used to access the web dashboard. The test drive tools revealed another active IP, 192.168.137.11, in the cloud management network, and five other active IPs within the external/public network. These were later used as another input for Zenmap and Sparta.

5.2.1.1 Zenmap output

Zenmap uses nmap as a scanner with a profile name of *“Slow Comprehensive Scan”*. This is translated Zenmap to the following nmap statement after including the targeted network of the management network, see Figure 5.2:

```
nmap -sS -sU -T4 -A -v -PE -PP -PS80,443 -PA3389 -PU40125 -PY -g 53 --script
 "default or (discovery and safe)" 192.168.137.10-11
```

Despite zenmap providing a list of nmap statements, we used the Slow Comprehensive Scan as it is uses the full capability of nmap and guaranteed that no useful information was lost. The summary of the analysis of the output is presented in Table 5.1.

Table 5.1 presented only two active IP addresses in the management network: 192.1638.137.10 and 192.168.137.11. Each IP address listed several open ports using TCP and UDP, where some

[3]https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-hyperv-image-download/

Host IP	**Port**	**Service**	**Version**
192.168.137.10	22/TCP	SSH	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
	3306/TCP	MYSQL	MYSQL
	5000/TCP	HTTP	Apache HTTPd 2.4.7 ((Ubuntu))
	67/UDP	DHCPs	?
	123/UDP	NTP	NTP v4 (secondary server)
192.168.137.11	22/TCP	SSH	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
	5900/TCP	VNC	VNC (protocol 3.8)
	5901/TCP		
	5902/TCP		
	5903/TCP		
	67/UDP	DHCPs	?
	123/UDP	NTP	NTP v4 (secondary server)

Table 5.1: Zenmap Analysis of Openstack Management Network 192.168.3137.0/24

ports are of interest and others are not, hence we choose to ignore the latter. For example, the IP address 192.168.137.10 listed the ports 3306/TCP for MySql service, 5000/TCP for HTTP (cloud dashboard) service and 123/UDP for the NTP service. Those ports are needed for the OpenStack packages that are vital for the installation of the OpenStack testbed as illustrated in Chapter 3. Additionally, port 67/UDP, which listens to the PHCP service, was ignored as although it was discovered via the cloud network IP, it was not relevant to the cloud system as it belongs to the network range 10.0.0.0/24 as shown in Figure 5.2. Zenmap didn't specify the OS of the node other than noting it was Linux. Port 22/TCP, associated with the SSH service, was ignored as it was open by default within the OS. However this port along with the HTTP port specifically indicates that the system is an Ubuntu Linux distribution. Hence, from the ports belonging to 192.168.137.10 a conclusion could be made that this IP belongs to the OpenStack Controller Node.

Logically, every cloud needs a hypervisor node and as there is only one IP remaining on the management network, Table 5.1 also shows the first IP traces of services run in the Controller Node. The second IP address can be presumed to belong to the Hypervisor (Compute Node).

Looking at the second IP address in the table and the associated open ports, this confirms our assumption. The port ranges of 5900/TCP – 5903/TCP belong to VNC service, which as explained above is used for remotely controlling the VMS inside the cloud. There are four VNC ports open, which indicated that there are four VMs running inside the cloud. At this point it was concluded that VNC ports are resident in the Compute Node, however we had no knowledge of which port belongs to which VM and who owns each VM. On that note, Zenmap flagged a warning on the VNC ports as there is no security type applied and the "Server does not require authentication". As in the Controller Node, we ignored the DHCP ports and we used SSH to identify the OS of the Compute Node to be Ubuntu Linux.

As the attacker is a tenant of the cloud, they have preliminary information about the two addresses of the public ranges. These belong to the Test tenant's two VMs, 192.168.100.11 and 192.168.100.16, which are used to access the instances remotely via the SSH protocol. Zenmap exposed additional addresses as listed in Table 5.2.

Host IP	MAC	Port	Service	Version
192.168.100.10	FA:16:3E:D0:0E:A9	68/UDP	DHCPC	
		1124/UDP	pvmmcontrol	
		1646/UDP	radacct	
		5002/UDP	rfe	
		16974/UDP	unknown	
		17533/UDP	unknown	
		18994/UDP	unknown	
		19039/UDP	unknown	
		19936/UDP	unknown	
		22029/UDP	unknown	
		28973/UDP	unknown	
		49174/UDP	unknown	
192.168.100.11	FA:16:3E:D0:0E:A9	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
192.168.100.12	FA:16:3E:5A:08:14			
192.168.100.13	FA:16:3E:61:6B:F7			
192.168.100.14	FA:16:3E:5A:08:14	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
192.168.100.15	FA:16:3E:61:6B:F7	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
192.168.100.16	FA:16:3E:D0:0E:A9	22/TCP	SSH (closed)	

Table 5.2: External Pentesting Zenmap Analysis of Openstack External Network 192.168.100.0/24

One thing to be noted after analyzing the information collected from Zenmap on the public network range is that there are seven active IP addresses and only three physical addresses. Several Public IP addresses share the same physical address. For example, the IP addresses 192.168.100.10, 192.168.100.11 and 192.168.100.16 share the physical address

FA:16:3E:D0:0E:A9 and those belong to the Test tenant. Therefore, it was concluded that at least there are another two tenants active in the cloud. The tenant's own resources are the IP addresses 192.168.100.12 and 192.168.100.14 because they share the MAC address FA:16:3E:5A:08:14. However, the other tenant owns 192.168.100.13 and 192.168.100.15 as they share the MAC address FA:16:3E:61:6B:F7. From the pre-knowledge we know that 192.168.100.11 is a Test VM which uses SSH to connect the VM from outside, while 192.168.100.16 is a VM reserved for internal testing. Test does not require an external connection to it as SSH is closed. Based on this we can assume that the other tenant IPs with SSH ports open might be the tenant's own IPs, however this is not confirmed at this point. At this level the resources with IP that do not show any open ports are still unknown in nature. The Test tenant knows of 192.168.100.10 as it is the vRouter connecting the Test VN. Table 5.2 shows numerous unknown ports other than DHCP, whose purpose we do not know at this stage.

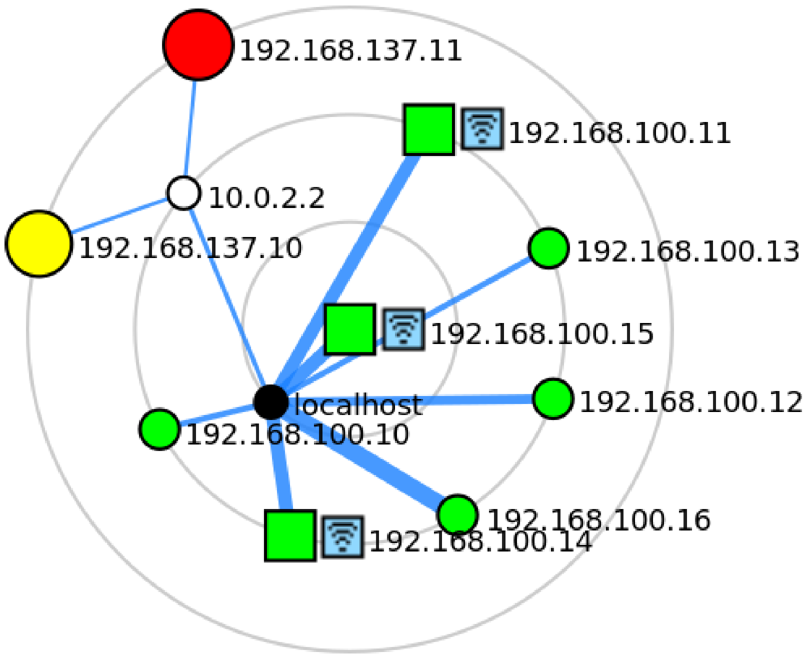


Figure 5.3: Zenmap External Pentesting Output Graphical Representation of OpenStack Cloud Active Resource Topology.

Zenmap provides a graphical representation of the scanned system based on the scans run. Figure 5.3 represents the OpenStack resources found and listed in Tables 5.1 and 5.2 by running two nmap scans on the OpenStack Cloud Networks 192.168.137.0 and 92.168.100.0. Their location was graphically represented by the network distance found using the traceroute command within each nmap script scan. Finally, IP address 192.138.100.15 is not part of the test as it is the address of the External Kali Linux where the Zenmap scan was launched. The targets, which are colour coded, represent how risky they are due to their open ports; 192.168.137.10 is represented by yellow to indicate that it contains between 3-6 open ports, while 192.168.137.11 is the riskiest as the Zenmap red colour code means the target contains over 6 open ports.

Using Zenmap, open ports and active and physical addresses can be collated and unknown resources or links noted.

5.2.1.2 Sparta Output

Sparta uses the nmap scanner as the initial action using preliminarily input from the pentester/attacker and the outputs are used further by other phases of testing with tools within Sparta, such as nikto, hydra and others. Sparta runs nmap in five stages, each with its own list of port scanning and options, and results from each stage are accumulated and used to run further tests. There are no additional IP addresses found by Sparta that are not found by Zenmap and all ports found by Zenmap were found by Sparta. However, Sparta exposed more ports that were not visible by Zenmap, especially those marked as filtered or open filtered. Due to the long list of ports provided by Sparta for each targeted IP listed in Tables 5.1 and 5.2, the findings are summarised in Table 5.3.

IP address/ Ports per stages	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**	**Stage 5**
192.168.137.10	80	3306, 137, 161, 162, 1434	500, 5060	4369, 5000, 5672, 6080, 8774, 8775, 9191, 9292, 9696, 11211, 25672	35357
192.168.137.11		161, 162, 1434	500, 5060	5900, 5901, 5902, 5903	
192.168.100.10					
192.168.100.11	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.100.12					
192.168.100.13					
192.168.100.14	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.100.15	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.100.16	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		

Table 5.3: Sparta: External Pentesting Analysis of OpenStack Management Network 192.168.137.0/24 and External Network 192.168.100.0/24

As ports found by Zenmap were also found in Sparta and, due to the five map stages scripts, more were ports found by Sparta. This result confirmed the doubts raised in Zenmap with regard to whether or not these were VMs or other components. We know that 192.168.100.11 and 192.168.100.16 are the Test VMs and Table 5.3 shows the pattern of ports found in each stage of Sparta's five test stages. The Test VMs match the ports found in 192.168.100.14 and 192.168.100.15. Hence, we confirm that both targets are VMs and they belong to different tenants as found in Zenmap through the shared MAC addresses. However, unlike Zenmap, which exposed some unknown ports related to 192.168.100.10, Sparta did not discover any ports related to the targets 192.168.100.10, 192.168.100.12 and 192.168.100.13. To our knowledge 192.168.100.10 is related to the Test vRouter, however this did not provide us with information about the other IPs represented by other tenant's vRouter.

Another Sparta finding is related to the target 192.168.137.11 and ports found in the stage four scan. Sparta confirmed the warning raised by Zenmap related to the VNC ports (5900, 5901, 5902, 5903), and the lack of security for authentication. Sparta uses an embedded tool called VNCViewer that takes the exposure a step further than Zenmap, and viewed the active VMs in the cloud through the VNC ports. This is shown in Table 5.4

VNC Port	**Desktop Name**	**Tenant**	**Username/ Password**	**Information collected**	
				Command	**Values**
5900	instance-00000004	Test	Root/toor	IP a	Eth0: 10.190.0.7 MAC: fa:16:3e:00:ca:d3
				IP route	GW: 10.190.0.1
				Hostname	KaliLight
5901	instance-00000001	Test	Cirros/ cubswin:)	IP a	Eth0: 10.190.0.6 MAC: fa:16:3e:62:a6:c3
				IP route	GW: 10.190.0.1
				Hostname	my_first_instance
5902	instance-00000003	Unknown	Cirros/ cubswin:)	IP a	Eth0: 192.168.20.3 MAC: fa:16:3e:05:a6:8e
				IP route	GW: 192.168.20.1
				Hostname	T1u2vm
5903	instance-00000002	Unknown	Cirros/ cubswin:)	IP a	Eth0: 192.168.10.3 MAC: fa:16:3e:aa:8b:b8
				IP route	GW: 192.168.10.1
				Hostname	T1u1vm

Table 5.4: Sparta: VNCViewer Finding from VNC Ports in the Target 192.168.137.11.

Sparta VNCViewer is used on each VNC port on the Compute Node to access the cloud's VMs remotely and found the following:

- Ports 5900 and 5901 belong to the Test tenant VMs, where the attacker knows their credentials as well as the network details and host name. See Figures 5.4 and 5.5.
- Ports 5902 and 5903 belong to other tenants whom we do not know at this stage in the testing,

but the credentials are given in the login prompt as well as the host name as in Figures 5.6 and 5.7.

- Both the other tenant VMs were created from the sample image provided by the cloud platform for testing and this is a possible lack of awareness of the tenant case. The effects of this are noted in Chapter 6.

- Guessing the credentials provides an increased chance of learning about other tenants's network details, which matches the information in the DeepDive in Chapter 3, such as IP address, MAC address and gateways as presented in Table 5.4 and Figures 5.4 - 5.7.

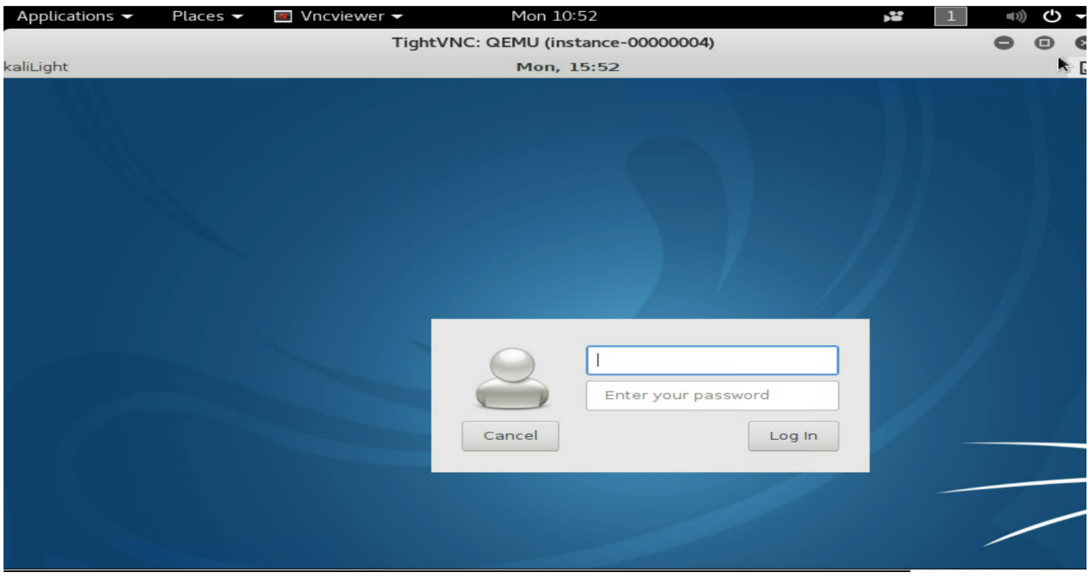
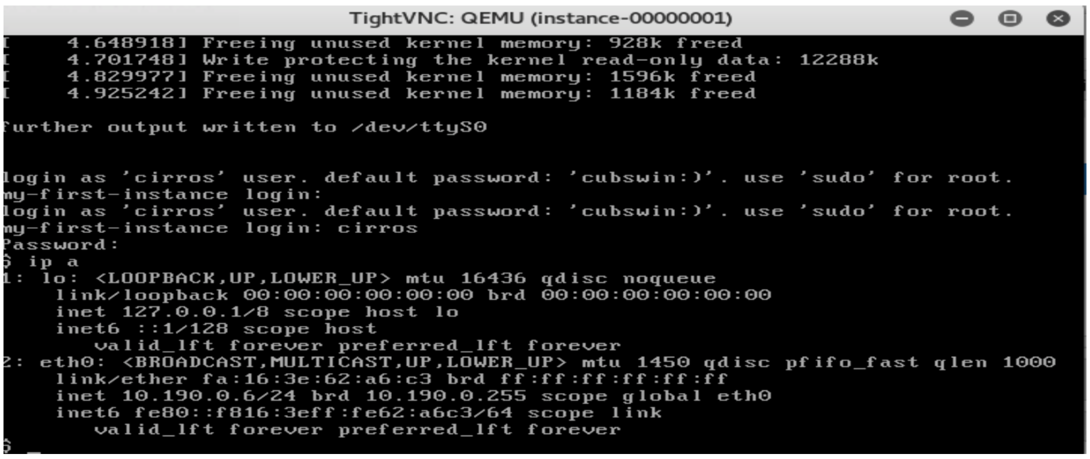


Figure 5.4: VNCViewer of instance-00000004.

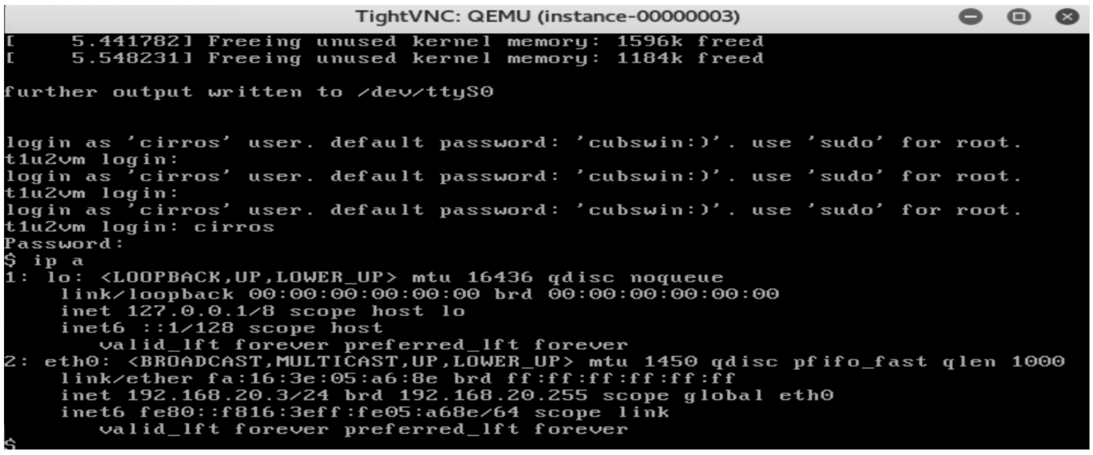


```
[     4.648918] Freeing unused kernel memory: 928k freed
[     4.701748] Write protecting the kernel read-only data: 12288k
[     4.829977] Freeing unused kernel memory: 1596k freed
[     4.925242] Freeing unused kernel memory: 1184k freed

further output written to /dev/ttyS0

login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
my-first-instance login:
login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
my-first-instance login: cirros
Password:
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc pfifo_fast qlen 1000
    link/ether fa:16:3e:62:a6:c3 brd ff:ff:ff:ff:ff:ff
    inet 10.190.0.6/24 brd 10.190.0.255 scope global eth0
    inet6 fe80::f816:3eff:fe62:a6c3/64 scope link
       valid_lft forever preferred_lft forever
$ _
```

Figure 5.5: VNCViewer of instance-00000001.



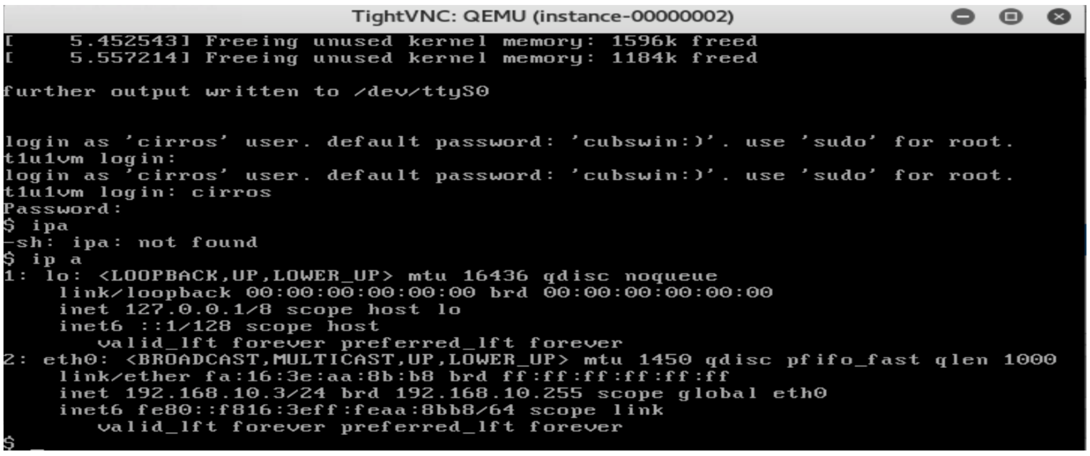
TightVNC: QEMU (instance-00000003)

```
[    5.441782] Freeing unused kernel memory: 1596k freed
[    5.548231] Freeing unused kernel memory: 1184k freed

further output written to /dev/ttyS0

login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
t1u2vm login:
login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
t1u2vm login:
login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
t1u2vm login: cirros
Password:
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc pfifo_fast qlen 1000
    link/ether fa:16:3e:05:a6:8e brd ff:ff:ff:ff:ff:ff
    inet 192.168.20.3/24 brd 192.168.20.255 scope global eth0
    inet6 fe80::f816:3eff:fe05:a68e/64 scope link
       valid_lft forever preferred_lft forever
$
```

Figure 5.6: VNCViewer of instance-00000003.



TightVNC: QEMU (instance-00000002)

```
[    5.452543] Freeing unused kernel memory: 1596k freed
[    5.557214] Freeing unused kernel memory: 1184k freed

further output written to /dev/ttyS0

login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
t1u1vm login:
login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
t1u1vm login: cirros
Password:
$ ipa
-sh: ipa: not found
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc pfifo_fast qlen 1000
    link/ether fa:16:3e:aa:8b:b8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.3/24 brd 192.168.10.255 scope global eth0
    inet6 fe80::f816:3eff:feaa:8bb8/64 scope link
       valid_lft forever preferred_lft forever
$ _
```

Figure 5.7: VNCViewer of instance-00000002.

Therefore, using Sparta allows an identification of infrastructure components and an understanding of tenant address values and names.

5.2.1.3 Nessus Output

Nessus is an "open source network vulnerability"[132] assessment tool [136][135]. Nessus is a toolkit providing several scanners, plugins, compliances, policies, scans and reports[131] [134][133]. The version used here for the External Kali Linux tests uses Nessus version "Nessus-6.10.3-debian6-amd64.deb" [4].

The Nessus test is the longest test in our penetration test, due to the limited information the scans could find from the cloud. We ran as many scans as possible and then analyzed the findings. We created 30 Nessus policies, and from these policies 60 scans were created to cover

[4]https://www.tenable.com/products/nessus-home

both 192.168.137.0/24 and 192.168.100.0/24 networks. Some of these scans revealed classified vulnerabilities as critical, medium and low and some revealed none. The Tables 5.5 and 5.6 show the Nessus scan results along with the targets and the number of classified vulnerabilities in each target.

S.Q	Scan	Targets	Vulnerabilities			
			Critical	**High**	**Medium**	**Low**
1	Basic Network Scan: All Ports 192.168.137.0	192.168.137.10			4	2
		192.168.137.11		2	1	2
3	Basic Network Scan: Custom 192.168.137.0	192.168.137.10			1	2
		192.168.137.11		2	1	2
3	Web Application Test: All Ports 192.168.137.0	192.168.137.10			2	
		192.168.137.11				
4	Web Application Tests: Custom 192.168.137.0	192.168.137.10		4	4	
		192.168.137.11				
5	Advanced Scan: Firewall Plugins-192.168.137.0	192.168.137.10		1		
		192.168.137.11				
6	Advanced Scan: General Plugins-192.168.137.0	192.168.137.10				
		192.168.137.11		1		
7	Advanced Scan: full scan-192.168.137.0	192.168.137.10		7	11	3
		192.168.137.11		5	7	3

Table 5.5: Nessus Scan: Vulnerabilities of 192.168.137.0/24 - Management Network Targets.

Nessus Scans provided a large amount of information of each target, for example the Network 192.168.137.0 has to target the cloud nodes (Controller and Compute Nodes). For those two targets 30 scans were run and two reports were collected from each scan; one is a summary and another is a custom report. Table 5.5 presents the data collected from the summary document detailing the number of classified vulnerabilities of each target. However, 7 out of 30 scans reported vulnerabilities across a range of vulnerability level classes, which range from High to Low as shown in Table 5.5. Appendix B contains the table of vulnerability names for the targets along with the suggested solutions.

The Nessus custom reports showed no information that would conflict with what has been found by Zenmap and Sparta, or any additional knowledge except for vulnerability details. Their severity for each target and specific ports used is also reported along with the best solution to secure that risk.

Similarly the Network 192.168.100.0/24 had 30 scans but only scans of three addresses showed vulnerabilities, as seen in Table 5.6. It is clear that the VMs are the most vulnerable part of the cloud and yet VMs can be secured better as demonstrated by 192.168.100.16 (KaliLight) showing no vulnerabilities compared to the VMs built by the test image provided by the cloud

S.Q	Scan	Targets	Vulnerabilities			
			Critical	High	Medium	Low
1	Basic Network Scan: All Ports 192.168.100.0	192.168.100.10				
		192.168.100.11	1		1	2
		192.168.100.12				
		192.168.100.13				
		192.168.100.14	1		1	2
		192.168.100.15	1		1	2
		192.168.100.16				
2	Basic Network Scan: Custom 192.168.100.0	192.168.100.10				
		192.168.100.11	1		1	2
		192.168.100.12				
		192.168.100.13				
		192.168.100.14	1		1	2
		192.168.100.15	1		1	2
		192.168.100.16				
3	Advanced Scan: full scan 192.168.100.0	192.168.100.10				
		192.168.100.11	1		2	3
		192.168.100.12				
		192.168.100.13		1		
		192.168.100.14	1		2	3
		192.168.100.15	1		2	3
		192.168.100.16				

Table 5.6: Nessus Scan: Vulnerability classes of 192.168.100.0/24 - External Network Targets.

platform. From investigating the vulnerability list of each VM, all VMs other than KaliLight use the same image source, hence, it is easier to reveal their system details more accurately. Target 192.168.100.13 unexpectedly exposed a high vulnerability named “Firewall UDP Packet Source Port 53 Rule set Bypass”. This target is a tenant vRouter as the port is 53/UDP, which represents a network service DNS server. However, despite the vRouters being consistent in the cloud, the question remains why only one vRouter has exposed a vulnerability and the other two did not.

The Nessus custom reports confirm all the information displayed by Zenmap and Sparta on the network 192.168.100.0/24 targets, and added the vulnerability details with the solution to manage them.

Using Nessus, vulnerabilities are listed for each target using the ports and protocols used. Thus, assuming an external attacker, the information gained from Nessus, Sparta and Zenmap has indicated network distances, open ports, addresses and both protocol and service vulnerabilities.

5.2.2 Internal Pentesting

Installing the Kali Linux version used in the External tenant was problematic and was solved using "Kali Linux Light 64 Bit" from Official Kali Linux Downloads [5], which had been modified to fit into the cloud systems and its hypervisor. The technical implementation issues are explained in detail in §5.5.

The internal penetration testing tool used is a Kali Linux VM installed by the rogue tenant –Test- as a VM within the cloud and is connected to the Test Tenant network. The Internal Penetration Testing followed the exact same process as the External Penetration Testing with the intention of learning further information as this test is the physically and logically closest to the targets. However, one of the major tests failed in the Internal pentesting; the Nessus scan failed to run even though we exhausted the troubleshooting resources and online support help as is illustrated in the OpenStack technical implications in §5.5.1. Hence, this test will only involve the Zenmap and Sparta scans.

One observation is that the management network (192.168.137.0/24) is not visible and is not connected to the inner side of the cloud, because of how the testbed is built using the OpenStack installation instruction document. As the Internal pentesting is launched from the Kali VM in the Test Tenant, only the private network Test resources are connected to, yet the public (external) network are visible by Zenmap and Sparta. The next sections represent a summary of the findings from the internal testing using Zenmap and Sparta.

5.2.2.1 Zenmap output

IP	MAC	Port(s)	Service	Version
10.190.0.1	FA:16:3E:21:F6:3E			
10.190.0.5	FA:16:3E:F5:47:A5	53/TCP 53/UDP	domain domain	dnsmasq 2.68 dnsmasq 2.68
10.190.0.6	FA:16:3E:62:A6:C3	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
10.190.0.7		22/TCP 68/UDP	SSH DHCPC	OpenSSH 7.4p1 Debian 6 (protocol 2.0)

Table 5.7: Internal Pentesting: Zenmap Analysis of the Openstack Test Private Network 10.0.0.0/24.

Table 5.7 shows the analysis of nmap as launched by Zenmap on the Test private network. Test

[5]HTTPS://www.kali.org/downloads/

vRouter contains the Gateway of the Test private network, which is 10.190.0.1. The IP 10.190.0.5 is new to the Test tenant and represents the DHCP server for the Test private network and uses a lightweight DNS/DHCP application (dnsmasq). Unlike other Test tenant resources, the DHCP server is only connected to the private network. The remaining IP addresses are the private IPs of the Test VMs.

The Test tenant owns two VMs, each with two IP addresses – shown in Tables 5.7 and 5.8. For example, the my_first_instance VM has a private IP address of 10.190.0.6 and an external IP address of 192.168.100.11. Similarly, the KaliLight VM has a private IP of 10.190.0.7 and an external IP of 192.168.100.16. The Test vRouter is connected to the two networks; the private test network 10.190.0.0 with the IP 10.190.0.1 and the external network with the IP 192.168.100.10.

The IPs that represent the VMs indicate that they are indeed VMs. However, in Table 5.8 the resources of different tenants, 192.168.100.12 and 192.168.100.13, show a similar group of ports and therefore indicate a resource similarity, except the deeper information of their nature is still hidden.

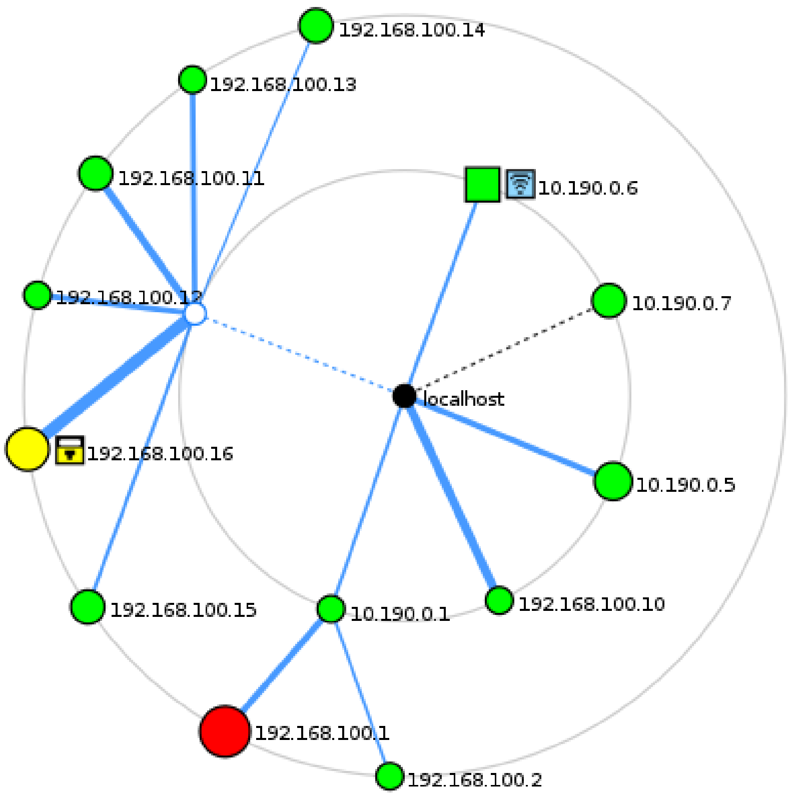


Figure 5.8: Zenmap Internal Pentesting Output: a Graphical Representation of the OpenStack Cloud Active Resource Topology.

The Zenmap outputs of the network distance of the targets are summarised in Tables 5.7 and 5.8

IP	Port(s)	Service	Version
192.168.100.10			
192.168.100.11	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
192.168.100.12	389/UDP	ldap	
	1040/UDP	netarx	
	18156/UDP	unknown	
	20003/UDP	commtact-HTTPS	
	20752/UDP	unknown	
	21476/UDP	unknown	
	21742/UDP	unknown	
	22109/UDP	unknown	
	26720/UDP	unknown	
	31337/UDP	BackOrifice	
	34555/UDP	unknown	
	42056/UDP	unknown	
	43195/UDP	unknown	
	48078/UDP	unknown	
	51554/UDP	unknown	
	53589/UDP	unknown	
	54281/UDP	unknown	
192.168.100.13	389/UDP	ldap	
	772/UDP	cycleserv2	
	999/UDP	applix	
	20710/UDP	unknown	
	20752/UDP	unknown	
	21742/UDP	unknown	
	22109/UDP	unknown	
	37813/UDP	unknown	
	49215/UDP	unknown	
	51554/UDP	unknown	
192.168.100.14	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
192.168.100.15	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
192.168.100.16	22/TCP	SSH	Dropbear SSHD 2012.55 (protocol 2.0)
	53/TCP	TCPwrapped	
	53/UDP	domain?	
	47808/UDP	bacnet	

Table 5.8: Internal Pentesting Zenmap Analysis of Openstack External network 192.168.100.0/24.

and are illustrated in Figure 5.8. All targets are plotted in two layers; the inner layer represents the rogue tenant (Test) where the Zenmap was executed from the KaliLight VM with the IP address 10.190.0.7, while the other IP addresses belong to the other VM, DHCP Server and private network Gateway. The Test vRouter outer port (external network) is presented in the same layer. The remaining targets are presented in the outer layer, which is an indication that Zenmap is missing the traceroute hop, which means it can not guess how far the nodes are, regardless that both 192.168.100.11 and 192.168.100.16 are external IPs of Test's VMs. The dotted lines in Figure 5.8 indicate an unknown connection length. The targets 192.168100.1 and 192.168100.2 are not part of the Cloud structure, however they are within the network range since the connection goes through them to the Internet.

Running Zenmap on the inner side of the cloud provided similar result as the external test; however, the positions and locations of the targets are detected accurately from the inner side rather than the outer side of the cloud. Moreover, the inner Zenmap test confirmed isolation of the tenants' private networks

5.2.2.2 Sparta output

Using a similar setup to the external pentesting, the internal pentesting Sparta output is presented in Table 5.9 with five stages for both the private network targets (10.190.0.0) and the external network targets (192.168.100.0).

Despite the fact that the Test VMS are connected to the two networks, Sparta does not list the same group of ports on the two IPs of the same target. All VMs found during the External Testing show the same pattern as in the Internal Testing. The targets that were ambiguous during the External Test show similar patterns to the Internal Test, as it shows in Table 5.9 stage 4. The targets 192.168.100.10, 192.168.100.12 and 192.168.100.13, are indicated to be vRouters of different tenants, as 192.168.100.10 is known to be the vRouter of the Test Tenant through port 9697.

Sparta revealed that ports on the same target with multi-IP addresses -e.g tenet's vRouters- are connected to the IP and not the target itself. Moreover, Sparta provided information that clarified what has been hidden in the external test through extra ports listed per target.

The OpenStack penetration testing was now complete and a duplicate procedure was applied on CloudStack.

IP address/ Ports per stages	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
10.190.0.1					
10.190.0.5				53	
10.190.0.6			22		
10.190.0.7			22		
192.168.100.10				9697	
192.168.100.11	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.100.12				9697	
192.168.100.13				9697	
192.168.100.14	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.100.15	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.100.16	80, 443	25, 135, 137, 139, 445, 1433, 3306, 5432, 137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		

Table 5.9: Internal Pentesting Sparta Analysis; OpenStack Test Private Network 10.0.0.0/24 and External Network 192.168.100.0/24.

5.3 CloudStack Penetration Testing

Similar to the OpenStack testing, the CloudStack Penetration Testing took two approaches, see Figure 5.9; that is External and Internal testing. The External Penetration testing in CloudStack used the latest version of Kali as, by the time the CloudStack Testbed was built, the new version was already released and there had been issues in installing the previous version in CloudStack. The Kali Linux distribution used was Kali-Linux VMware 64-Bit version 2018.1, that is the kali-linux-2018.1-vm-amd64.ova, "Linux kali 4.14.0-kali3-amd64 #1 SMP Debian 4.14.12-2kali2 (2018-01-08) x86_64 GNU/Linux" from the Kali Linux VMware and VirtualBox Image Download website [6]. The External Kali Linux was installed as a VM in the VMware

[6]https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/

virtualization where the cloud was built, while the Internal Kali Linux was a VM within the Test tenant account using the same Kali Light Image VM as had been modified for the OpenStack testbed.

To match the OpenStack Testbed penetration testing, Zenmap, Sparta and Nessus were used within the External and Internal penetration test through Kali Linux.

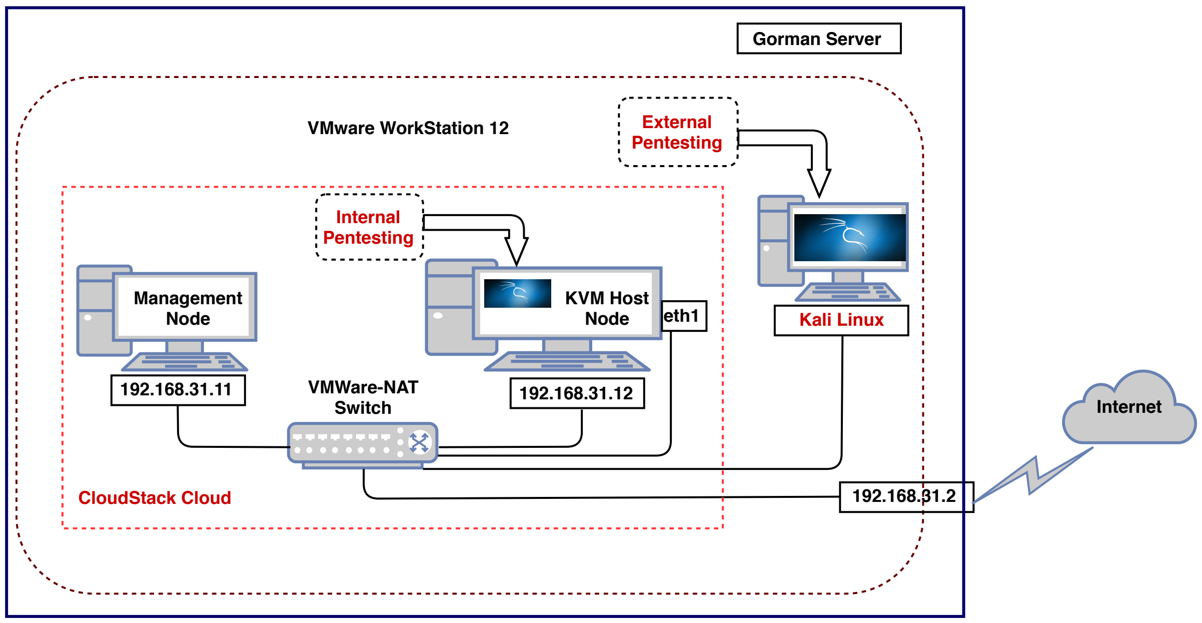


Figure 5.9: CloudStack; Bi-directional Penetration Testing Methodology.

5.3.1 External Pentesting

From the CloudStack Testbed installation process in Chapter 4, we know that there is only one active network for the entire cloud at IP address 192.168.31.0/24. Hence, we used the cloud network as input in the three required scans as discussed below.

5.3.1.1 Zenmap Output

As seen in Table 5.10, the target 192.168.31.11, with the NFS and Apache Tomcat service, indicates it is the management node. Apache Tomcat (8080) is used to run the web management console of CloudStack, while 2049 TCP/UDP lists links to NFS servers, which involves the primary and secondary storage of the cloud. The Host Node (KVM hypervisor Host Node) was identified as the target 192.168.31.12 due to the VNC ports. For a testbed similar to Openstack, it runs more VMs, and additionally, there is no indication of the number of tenants as every target has its own MAC address as found in OpenStack. Comparing the ports in each target,

IP	MAC	Port(s)	Service	Version
192.168.31.11	00:0C:29:C5:69:95 (VMware)	22/TCP	SSH	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.10 (Ubuntu Linux; protocol 2.0)
		111/TCP 111/UDP	rpcbind	2-4 (RPC #100000)
		2049/TCP 2049/UDP	nfs_acl	2-3 (RPC #100227)
		8080/TCP	HTTP	Apache Tomcat/Coyote JSP engine 1.1
		9090/TCP	zeus-admin?	
192.168.31.12	00:0C:29:72:D6:CF (VMware)	22/TCP	SSH	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.10 (Ubuntu Linux; protocol 2.0)
		111/TCP 111/UDP	rpcbind	2-4 (RPC #100000)
		5900/TCP 5901/TCP 5902/TCP 5903/TCP 5904/TCP 5905/TCP 5906/TCP 5907/TCP	VNC	VNC (protocol 3.8)
192.168.31.100	06:E9:A6:00:00:15	80/TCP	HTTP	JBoss Enterprise Application Platform
		443/TCP	HTTPS	
192.168.31.101	06:EE:34:00:00:16	80/TCP	HTTP	Apache HTTPd
		443/TCP	SSL/ HTTP	Apache HTTPd
192.168.31.102	06:00:CA:00:00:17			
192.168.31.103	06:2F:60:00:00:18			
192.168.31.104	06:59:68:00:00:19			
192.168.31.121	06:3D:D6:00:00:01	8001/TCP	HTTP	JBoss Enterprise Application Platform
192.168.31.125	06:D4:3A:00:00:05			
192.168.31.129	06:88:18:00:00:09			

Table 5.10: External Pentesting; Zenmap Analysis of the CloudStack Active Network 192.168.31.0/24

we can group the targets according to the running ports; such as 192.168.31.100-101 revealed similar sets of open ports, but 192.168.31.121 is a stand-alone target as it is the only one with port 8001/TCP. However, the targets 192.168.31.102-104 are similar as there are no listed ports, yet, due to prior knowledge it is known that the address 192.168.31.102 is the Test instance's vRouter, and hence the other two might be vRouters to other unknown tenants.

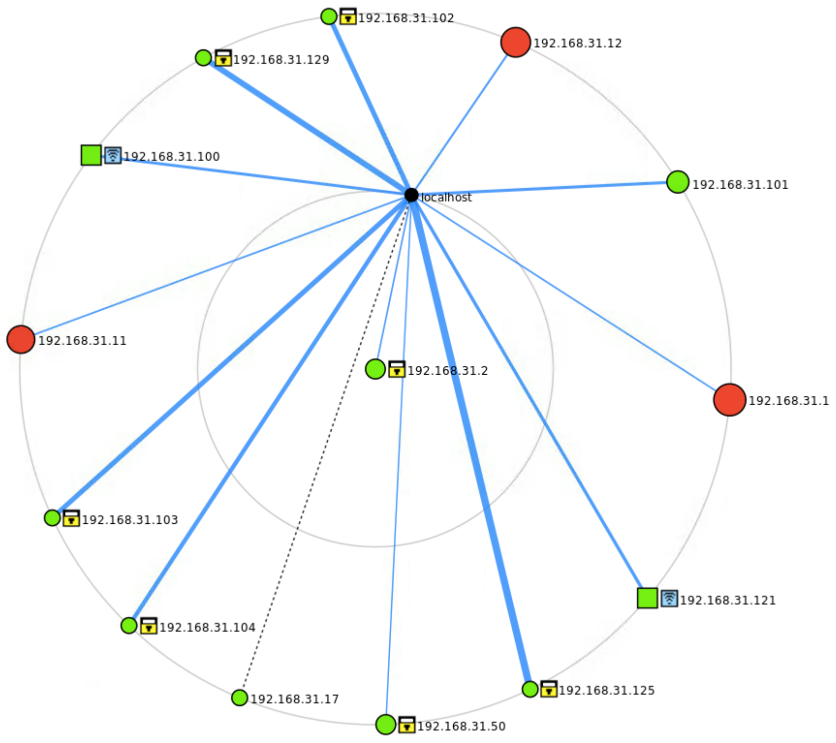


Figure 5.10: Zenmap External Pentesting Output; Graphical Representation of the CloudStack Cloud Active Resource Topology.

Zenmap presented all the targets graphically as shown in Figure 5.10, however, they are all apart by the same network distance, shown by the 2nd ring in the figure, and therefore we can not get a sense of the target locations within the cloud. The only useful information we can gain from here is that the cloud nodes at risk are marked with a Zenmap red colour code, which means each have more than 6 open ports.

Similar to OpenStack, the targets in CloudStack have revealed their functionalities through their open ports. Moreover, all the targets connected to the public network are exposed externally regardless of their owners. Finally, Zenmap externally is unable to plot the target distances and locations accurately.

IP address/ Ports per stages	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
192.168.31.11			22, 111, 2049, 8080	7080, 8250, 9090, 20400	33405,35900, 38462, 45671, 46270, 48158, 60419
192.168.31.12			22, 111	5900, 5901, 5902, 5903, 5904, 5905, 5906, 5907, 16509	38589, 44706
192.168.31.100	80	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.101	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.102	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.103	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.104	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.121	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060	8001	
192.168.31.125	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.129	80, 443	137, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		

Table 5.11: External Pentesting; Sparta Analysis of the CloudStack Active Network 192.168.31.0/24.

5.3.1.2 Sparta output

As attacker, the Test tenant knows about its own resources and details. This is used as a reference to analyse the remaining findings and identify the other tenant resources. For example, the target 192.168.31.102 is a Test Tenant vRouter, while the Test VMs are not visible on the External side, as they can connect to the Internet without the need to be individually assigned with public IP addresses. However, the Host Node presented eight VNC ports, while there are eight public IPs shown in Table 5.11. The VMs presented a pattern of ports similar to the OpenStack Testbed VMs, however the Test VMs are not listed in Table 5.11. Finally, the target 192.168.31.102 is the Test Tenant vRouter, and shows a pattern of VM ports, which means that the vRouter is a VM. It is most likely to be the remaining public IPs, presented in Table 5.11 and belongs to another tenant's vRouters, as all routers require a public IP to connect to the Internet. The only difference is that the target 192.168.31.121 shows an extra port that does not exist in any target, which means that this is a special type of resource.

Unlike OpenStack, CloudStack VMs are not connected to the Public network; hence, determining the number of tenant VMs is a challenging task. Moreover, the attacker tenant can configure many available resources and can use them to compare information collected from other targets to make an objective decision about the nature of the targets, ownership and functionalities.

5.3.1.3 Nessus Output

Nessus had to be installed in Kali as it is not part of the Kali Tool Kit. The application used is the "Debian 6, 7, 8 / Kali Linux 1 AMD64" Nessus-Home, which required an activation code from the original Nessus-home website [7]. By the time the CloudStack Testbed was ready for our Penetration testing plan, a newer Nessus had been provided "Nessus-7.0.1-debian6-amd64.deb", and that was installed in our External Kali Linux. The new version included some new policies that did not exist when the Nessus in OpenStack Testbed was built, such as Wanna Cry Ransomware, Spectre and Meltdown, Shadow Brokers Scan, Intel AMT Security Bypass, Credentialed Patch Audit and Bash Shellshock Detection.

We created 27 Nessus policies because there was only one active network, and learning from the Nessus OpenStack tests, the Advanced policies were limited to four rather than 20. Table 5.12 listed the scan policies and the targets with the exposed vulnerabilities. None of the new policies exposed any vulnerability on the external side.

According to Table 5.12 the target 192.168.31.11 (CloudStack Management Node) is the most critically vulnerable of all the targets from the 192.168.31.0 network. Both the custom Basic Network Scan and all the ports revealed critical security risks due to the NFS service (NFS Exported Share Information Disclosure) and the NFS service is used by CloudStack as a vital element of the System. The remaining vulnerabilities in the same target are relayed to the services used by the cloud such as the NFS, SSH and Web Server. The critical vulnerability revealed by the Advanced Scan is that all plugins enabled were related to Hydra:SSH2 and that was not a Cloud System fault. It means that the Node password as configured was not strong against the Hydra attack. The remaining vulnerabilities were related to the services used by the node such as OpenSSH and Web server. In regards to the Host Node, all the vulnerabilities are due to the weakness of SSH and the critical one was similar to the management node Hydra attack. The most interesting part is that one of the cloud targets, 192.168.31.101, revealed seven

[7]https://www.tenable.com/products/nessus-home [On line; accessed 13 Feb 2018]

S.Q	Scan	Targets	Vulnerabilities			
			Critical	**High**	**Medium**	**Low**
1	Basic Network Scan-All Ports	192.168.31.11	1		6	4
		192.168.31.12			1	2
		92.168.31.100				
		92.168.31.101			7	1
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				
2	Basic Network Scan-custom	192.168.31.11	1		3	3
		192.168.31.12			1	2
		92.168.31.100				
		92.168.31.101			7	1
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				
3	Advanced Scan-all plugins enabled	192.168.31.11	1	4	7	4
		192.168.31.12	1	4	6	3
		92.168.31.100				
		92.168.31.101			8	2
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				

Table 5.12: CloudStack External Nessus Scan; Vulnerabilities of 192.168.31.0/24 Network Targets.

medium vulnerabilities and a single low vulnerability related to the SSL protocol. Nessus did not find any vulnerabilities related to the remaining targets.

Nessus confirmed the knowledge learned from Zenmap and Sparta without adding additional information to enrich our cloud infrastructure knowledge. However, Nessus provided a critical point in terms of the system vulnerabilities. Vulnerabilities might not be directly involved in remapping the Cloud VN infrastructure components. However, the existence of vulnerabilities leads attackers to using those risks and performing harmful attacks against the cloud targets, some of which are Cloud VN components, and against the Cloud Nodes, which contain the Cloud Network Infrastructure components and isolation configuration.

5.3.2 Internal Pentesting

Although we used the same image of Kali from the OpenStack testbed, it was not as straightforward a task to upload the image to the cloud as in OpenStack. Instead, we created a simple webserver to put the image into the cloud. The details of this technical requirement is available in §5.5.2.

Moreover, the problem of the Nessus application running in the internal tenant's Kali Linux, which we faced in Openstack, also occurred in CloudStack. However, in this case, we could run Nessus as expected. The details of the process are stated in §5.5.2.

The rest of this section presents the outputs of the three pentesting tools run from Kali within CloudStack and an analysis of the most important findings. Moreover, the Internal Pentesting demonstrated additional information that was not externally visible to the Test tenant's Private Network (192.168.1.0/24).

5.3.2.1 Zenmap output

The internal testing used the same nmap statements as the previous nmap testing. However, it was done twice, once for each network; the public Network and the Test tenant's Private Network, as shown in Table B.1 in Appendix B.

The table demonstrated the internal pentesting of the targets from the 192.168.31.0 network. However, it exactly matched Table 5.10, the External pentest, above and hence there is no further information to be gained.

Table 5.13 consists of new information from the previous data collection and represents the Test tenant's resources. The target 192.168.1.1 is the Test Private Network Gateway on the Test vRouter, 192.168.1.73 is the Test VM, and 192.168.1.62 is the KaliLight VM. Although both

IP	MAC	Port(s)	Service	Version
192.168.1.1	02:00:52:86:00:02	53/TCP 53/UDP	domain	dnsmasq 2.62
		80/TCP	HTTP	Apache HTTPD
		8080/TCP	HTTP-proxy?	
192.168.1.73	2:00:1C:0F:00:01	22/TCP	SSH	OpenSSH 4.3 (protocol 2.0)
192.168.1.62		68/UDP	DHCPC	

Table 5.13: Internal Pentesting Zenmap; Analysis of the CloudStack Test Private Network 192.168.1.0/24.

of the Test VMs are connected to the Internet, they are not visible from outside the cloud, and hence it is assumed that none of the other tenant VMs are visible. Therefore, the list of targets in Table B.1 belong to resources other than VMs.

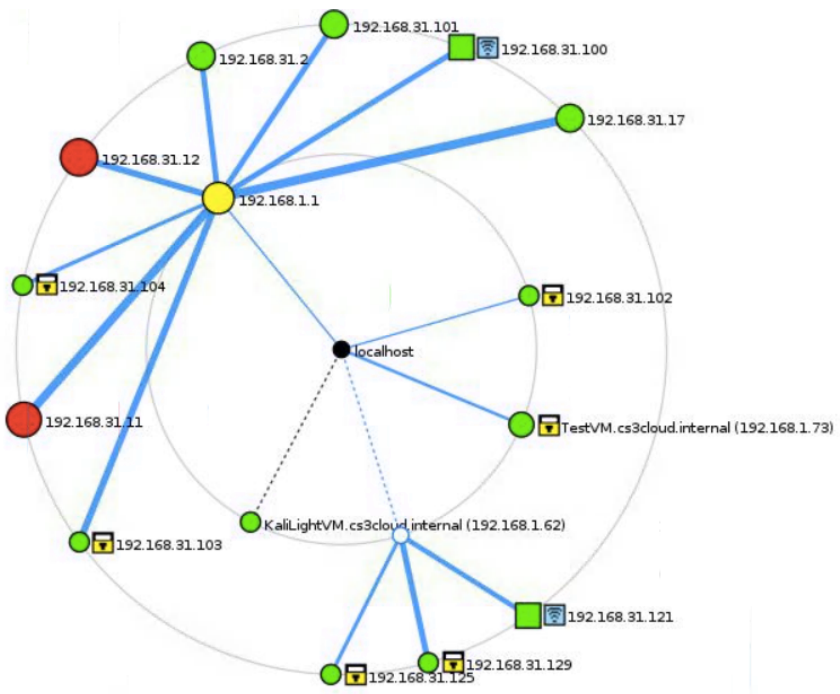


Figure 5.11: Zenmap Internal Pentesting Output; Graphical Representation of CloudStack Cloud Active Resource Topology.

Figure 5.11 provides more detailed information on the CloudStack resources graphical representation using each target network distance from the Zenmap output within the KaliLight VM (local host). Similar to Figure 5.10, Figure 5.11 shows that the Management and Host Nodes are the most risky due to the number of open ports. They are followed by the Test vRouter represented by the Test private network gateway which contains between 3 and 6 open ports. Figure 5.11 shows that the cloud targets are located within two layers, where the inner layer is inner and private to the cloud, and only shows the Test tenant resources. However, due to the cloud resources allocation, all the tenant resources are privately located at the same level. Moreover the vRouters connect the Tenant's inner side network with the public. For example, the Test vRouter's inner side network is connected with the router IP 192.168.1.1, and the public side is connected with the IP 192.168.31.102; although both IP addresses belongs to the same target but they presented as different entities; and both are placed on the same layer. Resources are graphically separated due to network distance differences; 192.168.31.121, 192.168.31.125 and 192.168.31.129 distances are unknown compared to the remaining resources, which are only at one hop distance. Therefore, we suggest that they are grouped as a similar kind of resource. The remaining resources, after removing the management and host nodes, were categorized into different groups with respect to their open ports. The 192.168.31.100 and 192.168.31.101 nodes contain similar open port patterns and 192.168.31.104 and 192.168.31.103 have no open ports, as with the test vRouter. As a result, the closest assumption is that 192.168.31.102-104 are tenant vRouters.

The internal Zenmap test matched the result from external Zenmap, however, due to the extra visibility in the cloud internal test, more knowledge is gained from exposing the private tenant network. Although the private network belongs to the attacker, the Zenmap test can indicate to the attacker how the other tenants isolated networks are structured, including behaviour, configuration and nature of the network components.

5.3.2.2 Sparta output

Table B.2 in Appendix B demonstrates that the Sparta results have no additional information other than what was already known from the External testing, except confirming that all the targets found are forms of VMs. Moreover, the Test Tenant resources found by Sparta have provided no additional information to what was found in Zenmap.

5.3.2.3 Nessus Output

The Nessus issue in the Test Tenant was solved by using the VM KaliLight. The details of the solution are listed in the CloudStack problems and implications section in §5.5.2.

The inner side of the cloud handled two network ranges, one is related to the Test Tenant private network and the other range is the public range that we tested in the external side. As the Test Tenant VMs are located within the cloud installing Kali in the Test VM is the closest location to the targets.

Table 5.14 shows the vulnerabilities collected from the public network targets. The Basic and

S.Q	**Scan**	**Targets**	**Vulnerabilities**			
			Critical	**High**	**Medium**	**Low**
1	Basic Network Scan- All Ports 192.168.31.0	192.168.31.11	1		3	3
		192.168.31.12			1	2
		92.168.31.100				
		92.168.31.101			7	1
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				
2	Basic Network- custom 192.168.31.0	192.168.31.11	1		3	3
		192.168.31.12			1	2
		92.168.31.100				
		92.168.31.101			7	1
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				
3	Web Application Test- All Ports 192.168.31.0	192.168.31.11		1	1	1
		192.168.31.12				
		92.168.31.100				
		92.168.31.101				
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				
4	Web Application Test- custom 192.168.31.0	192.168.31.11		1	1	1
		192.168.31.12				
		92.168.31.100				
		92.168.31.101				
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				
5	Advanced Scan- full scan plugins ON 192.168.31.0	192.168.31.11	1	4	7	4
		192.168.31.12	1	4	6	3
		92.168.31.100				
		92.168.31.101			8	2
		192.168.31.102				
		192.168.31.103				
		192.168.31.104				
		192.168.31.121				
		192.168.31.125				
		192.168.31.129				

Table 5.14: CloudStack Internal Testing; the Number of Nessus Scan Vulnerabilities in the 192.168.31.0/24 Public Network Target.

Advanced network scans on the 192.168.31.11 target showed no difference from what has already been found in the External Nessus scan shown in Table 5.12. However, an additional result from the internal pentesting was revealed using the Web Application Test scans, which was not detected by the External Nessus scan. All vulnerabilities revealed by the web application test scan were related to the web server on the Management node. Both 192.168.31.12 and 192.168.31.101 revealed the same vulnerabilities as in the External Nessus scans shown in Table 5.12.

S.Q	Scan	Targets	Vulnerabilities			
			Critical	High	Medium	Low
1	Basic Network Scan- All Ports 192.168. 1.0	192.168.1.1			1	1
		192.168.1.73			1	2
		192.168.1.62			2	
2	Basic Network- custom 192.168.1.0	192.168.1.1			1	1
		192.168.1.73			1	2
		192.168.1.62				
3	Spectre and Meltdown- custom 192.168.1.0	192.168.1.1				
		192.168.1.73	1			
		192.168.1.62				
4	Credentialed Patch Audit- All ports 192.168.1.0	192.168.1.1				
		192.168.1.73	15	90	157	9
		192.168.1.62				
5	Credentialed Patch Audit- custom 192.168.1.0	192.168.1.1				
		192.168.1.73	15	90	157	9
		192.168.1.62				
6	Spectre and Meltdown- through 192.168.1.0	192.168.1.1				
		192.168.1.73	1			
		192.168.1.62				
7	Advanced Scan- full scan plugins ON 192.168.1.0	192.168.1.1	1		5	1
		192.168.1.73		9	11	5
		192.168.1.62				

Table 5.15: CloudStack Internal Nessus Scan; the Number of Vulnerabilities in the 192.168.1.0 Internal Network Targets.

Table 5.15 lists the Nessus scan results from within KaliLight from inside the cloud. Seven scans exposed the Test Tenant vulnerabilities and four of those scans are from the new scans mentioned in §5.3.1.3. The Internal Nessus scan for the Test Tenant cloud components was run to make sure that the cloud provisioning service is vulnerability free. Another reason was that due to the cloud service provisioning, what is offered to one tenant is most likely what is offered to other tenants. Hence, if a vulnerability exists in a tenant, it will most likely occur in other tenant components.

It is not a surprise for a VM to reveal more vulnerabilities than other components. However, for the cloud platform to provide a test image, TestVM 192.168.1.73, with a large number of vulnerabilities, especially in scan numbers 4 and 5 in Table 5.15 above was a surprise. The attacking VM, KaliLIght, was also revealed as the least security risk. Also, the Test Tenant vRouter provisioned by the cloud revealed some critical and medium vulnerabilities related to the DNS service and dnsmasq application used by the router to implement the DNS. The lower level vulnerabilities are related to the DHCP server detection.

The newly learned information from the internal Nessus test is that the VM images provided by the cloud platform are vulnerable. Other tenants should not use publicly provided VM images even if they are a standard version provided by a cloud service. Additionally, an essential cloud system revealed vulnerability is a serious risk as although it is not directly involved in cloud networking, it connects all tenants facilities including their VNs. A more surprising security risk is that the vRouter of the tenants are in fact vulnerable VMs.

5.4 Summary of Findings

After a long process of testing, collecting and analyzing data from the two cloud testbeds penetration testing, we summarised all the findings in Tables 5.16 and 5.17. The table compares the OpenStack and CloudStack testbed's External and Internal Penetration Tests, and aummarised the findings of the analyzed data from the Zenmap, Sparta and Nessus tools.

5.5 Technical Difficulties

This section lists the most important technical complications encountered during the bi-directional Penetration Testing of both the OpenStack and CloudStack Testbeds.

Kali Linux was used as penetration testing toolkit for this research to leak Virtual Network information. As mentioned above, the source of Kali Linux was from the original provider by the Offensive Security organization which offers Kali images in different formats. We selected two formats; the Kali image built for VBox used by the OpenStack Testbed and the VMWare image used by CloudStack. The pentesting with Kali did not encounter any difficulties on the External Penetration Testing for both testbeds, however, it was different when the Kali Linux images were intended to be used from within the cloud. The following sections lists the technical difficulties encountered by each testbed.

5.5.1 OpenStack

The following list of technical complications were encountered while installing Kali Linux as a cloud VM in OpenStack.

Cloud Testbed	Penetration Test	Testing Tools		
		Zenmap	Sparta	Nessus
OpenStack	External	Cloud nodes are exposed using the open port in each node, e.g. HTTP Port runs in Controller node.	Exposed similar port list as Zenmap and more.	Confirmed Zenmap and Sparta's findings
		VNC port reveals the cloud node as a compute node, and the number of VNC ports reveals the number of VMs running in the cloud.	Confirmed VNC port security vulnerability exposed by Zenmap	Lists the cloud vulnerabilities per to its network ranges (Management network and External Network)
		Vulnerability warning against insecure VNC connections.	Gave a view of the VMs and roughly distinguished each tenant VM using VMs snapshots	Scored the cloud vulnerabilities according to its harmfulness (Critical, high, medium and low).
		Grouped several IP addresses under a single MAC address, and revealed the number of Tenants in the cloud.	Used VNC vulnerability to leak vital VM information such as IP address, MAC address, Gateway , hostnames, etc.	Controller node is more vulnerable than the Compute node in terms of the number of vulnerabilities exposed.
		Graphic representation: controller node is at more risk to attacks since it contained more opened ports.		The tenants VMs are the most vulnerable components in the cloud among the cloud components previsioned.
	Internal	Used the knowledge from test tenant internal components (e.g VMs) and the discovered port to compare and distinguish other tenants components	If a target assigned more than one IP address the scans revealed different sets of ports per IP address in that target. Hence, a target could be at risk caused by different IP communication.	Due to technical issues data was not collected.
		Graphic representation: Internal testing graph showed a better view of the targets and its position/distance from the attackers than the external testing.		

Table 5.16: Finding Comparison of OpenStack Testbed Penetration Testing.

Cloud Testbed	**Penetration Test**	**Testing Tools**		
		Zenmap	**Sparta**	**Nessus**
CloudStack	External	Cloud nodes are distinguishable from the open port discovered (e.g, 8080 tomcat webserver in management Node)	If the number of open VNC ports is more that the number of IP addresses discovered, there are VMs not discoverable through IP addresses.	The Management node is the most critically vulnerable among the cloud nodes.
		Some services are preparatory to CloudStack (such as NFS). This tells what nodes are running in the cloud and what their functionalities are.	Test tenant components, vRouter open ports are similar to the ones discovered in VMs, hence all the routers provisioned by the cloud are VMs.	CloudStack system VMs(cloud functionality components) suffer from 7 medium and 1 low vulnerabilities.
		Nodes runing in The cloud is found from the open ports (e.g VNC runs in one node, hence the cloud has one Host node)	Unlike OpenStack, VNC showed no security vulnerability	The CloudStack testbed showed higher level vulnerabilities than the OpenStack testbed.
		Graph: cloud nodes presented a similar risk level-red- (see figure 5.10). The cloud components-distance to the attacker is not indicated.		Although CloudStack showed higher level vulnerabilities than OpenStack, it leaked less information than OpenStack.
	Internal	VMs are not exposed externally, because they connect to the Internet without the need to have an external IP address	Confirmation the findings of Zenmap. There is no other knowledge gain from this test.	VMs (cloud provided VMs) are the most vulnerable component in the cloud
		Results found using the internal test matched those found externally		Using the Test tenant private IP address, new vulnerabilities were discovered in the vRouter (a Cloud provisioned component) DNS and DHCP services.
		Graphically, the internal test represented targets in much clearer view in terms of their distance from the attacker		

Table 5.17: Finding Comparison of CloudStack Testbed Penetration Testing.

1. To match the Internal pentesting with the External testing, the logical move was to use the exact same image in both tests. However, the Offensive Security Organization provides ISO images of Kali for fresh installation or configured images for VBox. VMware and Hyper-V. Although the OpenStack Image uploading form in the Horizon dashboard gave a list of acceptable image formats, none of them was a close fit for the Kali images due to the cloud hypervisor used (KVM/QEMU).

2. Another option we tried was using the Kali ISO image and installing a fresh version from within the cloud. This option failed due to the inexistence of a VM disk. Also defining the VM flavour in OpenStack does not give the Disk specification but an actual disk must be provisioned by another service called Cinder, which was not configured in our OpenStack Testbed.

3. We used an external Ubuntu16.04 LTS Desktop device with a QEMU/KVM: qemu version 2.5.0 (Debian 1:2.5+dfsg-5ubuntu10.6) and installed a fresh Kali image from the Kali ISO image and then saved it with the qcow2 format.

4. After transporting the new Kali image to the Compute Node we used the glance command to upload the Kali Linux image to the OpenStack cloud. We then created a Kali VM under the Test Tenant.

5. Another technical complication occurred during the VM execution; the system froze up to *"Started Update UTMP about System Run level Changes"*. After an investigation and through comparing the XML files of the running VM, we found that the cloud hypervisor configured the VMS disk and network with the KVM configuration Virtio model type. With this new information, and making the adjustment, the Kali VM was successfully run although its performance was slow.

6. Going to a lighter version, without a graphic interface, was the next move, except we failed to run Kali with the console interface alone.

7. The final trial used the Kali Light ISO from Offensive Security website, which has fewer tools installed. In the QEMU/KVM virtualization environment running in the Ubuntu desktop, as mentioned above, we installed Kali Light Linux with the Virtio model as required, installing only Zenmap, Sparta, Nessus, Ettercap and Wireshark. Following steps 3 and 4 we launched Kali Light as a VM in the Test Tenant and ran the Internal penetration test as in §5.2.2 and §5.3.2.

8. An issue that we failed to solve was with running Nessus in Kali Light in the OpenStack

External Network, which was configured – per to the installation guide- to be reachable outside the cloud but not from the Internet. Different solutions would not solve the Nessus issue, hence, we executed the OpenStack Internal Penetration Testing without Nessus.

5.5.2 CloudStack

The following list of technical complications were encountered while installing Kali Linux as a cloud VM in CloudStack.

1. The same modified Kali Light used in the OpenStack Testbed was also used in CloudStack and carried the same issue of reachability to the Internet. CloudStack refused to operate without a real Internet connection to its System VMS and other cloud inner components. Nessus was re-installed with a new activation code and was updated with the latest plugins and new policies.

2. The CloudStack testbed differs from the OpenStack testbed in that the OpenStack instances are configured to be accessed remotely via SSH because this allows those instances to reach the Internet. However, Cloudstack configures the vRouter to perform the traffic forwarding without giving the instances a public address. This does not allow the Tenant to access their VMs via SSH. Remote access via SSH can be provided to CloudStack instances in the following ways:

   a) Using an SSH key pair [168], the VMs can run. However, there is no option to edit the VM configuration with the key pair unless the VM is recreated.

   b) Configuring port forwarding and the firewall[169]. However, the testbed firewall is disabled and controls the traffic security through the ACL. Configuring the SSH via Port Forwarding was not successful even with editing the ACL to allow SSH.

   c) Enabling the Static NAT[170], which is the method applied in OpenStack. However, testing the connectivity and the SSH [171] was not successful either.

5.6 Conclusion

The focus of this chapter was to achieve the fourth research objective; to collect leaked information from running Penetration Tests (Internally and Externally) on the two cloud testbeds and to analyse that information to determine the components and configuration of the tenant VNs.

We selected one of the cloud tenants (Test tenant) to be a rogue tenant that would launch the penetration tests using kali Linux against the testbeds (CloudStack and Openstack). Penetration tests were performed as bi-directional, that is both externally and internally, for both testbeds. The outcome of the penetration testing revealed vulnerabilities as well as cloud design weaknesses. In section 5.4 a summary comparison of the two testbed penetration testing result was listed.

OpenStack and CloudStack Network Infrastructure include all the Tenant VN components, configuration and isolation mechanisms. Both infrastructures use SDN and NFV to build up the Cloud networking and tenant network service provisioning. Although both tested clouds adopted similar technologies, the platforms implemented them differently. Chapters 3 and 4 revealed both Clouds' network infrastructure and located the tenant's network resources.

This chapter aimed to use the attacker tenant view to leak cloud infrastructure information through Penetration Testing and expose other tenant's network components. Penetration Testing tools were successful in obtaining knowledge about the Cloud Network Infrastructure, especially from NFV technology and less from the SDN technology. Moreover, without an additional third party security layer and, depending only on the cloud platform to provide the required security for its network infrastructure, the cloud vulnerabilities found with our testing made the cloud an easy target for even non-expert attackers.

Finally, our testing revealed a major Cloud Network Infrastructure mis-design and configuration, where both clouds connected their tenants through a shared-public network range and exit interface, which contributed greatly to our Penetration Testing security exposure and data leakage. Chapter 6 expands the discussions on the findings of Penetration Testing, the security impacts and potential mitigations and solutions.

CHAPTER SIX

RESULTS AND DISCUSSION

This chapter summarises and discusses the findings from the experiments described in the previous three chapters.

The outline of the objectives in terms of the cloud platform used, the resources needed and the VN component methodology in previous chapters will be revisited. The decision to reject or accept the research hypothesis (of §1.5) will also be discussed in this chapter. Further, we discuss the method of identifying leaked data and their characteristics, and mitigation strategies to improve Cloud Virtual Network Isolation security.

6.1 IaaS Cloud Platform Testbeds Discussion

The following list details the results of the DeepDive experiments, the two testbeds and the penetration tests. It is followed by a further analysis of each testbed individually.

6.1.1 Cloud Testbed Setup

The following lists the important observations gained during the Cloud testbed setups:

1. Using the guidelines from both cloud organizational documents, and comparing different examples of setting up cloud models for both OpenStack and CloudStack led to the creation of Testbeds used in this research.

2. Both clouds have their own distinguishing setup procedures. However, both have a similar network setup throughout the cloud infrastructure by using bridges, management network, public and guest networks.

3. Cloud nodes were reasonably easy to install and set up, because the setup guide installed the cloud web service within the Controller or Management Node. Hence, tenants would know the IP address of the Controller and could easily scan the whole internal network to discover any other active IP addresses. They could run security and vulnerability assessments.

4. It is easy to distinguish between the Cloud Nodes from the ports listed in each node, which represent the active services. The entire cloud infrastructure and the tenant components, including the VNs, are located in the testbed nodes.

5. Both OpenStack and CloudStack guidelines dictate a network range through the public IP addresses, which all tenants share. Although Openstack uses a different range for management from the public network, CloudStack uses the same network range for both Management and Public network.

6.1.2 The DeepDive Methodology

The following lists the important research findings from the DeepDive stage:

1. The cloud network infrastructure gained from the DeepDive provided a clear indication about the information needed to match details from the pentesting results. Hence, the result of the DeepDive was not only translated into a diagram as a cloud infrastructure map for easier understanding, this map also became a guiding reference for the depth of cloud network infrastructure data leakage.

2. From the DeepDive we understood which nodes could be a Host, a tenant VM or a tenant virtual network component. However, in Chapter 5, Black Box testing (part of the Penetration Test by the attacker or Test Tenant) was used to collect information to get closer to the original data gained by the DeepDive .

6.1.3 Penetration Testing

The following list is the research findings gained during the Penetration testing stage:

1. Using one of the Tenants as an attacker to enhance the knowledge of other tenants through penetration testing was applied throughout the entire cloud. The Test tenant already knows

their own resources and could analyse the remaining information to draw a conclusion as to what resources are available in the cloud.

2. VMs are connected to a private tenant network and hence are visible to the tenant. However, once they are associated with public IPs, they are visible to every tenant.

3. vRouters are designed to connect the private side of the tenant and the public side of the cloud. They are not visible to the tenants from inside the cloud, however they are visible using their public IP address.

4. In the penetration test the collected data from Zenmap, Sparta, Nessus are analysed. The majority of this data includes IP addresses, MAC addresses and port numbers.

5. Knowing the detailed data from the DeepDive including IP addresses, MAC addresses, port names, services running and connections, gave an indication about the meaning of the information collected using pentesting. The pentesting leaked information such as open ports and associated services, IP and physical (MAC) addresses, and system information, such as the OS used.

6. Both penetration test directions (external and internal tests) show almost exactly the same leaked information and the same vulnerabilities. Data leaked – of the two testbeds built in Chapter 3 and 4 – was found both direction and some were surface information (e.g: port discover, tenants components linked to public address range, security vulnerabilities, etc).

7. Some internal hidden cloud components revealed their information and showed some weaknesses, such as virtual routers revealing open ports that listened to a networking service, which would indicate a network component.

8. Having a Test tenant as a legitimate tenant who can create all available types of VMs from the images available, all network options and components a cloud provides, allows the viewing and study of all the relevant network data. A comparison of the pentesting results from the internal and the external tests for cloud resources ensures information collected from other components leads to distinguishing, or estimating the nature of the target, VM or network component. Even if the current information does not indicate which tenants they belong to, the value of the test has increased.

9. Resource categories and provisioning are available to all the tenants; hence a rogue tenant can create many resources and adapt the configuration. Using our methodology to compare

their new resources with other resources revealed by pentesting, a rogue tenant can discover co-existing resources that belong to other tenants.

10. The Zenmap network distance diagram was a useful aid to group the cloud resources and make an estimate of the resource nature based on groupings.

11. Both cloud platforms used Virtual Network Computing (VNC) as a web remote desktop for their VMs, yet the results from each testbed were different.

6.1.4 Mitigation and Recommendation Guidelines

The following lists the important observations related to recommendations for the mitigation of vulnerabilities in both OpenStack and CloudStack:

1. Cloud IaaS are built on the integration of hardware and software that are configured to provide the infrastructure design. Our methods discovered a major mis-configuration that led to infrastructure mis-design, and concluded that regardless of the effort to securely isolate the cloud VN resources within the cloud, one mis-configuration could damage the entire isolation effort.

2. Following the cloud installation guide from the cloud platform developer does not cover the security aspects of the cloud platform; hence the cloud nodes must be investigated for any security risks and made secure against all vulnerabilities discovered.

3. The VMs are the most vulnerable part of the tenant resources and therefore any VM must be tested and secured before they are configured to be accessed publicly.

4. VMs are not part of the VNs, however, VMs are a vital tenant component connected to the VNs. Attacks against the VNs can be launched through the VMs vulnerabilities.

5. Vulnerabilities are not necessarily VN isolation related, but they allow potential attacks on the cloud and negatively impact tenant isolation, and consequently, the VN isolation.

6.1.5 OpenStack Results Discussion

While the above lists indicate the findings that apply to both clouds, the following points note the OpenStack findings from pentesting. These are followed by a graphical representation, similar to the DeepDive diagram in Figure 3.7, but only showing the penetration test findings:

1. OpenStack designed the VM connections between the tenant resources using private IP and the connections to the Internet through Public IP addresses. Therefore, all the VMs

associated with Public IP addresses are exposed to all tenants as they all share the public network.

2. Tenant vRouters connect the tenant's resources on two sides, the private and the public side. All the public IP addresses allocated for the tenants resources are listed under the public Gateway on the vRouter, therefore they all share the same MAC address, as shown in Table 5.2. It is due to this design that the penetration testing revealed the number of tenants resident in the cloud. Further, because of the cloud consistency in design, the Test tenant shared a MAC address with all their resources aa did the other tenants.

3. It is fairly easy to classify the tenant's resources, as the penetration testing tools listed all possible ports, even if filtered, and the pattern of ports made the VMs distinguished from the other resources, especially as each VM is allocated a public IP address.

4. All the vRouters are configured as Linux containers called namespaces, as shown in Table 316. This configuration is the reason behind not being able to detect any running port. Describing their nature was therefore difficult, however, because of cloud resource consistency. The test tenant vRouter was similar in set up to other tenant vRouters and decisions as to identification were almost correct according to the DeepDive .

5. Although there was cloud consistency in the implementation of the isolation method (VLAN and VXLAN) on all tenants VNs, neither the penetration testing nor the tenant account revealed the network isolation method used by the cloud. Hence the attacker cannot predict the isolation mechanism of its own VNs and those of the other tenants.

6. The Compute Node listed the VNC ports each related to a VM running in the cloud and therefore anyone could detect the number of VMs running in the Cloud. A major security breach was when OpenStack did not configure the VNC security properly (no authentication was demanded); hence the Sparta-VNCViewer used the VNC open ports and gave access to the VMs, each with its corresponding VNC port. Without a login to the VM, the VNCViewer exposed the following information illustrated by Table 5.4 and Figure 5.6, respectively: the VM Hostname (the Name within the VM OS) and the VM name given by the cloud system (an automated name given by the cloud, for example, instance-00000003), which is the name of the VM XML file that contains all the VM configurations including the connections to the Linux bridges. The Linux bridges are shown in Table 3.10. The OpenStack hypervisor uses the default path to store these XML files as mentioned in §3.4.5.2. Knowing the name of the files, attackers can use the SSH Server CBC Mode Ciphers Enabled [1] ; vulnerability - see

[1]This vulnerability was discover in one of the OpenStak targets and it enables attackers to recover plaintext

Table B.3 - to steal those files and expose more of the cloud infrastructure.

7. All tenants used the sample image provided by the cloud platform, which happened to display the username and password on the login prompt. All the tenants have the option to upload any (potentially buggy) image and set it as public for all the tenants to use, which is an opportunity for the attacker to use a VNC security flaw to access other tenants and use it as an entry point to the other tenants' resources, such as their VN components.

8. The Nessus result discussed in §5.2.1.3, revealed a higher risk on the Controller node with a maximum vulnerability levels of 7 High and 11 Medium. The Compute node had a maximum of 5 High vulnerabilities and 7 Medium. Most of the High vulnerabilities on the Compute node are either for OpenSSH or VNC, while the Controller node was mostly vulnerable due to the OpenSSH and the Apache webserver.

9. In the targets with Public IP, the most vulnerable were those representing the VMs each with 1 critical, 1 medium, and 2 low vulnerabilities for SSH. The other Public IPs represented the vRouters and showed no vulnerabilities because an OS does not operate them as the VMs do; instead they are namespaces (Linux containers) which are more robust than the VMs.

10. Despite Nessus failing to run in the internal pentesting, the outputs from the bi-directional testing are similar in the two testbeds. We believe that any lost information from an internal Nessus test was not critical.

11. The internal Zenmap graphical representation of the scanned targets, shown in Figure 5.8, was clearer than the external representation –Figure 5.3.

12. The ports listed under the private IP sometimes differ from those listed under the public IP of the same target. However, all the ports do belong to the same target.

The following is a reproduction of the DeepDive diagram (Figure 3.7) listing showing what was learned from the external and internal penetration testing.

Figure 6.1 plots all the knowledge gained about the OpenStack cloud Infrastructure from the leakage of information through penetration testing. Although we did not achieve leaking core cloud networking information, such as Isolation mechanisms, OVS and bridges ports, due to the cloud designs for linking to the public network, we successfully revealed partial cloud infrastructure information. This included the number of tenants and their components, the logical and physical addresses of these components, and identified a major tenant's VN components, their vRouter and DHCP server.

messages from cipertext.

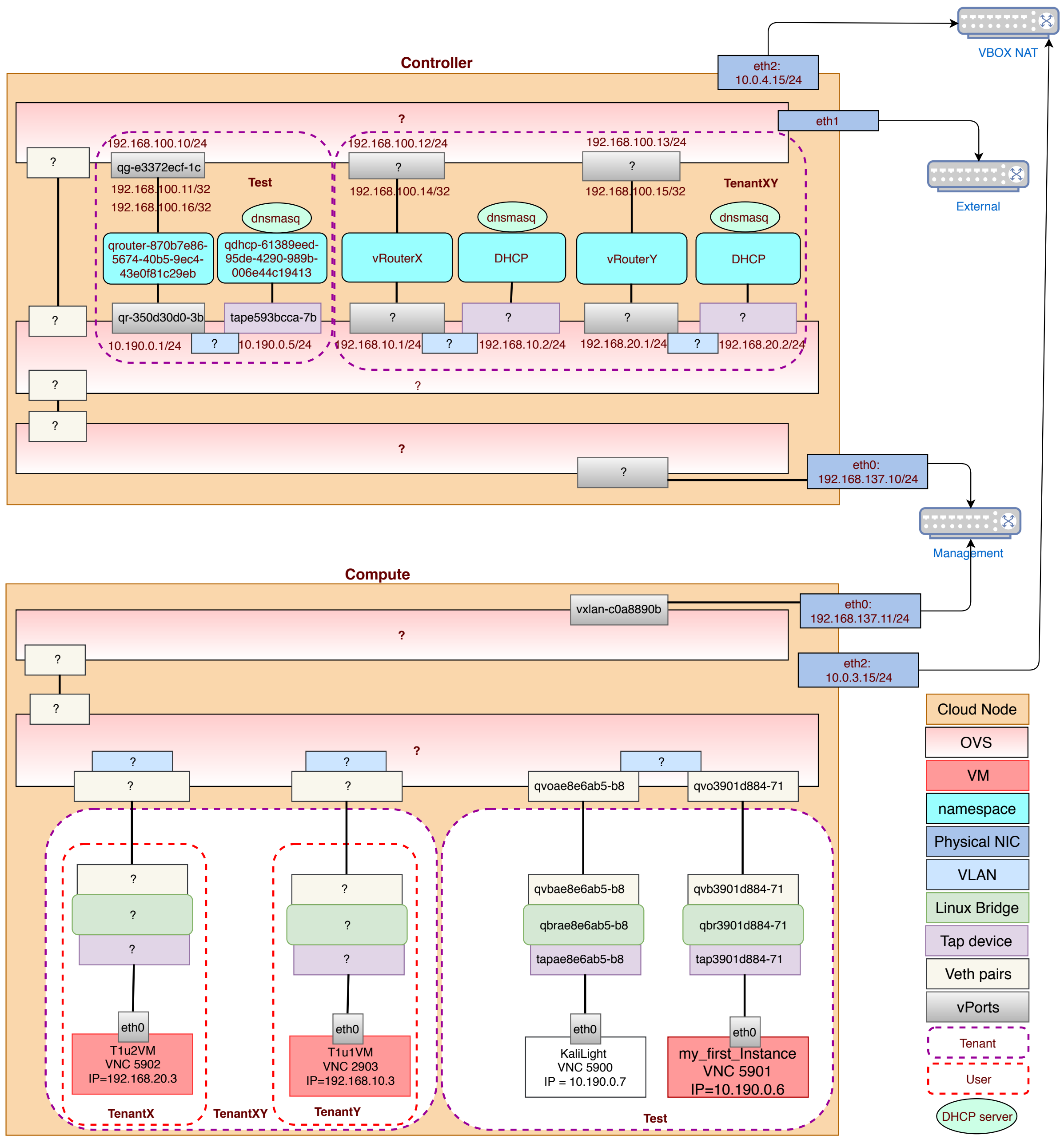


Figure 6.1: OpenStack Infrastructure: Leaked Information from Penetration Testing.

Elements in Figure 6.1 are colour coded according to the highest level of vulnerabilities discovered by Nessus. The nodes (Controller and Compute) represent a High vulnerability coloured by orange, while the VMs are exposed as a critical (red colour code) vulnerability.

Penetration testing leaked information about all tenant public IP and MAC addresses, while ports identified the vRouters. However, any addresses and components connected to the tenant private networks such as VMs details, IP and MAC addresses, and DHCP servers were leaked due to the vulnerabilities of the VNC service in the cloud.

Finally, all the ports (within Linux and OVS bridges) in the infrastructure connecting the Test tenant components are discovered through the Test account in the Openstack web interface. Additionally, all the interface codes are listed with the same codes shown in Figure 6.1. None of the other tenants' ports were learned from this current methodology, as that information is granted to the account owners only.

6.1.6 CloudStack Results Discussion

The following points outline the issues and findings of the CloudStack pentesting. This is followed by a graphical representation, showing the penetration test results:

6.1.6.1 CloudStack Testbed Setup

The following list relates to observations on the CloudStack Testbed specific to the setup stage

1. Following the official cloud installation guidelines, the testbed executed the Management Node, tenants components and its public communication used a single network range. Therefore, all the elements that were granted a public IP address were visible from both directions of the penetration test, such as Nodes, System VMs and Tenant provisioned components.

2. Although OpenStack separated the components that assigned the management IPs and public IPs, unlike CloudStack, this did not make OpenStack components more isolated and the final result was the same for both testbeds in that the tenant components connected to public IP address were visible to everyone.

3. The graphical representation of the cloud components (system and tenant VMs) from the internal Zenmap test, shown in Figure 5.11, was more descriptive than the external Zenmap test illustrated by Figure 5.10.

4. Unlike OpenStack, CloudStack was configured to be connected to the Internet throughout the cloud setup and operation. Hence the VMs were connected to the Internet without the need to be assigned to public IP addresses. However, without IP addresses they are not accessible via SSH.

5. Each system VM was assigned more than one public IP address – see Figure 4.6- and were grouped according to their open ports, which were found by Zenmap. However, the pentesting was not successful in defining what the ports were used for.

6. As mentioned in the previous chapter, to match OpenStack in configuration, the configuration of CloudStack VMs with SSH failed. However, during the configuration the public IP address of VMs were assigned successfully, although they did not connect to anywhere except to the VM itself and a vRouter, which was not sufficient to enable SSH. Moreover, during the troubleshooting it was found that the public IP address assigned to the VM was in fact allocated the same vRouter port as the private network Gateway and was sharing the MAC. This is very different from OpenStack where the port that connected the router to the public network did not connect to the private network and the VMs Public IP address shared the MAC address of the port on the public network side. However, SSH worked under this configuration in OpenStack. If the CloudStack SSH configuration was a technical bug and was configured to be like OpenStack, then we would get the same result as OpenStack, and the number of tenants and the resources belonging to each of them would be known. If the configuration is as built in CloudStack, the following two cases apply;

   - Those public IP addresses could be not detected by penetration test tools, which is unlikely as this IP address must be accessible from outside the cloud to connect to the VM.

   - The more likely case is that VMs Public IP address would be detected by the pentesting tool along with the MAC address. Another piece of exposed information is the MAC address of the tenant private network Gateway and that is one step further deeper inside the cloud than OpenStack.

6.1.6.2 Penetration Test

The following list comprises issues arising during the CloudStack Penetration testing stage:

1. Similar to OpenStack, neither the penetration testing nor the tenant accounts revealed the network isolation methods (VLAN and Graduate Record Examinations (GRE)) used by the cloud. Hence the attacker cannot predict the isolation mechanism of its own VNs and, by extension, the other tenant VN isolation mechanisms.

2. OVS bridges, interfaces, flow rules and any other configuration details are vital components of Cloud Infrastructure and none of this information was leaked by penetration testing.

3. The Tenant Test vRouter had an additional interface belonging to the cloud internal management network, 169.254.0.0/24, as shown in Figure 4.6. Although it is a shared network among all the tenants VN components and the cloud system VMs, the Test tenant is

not granted this information either by the Management server console or through penetration testing.

4. The penetration tools exposed the nature of the vRouters after a comparison of the results with the Test vRouter. This is shown in Tables 5.11, 5.13 and 5.15, and was used as knowledge base to show the network provisioned resources are. In fact, VMs are configured to act as routers. This matches the results from the DeepDive –Table 4.13- which show that the vRouters are System VMs configured to take on the role of routers and DHCP.

5. The number of VNC open ports –Tables 5.10 and 513- exceeded the number of tenant VMs created, and each port represented the tenant VMs and other special VMs, such as the two main System VMs (proxy and storage) and vRouter VMs.

6. None of the Tenant VMs are assigned public IP, which means they are not detected in the penetration test. However, they were partially detected from the open VNC ports as it was known that each VM is related to a VNC port. However, it was not revealed which VM related to which VNC port. Additionally, CloudStack was stronger in securing the VNC with authentication in a way that even if the VNC port was detected by Zenmap and Sparta, they are not accessible using the VNCViewer due to the authentication security applied to each port.

7. Unlike the OpenStack cloud, the CloudStack hypervisor configured the VNC port in each VM XML file –Tables 4.8, 4.9 and 4.10. Therefore, in case the attacker managed to exploit the vulnerabilities of the KVM Host Node (shown in Tables 5.12 and 5.14), and executed the virsh command, which is automatically installed with the hypervisor, the attacker could read the XML file of each VM. Besides the VNC port configuration, this file contains information about the automated instance name given by the system, theVM Virtual Network Interface Card (vNIC)(s) MAC, the names of the OVS bridge(s), the OVS interface(s) and the VLAN of the tenant network that connects to the tenant VMs and vRouters.

8. The Nessus external test, summarised in Table 5.12, in CloudStack revealed different levels of vulnerabilities in the cloud nodes and they ranged from Critical (1), High (4), Medium(6 or 7) to Low (3 or 4). The detected vulnerabilities (in Appendix B.2, Table B.5) were either NFS (storage system) or weak password (Hydra) and SSH.

9. In the external Nessus test, shown in Table 5.12, another target (192.168.31.101) other than cloud nodes revealed eight medium vulnerabilities, which were all related to the SSL protocol. From the DeepDive , this target was known as an IP belonging to the System VM,

the Secondary Storage VM, which is a vital functional component of the cloud. For this component to be revealed with this number of vulnerabilities was a serious security risk.

10. Although the public IP addresses of the System VMs have been exposed and identified as VMs, those VMs are not privileged for tenants to access them and to login.

11. The information about the OVS bridges and their ports was not gained by the penetration methodology. The only way to gain access to the Host Node, in this scenario, is via its vulnerabilities and through learning about the OVS bridges and ports connected to it.

12. The Nessus internal test in §5.3.2.2, scanned the public and Management network that were already scanned by the external test, §5.3.1.3. Additional network tests needed were the tenant private network targets. The internal test matched the external test for the public and Management network targets and no further information was detected. However, the private network targets showed other issues. For a vRouter to carry a medium vulnerability (DNS) and a low one (DHCP) is of concern, see Table 5.15. Although they are detected on the inner side of the cloud by the tenant who owns the vRouter, this research showed that cloud components are often consistent in structure, and therefore vulnerabilities that exist in any tenant vRouter are likely to exist in others. Moreover, we limited this research to three pentesting tools. However there may be other tools and more persistent attackers that could potentially break cloud isolation if such information was discovered.

13. Similar to OpenStack, the tenants instances (VMs) are the most vulnerable components among all the cloud resources. The sample image that CloudStack provided revealed a total of 271 vulnerabilities distributed among the four levels of vulnerabilities, see Table 5.15. As mentioned with OpenStack, this is an opportunity for the attacker tenant to provide a vulnerable image for others to use. Further, because of the CloudStack structure we could not confirm this until we ran Nessus from within the Test tenant KaliLight VM.

The following diagram reproduces the DeepDive diagram from Figure 4.6, and lists the findings of the external and Internal penetration testing.

Following the steps carried out in the OpenStack experiments, Figure 6.2 maps the leaked information via the CloudStack penetration testing. First of all, the two nodes, Management and Host, are represented with a different colour code as the highest vulnerability level discovered by Nessus in the Management Node was Critical. Hence it is shown in red colour. The Host Node is represented with orange to show that it has a High vulnerability level. The other cloud infrastructure components that showed different levels of vulnerability include the

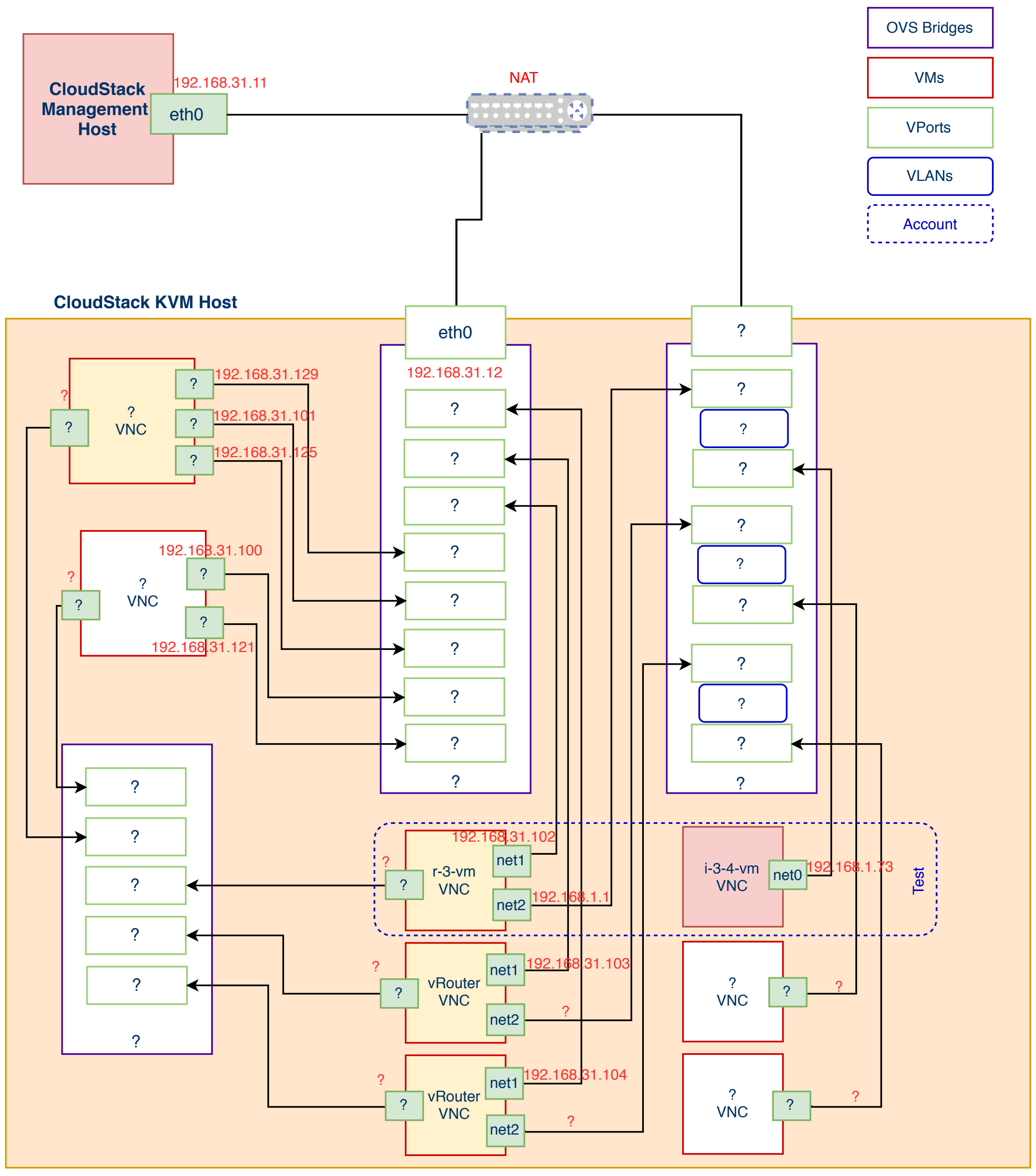


Figure 6.2: CloudStack Infrastructure; Leaked Information from Penetration Testing.

Secondary Storage SystemVM with a medium vulnerability, the Test VM (i-3-4-vm) with a Critical vulnerability and the vRouter with medium vulnerabilities.

All the other tenant vRouters were discovered through their public IP address, their network distance from the attacker and the list of ports that exposed their nature as a VM. The data analysis compared the collected information from the Test Tenant and other targets showed which target was a vRouter. At the same time although the Test vRouter presented medium and low vulnerabilities on the private side, other vRouters with the same vulnerability are likely as the cloud service provisioning is consistent amongst the tenants.

None of the VMs are publicly connected, hence there is no identification of their type of vulnerability. However, an indication of the number of VMs running in the cloud was noted due to the number of Open VNC ports, which represented the System VMs (Console Proxy VM and Secondary Storage VM), all tenant instances and vRouters.

The CloudStack testbed had managed to conceal the number of tenants and their resource ownership unlike the OpenStack testbed. We cannot conclude if CloudStack is more secure than OpenStack, due to differences in the configuration of the Internet connection. However, it also failed to apply SSH connection via assigning a public IP address for each VM. In case of a successful discovery of Public IP for VM SSH through penetration testing, CloudStack security would be weakened.

Unlike OpenStack Dashboard, CloudStack concealed any information relating to OVS bridges, ports and VLANs, even from the tenants that own the resources.

In conclusion, the penetration tools used exposed Data Leakage from both Testbeds. Specifically, the tools that enhanced the knowledge of attackers about the cloud infrastructure were Zenmap and Sparta, while Nessus confirmed what the prior tools leaked and added the vulnerabilities in the system that have a negative security impact. Most of the knowledge plotted in Figures 6.1 and 6.2 were learned from analysing the leaked data of Zenmap and Sparta, while Nessus participated in the colour coding of the targets to show the vulnerability levels. Due to the limited knowledge of the cloud infrastructure, configuration, components and mechanisms, targeting the cloud network tenant components was not feasible without a DeepDive for basic knowledge. This was considered a discovery phase to indicate whether the penetration test process and methodology could gain pertinent information.

6.2 Hypothesis

We have previously provided some evidence of leaked information from penetration testing in Chapter 5 and showed more specific information in §6.1.1 and §6.1.2. Therefore, we reject the null hypothesis, stated in § 1.5, as the security of the IaaS Cloud Virtual Network Isolation is not robust against data leakage caused by the penetration tools used.

In general, the following summarizes the two testbed's security design weaknesses. Scoring criteria is not currently feasible at this stage of research and is regarded as future work:

- The OpenStack VNC was much weaker than CloudStack due to its security configuration and vulnerabilities.
- OpenStack exposed more information about the VMs as they were granted a public IP address each.
- OpenStack tenant ownership is more obvious than CloudStack with the current state of our Testbeds.
- The vRouters (VMs) in CloudStack are more vulnerable than the vRouters (namespaces) in OpenStack .
- Cloud nodes in Cloud Stack are more vulnerable than OpenStack.
- CloudStack major functionality management components (System VMs) were exposed and displayed vulnerabilities.
- OpenStack revealed the OVS and Linux ports codes for their tenants via the dashboard, unlike CloudStack.

This research is one of the few empirical works on cloud security, and this is only known work on CVNI. It provides a novel structured method to identify the cloud network components; configurations and mechanisms. Those components to need to be tested to evaluate the system robustness.

Consequently, the significance of our findings apply to both the scientific world and the public community. Several recommendations are urgent:

1. *Users should be informed that the current state of these cloud structures does not reach an adequate level of Network Isolation security.*
2. *Cloud manufacturers/ developers/ deployers need to follow a structured methodology to identify infrastructure components and appropriately test their implemented isolation mechanisms.*
3. *CVNI should not be considered robust unless the isolation has been analysed and evaluated.*
4. *Using common penetration tools, inside and outside a cloud, an attacker can detect data leakage (chapters 5)*
5. *The Cloud industry needs to evaluate their product testing in line with the outcomes of this research.*

6.3 Mitigation Strategies Suggested by Nessus

Nessus scanned the entire cloud testbeds and gathered valuable information that informed an attacker tenant about the cloud infrastructure, other tenants' resource ownership, and different levels of vulnerabilities. Moreover, Nessus provided a list of solutions to mitigate the discovered problems. This section lists a summary of these solutions for both testbeds.

The complete version of vulnerabilities and solutions can be found in Appendix B. The Tables B.1 and B.2 (Appendix B) represent the vulnerabilities and solutions found by Nessus on OpenStack and Tables B.3 and B.4 relate to the Nessus tests on CloudStack. The appendix also lists the vulnerabilities related to major cloud services and cloud infrastructure network components.

6.3.1 Potential Attack Scenarios

This section uses the vulnerabilities collected from OpenStack and CloudStack Nessus testing. A list of five example attack scenarios per cloud testbed is given. However, a full version of vulnerabilities and potential attacks are listed in Appendix B.

6.3.1.1 Potential Attacks on the OpenStack Testbed

Table 6.1 lists five examples of potential attacks on OpenStack driven from vulnerabilities discovered in the Nessus external and Internal tests – see Appendix B Table B.1 and B.2-.

Vulnerability	Target	Attack
SSH Diffie-Hellman Modulus <= 1024 Bits (Logjam)	Tenants VMs	SSH server allows connections with one or more Diffie-Hellman moduli less than or equal to 1024 bits. This allows an attacker to recover the plaintext or potentially violate the integrity of connections.
Firewall UDP Packet Source Port 53 Rule set Bypass: Firewall rule sets can be bypassed.	A tenant vRouter 192.168.100.13 (namespace)	An attacker may use this flaw to inject UDP packets in the remote hosts, in spite of the presence of a firewall.
TCP/IP Sequence Prediction Blind Reset Spoofing DoS	Controller and Compute Cloud Nodes	The host is affected by a sequence number approximation vulnerability that allows an attacker to send spoofed RST packets to the remote host and close established connections. This may cause problems for some dedicated services (BGP, a VPN over TCP, etc).
VNC Server Unauthenticated Access: Screenshot	Compute Cloud node	The VNC server installed on the remote host allows an attacker to connect to the remote host as no authentication is required to access this service. It was possible to log into the remote service and take a screenshot.
TCP/IP Predictable ISN (Initial Sequence Number) Generation Weakness	Compute and Controller Cloud nodes	An attacker may use this flaw to establish spoofed TCP connections to this host.

Table 6.1: OpenStack Testbed Attack Scenarios

6.3.1.2 Potential Attacks on the CloudStack Testbed

Table 6.2 lists five examples of potential attacks on CloudStack driven from vulnerabilities discovered in the Nessus external and Internal tests –see Appendix B Table B.3 and B.4-.

Vulnerability	Target	Attack
SSLv3 Padding Oracle On Downgraded Legacy Encryption Vulnerability (POODLE)	Secondary Storage System VM	The remote host is affected by a Man-in-the-Middle (MitM) information disclosure vulnerability known as POODLE. MitM attackers can decrypt a selected byte of a cipher text in as few as 256 tries if they are able to force a victim application to repeatedly send the same data over newly created SSL 3.0 connections.
NFS Exported Share Information Disclosure	Management Cloud Node	At least one of the NFS shares exported by the remote server could be mounted by the scanning host. An attacker may be able to leverage this to read (and possibly write) files on the remote host.
DNS Server Spoofed Request Amplification DDoS	Tenant vRouter and Host Cloud Node	By spoofing the source IP address, a remote attacker can leverage this ’amplification’ to launch a Denial of Service attack against a third-party host using the remote DNS server.
dnsmasq < 2.73rc4 setup_reply() Function Return Value Checking Information Disclosure	Tenant vRouter and Host Cloud node	An unauthenticated, remote attacker can exploit this to disclose sensitive information.
Linux Kernel TCP Sequence Number Generation Security Weakness	Tenant -Cloud provided image-VM and Host Cloud node	An attacker may use this vulnerability to create a Denial of Service condition or a Man-in-the-Middle attack.

Table 6.2: CloudStack Testbed Attack Scenarios

6.4 Potential Solutions Derived from Literature

Several research projects propose solutions to enhance and strengthen Virtual Networks. However, based on the findings of our research, testing the basic cloud setups, the specifications proposed by these other solution which have not yet been applied in the IaaS Clouds we tested.

1. The security mitigation of a traditional network is not sufficient for the new challenges under SDN as they are designed and applied to a non-dynamic network configuration, the opposite to SDN. Zhou et al [110] has introduced an Evolving Defense Mechanism (Evolving Defense Mechanism (EDM)) succeeding new network security issues including dynamic feature driven SDN [110].

   - The question is, if we assumed that [110] claim is true, does the cloud network embed such a, or similar mechanisms, in their network implementation to overcome the network security issues that might have been elevated with the wide range of facilitation to tenant

VNs? Throughout our research, no such claim has been encountered to have a similar effect.

2. EDM aims to detect continuous attacks on a dynamic network system, such as SDNs, regardless of the configuration or dynamic configuration of the network [110].

   - There is no evidence of such systems having been used on IaaS clouds with heavy cloud networking loads or the use of SDN technologies within its original model. To the best of our knowledge, this has never been recommended as the best practice for cloud providers.

3. EDM encrypts the communication between the Controller and switches, tenants, administrator and infrastructure. It uses an authentication approach that continually changes within the infrastructure to protect the crucial management data of the SDN and infrastructure [110].

   - We know that there is an authentication approach between cloud integrated services, which is not as dynamically changing as discussed in this solution. Moreover, there is no evidence of encryption of communication between SDN elements. This gives rise to the question of whether this would effect the leakage of crucial network management data in the cloud networking?

4. A (Virtual Network Diagnosis (VND)) cloud controller could be used to schedule and isolate (with tunnelling) the bandwidth and traffic of the network from the tenant traffic. This would protect the cloud management data and platform [74].

   - This technique is not implemented by default in current cloud frameworks, thus cloud development must improve to reach the level of VND functionality.

5. The target of establishing the VN is to accomplish four goals: isolation, flexibility, performance and cheaper service. Building a technology that combines all four features is challenging. Moreover, implementing VN that is fundamentally close to the physical layer results in modest isolation. As the idea behind VN is to abstract and slice the control and the data plane, the Parallelizing data plane (PdP) project proposed a VN solution to virtualize the control and data plane for VN in a VM. As the VN here would be contained in a form of VM then it would be flexible enough to customize the control and data plane. Moreover, VM isolation is already an advanced technology [172].

   - In our research only one testbed, CloudStack, represented a similar solution, although not for the entire VN [148]. Only one part of the tenant VN had been configured as a VM and implemented all the necessary network functions with regards to the tenant requirements. CloudStack uses VMs to create its vRouter and the tenant may choose any network function such as routing, Network Address Translation (NAT), DNS, DHCP, Load Balancer (LB).

6. A cloud network VN solution called CloudNaaS has been proposed for a cloud to fill in the requirement of the on-demand, dynamic cloud networking service [77] and tenants to be able to create isolated networks with their own networks: adding IP range definition, servers selection, NFV services and "middlebox" choices [56]. The main aim of this solution was to facilitate Virtual Networks with security assurance; isolation and performance guaranteeing and adaptable middleboxes, which are specialized components for network functionality such as intrusion detection or NAT . CloudNaaS is designed to be part of the cloud network infrastructure empowered with its programmable network devices/switches to confirm the network effectiveness. CloudNaas is based on the OpenFlow protocol that provides the ability to establish the networking roles without dropping the infrastructure performance even with considerable amount of failed provisioned services, devices and links. OpenStack is a potential cloud stack that could be adapted to CloudNaaS [77].

   - CloudNaas been mentioned as a potential solution for the lack of network control and configuration flexibility by the cloud tenant to add network isolation, tenant defined address scheme, IDS, etc. CloudNaaS is not considered to be a specialized security solution. However, it may enhance VN Isolation, thus it might contribute to VN security in general. But it is not clear how is it used in the cloud market, if at all? If such a solution is embedded within a cloud, would it protect against VN Data Leakage found by our research methodology?

7. Xiao et al [10] describe a case of Data Leakage using indexing and refer to indexing as it has traditionally been used in database search. In this case, based on the research of Squicciarini et al[173], an index based on keywords on the tenant data used only the "read" privilege. However, an indexed document is hardly secured or prepared with a privacy set mode, and hence it is a significant cause of data leakage. According to Xiao et al [10] there are three levels of data protection structures on the tenants data for cloud privacy: Information centric security, Trusted computing and Cryptographic protocols.

   - The above is a case and solution for tenant Data Leakage in the cloud caused by an indexing structure. However, is it applied on the infrastructure functionality data and configuration files? Is it the search using indexes on keywords that can be found in the operation and configuration files that leads to cloud network infrastructure information leakage? Or, does the cloud protect the indexing that is relevant to the tenant but not its own infrastructure files?

8. The Cedo project [174] is a solution that fights against less trustworthy platforms or administrative actions without monitoring. It detects any rogue components prior to initiation and causing risk or vulnerabilities, such as leaking data.
   - It is unclear if the cloud uses a similar mechanism to measure , or provide pre-defined levels of infrastructure trust, or detect if the components launched are not manipulated harmfully.

9. SilverLine [175] is proposed as a "security as service" solution to a cloud provider to enlist it as a tenant service offering tenant data protection. SilverLine boosts the VM's ability to provide data and network isolation. Moreover, it powers the tenant VM's OS with further data isolation to prevent Data Leakage caused by "compromise, misconfiguration, or side-channel attacks" triggered by neighboring tenants.
   - Scott-Hayward et al [27]noted that solutions to Data Leakage and Data Modification are not yet mature and are not suitable to be applied in a production environment. Therefore, we conclude that it is unlikely that these otherwise useful ideas are implemented in current clouds. Moreover, Mundada et al's Silverline system [175] secures tenant data, yet we are looking at another direction, that is, preventing network service configuration from leaking information. We believe that the cloud service and configuration information is as important as the tenant data, and leaking functional information may effect the whole cloud's security.

Based on our experience in investigating and setting up clouds, any addition to a cloud will result in unexpected changes or affect cloud production. Thus, any change to the cloud must be tested. For example, each solution must be tested thoroughly before officially integrating a new component into a cloud. Moreover, according to Penetration Testing general guidelines [6], testing should be automatic when there are infrastructure changes.

6.5 Cloud Infrastructure Design Mitigation and Guidelines

Based on the results of our in-depth DeepDive study, the experiments on cloud infrastructures and penetration testing, we propose the following mitigation strategies and guidelines.

1. The Dashboard/Management Server Console should be run under an IP that is not part of the Management Node, or it should be disguised from the real IP address.

2. Separate the Management network from the public network to avoid the visibility of Cloud nodes/hosts. This is more applicable to CloudStack and OpenStack if the simpler setup

approach is used.

3. Public networks are a must in the cloud to provide connectivity to the tenant. However, using a shared network is dangerous and defies the multi-tenancy of the cloud. Isolation as a process might be intact and function well for traffic handling, but penetration testing was used to collect and extract the infrastructure and network information. This can lead to breaking isolation by a rogue tenant applying attacks against the cloud network components.

4. One feasible solution is to provide each tenant with a physical interface to the public network with its own range that it not shared with any other tenants. However, this solution is neither practical nor flexible as physical environments have a specific capacity that will not cope with the increasing number of tenants with additional network interfaces. Hence, a shutdown may need to be performed, which leads to cloud system interruption or, separately, moving loads onto other servers if load balancing is available.

5. Another solution is to apply the same idea as in (4), except using a logical approach. As the cloud already applies automated network configuration, such as adding ports to the network bridges when a network component or a VM is added. We suggest using the same approach to provide a further separation for public access. In this way, when a tenant is added, a specific set of management configurations must be established. For example, adding a VLAN to separate logical ports from the physical ports and adding more configuration details to the bridges and firewalls to guarantee the successful connection:

   - E.g: if ethX is the public physical network, which is the address to the bridge that is designated for public connection, then ports can be logically sliced into logical ports each isolated with a VLAN, such as ethX.1 for Tenant 1 with VLAN1 and configured within a network range that is not shareable with other tenants. Hence, there can be no pre-knowledge of what other tenant components are connected to, therefore they cannot be detected through rogue tenant pentesting.

6. The solution in point (5) above has not been empirically tested, as this is out of the research scope. Such a solution requires an in-depth knowledge in cloud platform development and its implementation requires further time investment.

6.6 Conclusion

The outcomes of Objective 3, the DeepDive methodology, and Objective 4, penetration testing, were used to inform this chapter. The discussion was divided the into two sections: the common issues of the two testbeds (OpenStack and CloudStack) and the specific issues and findings of each testbed.

As a result of the research, the leaked information from the cloud infrastructure was displayed in diagrams similar to the DeepDive diagrams presented in earlier chapters, one for each testbed.

From this work, we rejected the null hypothesis from § 1.5 as “The security of the IaaS Cloud Virtual Network Isolation is not robust against Data Leakage caused by the current penetration tools. “

Lastly, we listed several mitigations and guidelines for Cloud Virtual Network Isolation to achieve our Objective 5 which was to improve CVNI.

Interestingly, the research literature presented several projects that showed potential solutions to improve the security of CVNI but were not yet implemented in the cloud or published. However, based on our experience with cloud structuring, such systems are tricky to troubleshoot where any additional technology to the cloud impacts functionality. Therefore we recommend thorough testing throughout any solution adoption.

Chapter Seven

Conclusion

7.1 Overview

The overall aim of this research was to detect Virtual Network (VN) Isolation methods used in Cloud testbeds, as a primary validation test case, and to identify any VN isolation vulnerabilities causing data leakage through penetration testing.

Additionally, this work was concerned with determining the relationship between isolation methods used in cloud VNs and the type of data being leaked through applying penetration tests. The goal of the experiments was reveal what information in a cloud network setup should be isolated to ensure security. The experiments successfully identify Virtual Network components, which could be compromised to effect data or asset leakage [176].

7.1.1 Research Question

To remind the reader, the overarching research question was: How much Cloud Virtual Network information can be found by using commonly available, open source, public penetration testing tools as would be used by a malevolent or inquisitive user? That is, can common scanning and reporting tools such as the Kali testing suite penetrate a Cloud VN?

The Research Hypothesis derived from the research question was:

Hypothesis1: an attacker can discover Data Leakage of Cloud tenant VN information due to poor Cloud Virtual Network Isolation and infrastructure design.

The Null Hypothesis is therefore that the security of the IaaS Cloud Virtual Network Isolation is robust against Data Leakage caused by the current penetration tools.

To answer this Hypothesis, two empirical based sub-hypotheses were stated as below and tested in Chapter 5:

1. **Hypothesis 1.1**: Using standard pentesting tools, through an externally run Kali (situated outwith the cloud), information about Tenant2's VN can be gained by rogue Tenant1, (owned by attacker "Eve"), within the same cloud environment. (External Penetration Testing; §5.2.1 & 5.3.1)

2. **Hypothesis 1.2** Using standard pentesting tools, through an internally run Kali (run within a tenant VM), further information about Tenant2's VN can be gained by rogue Tenant1 within the same cloud environment. (Internal Penetration Testing; §5.2.2 & 5.3.2)

7.2 Contribution

As stated in Chapter 1 (§1.4), this research aimed to empirically analyse Cloud Virtual Network Isolation, using two testbeds. Two major contributions following from this are:

1. This research is one of the few empirical works that investigate Cloud Security and one of the earliest that investigate Virtual Network Isolation (CVNI) Security for Data Leakage. Other empirical papers consider VM instance placement and Cache Side Channel attacks such as Ristenpart et al. in 2012[42] and Saeed et al. in 2018 [122] investigate Network Channel Attacks on Virtual Machines and Virtual Networks.

2. This research has produced a methodology to test CVNI; a Deep Dive test determines components and is followed by paired bi-directional tests, from external and internal to the Cloud, which uses penetration testing to simulate a rogue tenant's activities.

7.3 Attained Objectives

The research objectives were achieved throughout by answering the research objectives as follows:

1. What cloud platforms could be a possible environment to apply the research question on?
   - Cloud platform environment: We used a *private single server IaaS Cloud* to investigate the occurrence of Virtual Network (VN) configuration and functionality information leakage.

2. What are the required resources to evaluate CVNI?

a) Required resources: we built *two IaaS cloud Testbeds* with advanced VN isolation provisioning services and compared their structures.

   i. *Chapter 3: OpenStack IaaS Cloud Testbed – §3.2*

   ii. *Chapter 4: CloudStack IaaS Cloud Testbed – §4.2*

3. What is the best methodology to identify the construction of the cloud network infrastructure, VN components and Isolation mechanism?

   a) *Deep Dive Methodology (our own version of White box Testing).*

      i. *Chapter 3: OpenStack Infrastructure Deep Dive – §3.4*

      ii. *Chapter 4: CloudStack Infrastructure Deep Dive – §4.4*

4. How can Data Leakage be detected or determined? And what are the characteristics of the data leaked?

   a) We collected leaked information from running *Penetration Tests (Internally and Externally – Chapter5)* and *analysed information to determine* the components and configuration of the tenant VNs within the Cloud Network infrastructure of the two testbeds.

5. How can we improve the security of CVNI in the presence of Data Leakage?

   a) We presented the *vulnerabilities and misconfiguration / mis-design* of the Cloud Network infrastructure and proposed *mitigation* and security *guidelines in Chapter 6.*

The expectation was, as stated by Hakiri et al. [91] that cloud virtual networks should be a provisioned service with complete network isolation and self-defined network addresses supported by a variety of network services.

However, this research concluded that the security of the IaaS Cloud Virtual Network Isolation is **not robust** against Data Leakage caused by the current penetration tools. Hence, an attacker can discover Data Leakage of Cloud tenant VN information due to poor Cloud Virtual Network Isolation and infrastructure design.

Nguyen-ngoc et al. [86] investigated the relationship between VN isolation and traffic congestion. Their results showed that VN isolation breakage was achievable if the isolation performance is compromised via the congestion level of the exiting interface.

Our research found that the two tested IaaS clouds adopted a design where all the tenants shared the same public/external network exiting from the cloud. This is an indication of cloud network infrastructure mis-design that breaks tenant VN isolation in case the cloud is challenged by a traffic rate rise.

7.4 Future Work

This research concluded that IaaS Cloud Networking Infrastructure is not up to the security expectation of preventing leaking cloud infrastructure and tenant VN information. It is one of the earliest empirical works to test the CVNI security for Data leakage and the methodologies used remain immature. Hence, our research provides a foundation for future work to enhance security issue discoveries and mitigation. The following are potential future work that could contribute to the field:

1. Advanced Cloud Network Isolation Testbed/Scenarios:
    a) SR-IOV:
        i. SDN was a crucial component in cloud infrastructure for reliable, less delayed traffic transfer between the cloud services when applications with technical or scientific requirement run within the cloud. Single Root Input/ Output Virtualization (Single Root Input/ Output Virtualization (SR-IOV)) is another suggested solution by Rad et al. [50].
        ii. Moreover, deploying server-side network virtualization compromised the security and isolation needed by multi-tenancy, in the cloud environment. Callegati et al. [62] suggested SR-IOV technology using external switches working with Virtual Ethernet with Port Aggregator (VEPA). Together, these technologies receive all the local and external traffic and process them instead of the cloud host doing the processing.
        iii. Both the above SR-IOV technology solutions alter the Cloud Network infrastructure (Deep Dive) of an IaaS testbed. Moreover, although [50] [62] claim that SR-IOV technology would enhance cloud networking isolation, security and performance, further research is required to test the CVNI security for data leakage on such a structure.
    b) OpenVirteX (OVX) [16][93]:
        i. OpenVirteX (OVX) is a virtual Software Defined Networks (vSDN) platform for more tenant network customization in terms of their topologies, address scheme and choice of Network OS (NOS). OVX virtualized the network environment,

SDN, to provide a network virtualization platform in the form of slices of vSDN.

ii. OVX aimed for each tenant to create their own logical SDN that can have snapshots and be migrated, as well as give a better switch connection elasticity. Moreover, it could reduce the infrastructure provider management responsibilities and complexity by giving the tenant the privilege to customize and define their own network topology, policies and functionalities.

iii. OVX provided the OpenStack and OpenCloud [1] Neutron plugins. Therefore, implementing an OVX solution for Cloud Networking alter the Cloud Network infrastructure (Deep Dive) of the IaaS testbed.

iv. Al-Shabibi et al. claim [16] [93] that OVX showed better security performance that the traditional SDN. However, empirical evidence for CVNI security for Data Leakage requires more research.

2. Test Security enhanced CVNI against Data Leakage:

   a) One of the interesting cloud networking SDN-based solutions is the Cloud Networking Gateway Manager [78], which extends cloud network management throughout SDN-based control from a single cloud provider into distributed cloud networking providers [78].

      i. Our research testbeds are the application of two individual, standalone single cloud provider testbeds for simplicity, and not a distributed cloud. Therefore, a cloud application based solution such as the Cloud Networking Gateway Manager is worth researching for CVNI security for Data Leakage.

   b) The best practice recommendation by Scott-Hayward et al. [27] is to secure shared network environments from various security issues such as “unauthorized access, data manipulations, and preventing data leakages”. This was to be done not by using abstraction via VN, instead, by isolating the network resources via slicing, as slicing provides a better resource isolation and allocation dedication than VN.

      i. Our two testbeds depend mainly on VN based on SDN and NFV, and supported by cloud functionality network control. Scott-Hayward et al.[27] list a well-defined gap of SDN security issues for Data Leakage. Hence, adopting the cloud networking based on slicing and empirically testing the cloud for CVNI Data Leakage is a potential future research.

[1] https://ovx.onlab.us/openstack/

c) Smart Applications on Virtual Infrastructure (Smart Applications on Virtual Infrastructure (SAVI)) [56]:

   i. This project established a Software Defined Infrastructure (SDI) mechanism, which is a pluggable module combining both cloud controller and SDN controller to manage both compute and network service and resources.

   ii. SDI architecture is built based on Openstack representing the compute management and OpenFlow technology for the SDN controller. As any SDN technology it is responsible for creating slices, implanting isolation and controlling traffic flow in the VN.

   iii. SAVI is a solution for cloud networks based on the slicing approach and it fits perfectly with Scott-Hayward et al's [27] recommendation in point (b) above.

d) SDN network isolation mechanisms [79]:

   i. SDN development did not pay close attention to securing the data plane and simply treated all the flows to be of the same kind, which makes it easy for an attacker to distinguish the traffic flow patterns. Hence a traffic injection attack could break the network functionality.

   ii. It is vital for SDN to provide network isolation mechanisms for the data flow across the entire cloud. Such a solution is managed by the network controller as a multi-tenant isolation mechanism. A protocol, such as OpenFlow, performs dynamic physical data flow isolation in the cloud networking.

   iii. Unless using a third party controller, a cloud infrastructure builds its cloud networking without a stand-alone SDN controller, and an SDN uses a central basic default built-in controller. With a highly sophisticated controller a multi-tenant network isolation might not be fully granted, and then the default setting of the cloud as we built is risky.

   iv. A cloud developer needs to investigate the currently available controllers and rank them according to the cloud requirements and offer these choices for the default cloud setup.

   v. From the available controllers, the cloud developer may create a cloud network controller that completely fits with their cloud requirement as a core part of the cloud setup. This is a potential future research work on the cloud networking with SDN.

3. Cloud Network-SDN configuration data leakage:

    a) Based on Scott-Hayward et al[27] listed SDN security criteria and several categorisations of the security issues associated with the SDN Framework. The Data Leakage issues contained some aspects associated with the SDN layer. One issue is “Flow Rule Discovery” which can be exposed by a “Side Channel Attack on Input Buffer”. This is expected to target only the SDN “Data layer”. Another issue is “Forwarding Policy Discovery” via “Packet Processing Timing Analysis”, where the targeted layers of these issues are the Control layer, the Control-Data interface and the Data layer.

        i. Our research testbeds adopted SDN and NFV technologies. The security methodology used in this thesis, Penetration Testing, successfully leaked vital information about cloud network infrastructure including what was owned by different tenants. However, the leaked information did not include any part of the SDN technology within the infrastructure. Therefore, discovering SDN data such as Flow Rules and Forwarding Policies using a different approach or methodologies is a potential research project.

4. Automated CVNI Data Leakage security tool:

    a) The Deep Dive methodology presented in Chapters 3 and 4, along with the Penetration Testing method illustrated in Chapter 6, were processed manually in this research. As mentioned earlier, these methodologies and applications require further development. Therefore, once the technologies and tools develop further, automated testing tools could be used to assess CVNI security for Data Leakage.

5. Cloud IaaS Infrastructure design modification:

    a) In chapter 6, we listed mitigation guidelines based on our own observations on the two testbeds we tested. One point in §6.5 specifically illustrated an Infrastructure design modification.

    b) A potential future work is to build on this research and investigate the feasibility of implementing the infrastructure design modification we suggested or further develop this design.

7.5 Conclusion

This research developed a methodology to test cloud virtual network isolation security for data leakage through a cloud infrastructure Deep Dive and penetration testing. The work was time consuming in having to setup cloud testbeds with limited online or technical support. Further, the testing of cloud network security was supported by very little academic or industrial literature.

The Deep Dive was customized after troubleshooting the cloud testbeds. However, this led to a better understanding of the Deep Dive procedures and eventually led to answering the research objectives. Consequently, the Deep Dive procedures are a novel contribution to this field. The Deep Dive was a necessary process to both compare the data leaked through the external and internal penetration testing phases and, separately, to determine what virtual network isolation mechanisms were used by the cloud network service. These processes demonstrated a concern about the amount of data leakage and therefore the Null Hypothesis is proven in this thesis: an attacker can discover Data Leakage of Cloud tenant VN information due to poor Cloud Virtual Network Isolation and infrastructure design

As one of the few empirical works on cloud virtual network isolation security, this thesis demonstrates that cloud security is a wide open research challenge and concern.

APPENDIX A

APPENDIX-A TESTBEDS SETUP AND DEEP DIVE METHODOLOGY EXTRA DETAILS

A.1 OpenStack Testbed Setup and Deep Dive

Requirements	**Controller**	**Compute**
Packages installed	cloud-archive:mitaka ntp rabbitmq-server mysql-server python-pymysql python-OpenStackclient openvswitch-switch keystone apache2 libapache2-mod-wsgi glance nova-api nova-conductor nova-consoleauth nova-novncproxy nova-scheduler neutron-server neutron-DHCP-agent neutron-l3-agent neutron-metadata-agent neutron-plugin-ml2 neutron-plugin-openvswitch-agent	cloud-archive:mitaka ntp python-OpenStackclient nova-compute sysfsutils neutron-openvswitch-agent
Services running	mysql apache2 glance-api glance-registry nova-api nova-consoleauth nova-scheduler nova-conductor nova-novncproxy neutron-l3-agent neutron-metadata-agent neutron-openvswitch-agent neutron-DHCP-agent neutron-server	nova-compute neutron-openvswitch-agent

Table A.1: Mitaka Cloud packages installed in each node.

A.2 CloudStack Testbed Setup and Deep Dive

Node	File	Configuration
Controller	/etc/neutron/l3_agent.ini	interface_driver = neutron.agent.linux. interface.OVSInterfaceDriver
	/etc/neutron/plugins/ml2/ openvswitch_agent.ini	[agent] tunnel_types = vxlan [ovs] bridge_mappings = external:br-ex [securitygroup] firewall_driver = neutron.agent.linux. iptables_firewall.OVSHybridIptablesFirewallDriver

	/etc/neutron/DHCP_agent.ini	[DEFAULT]
		interface_driver = neutron.agent.linux.
		interface.OVSInterfaceDriver
		dnsmasq_config_file = /etc/neutron/dnsmasq.conf

	/etc/neutron/plugins/ml2/
	ml2_conf.ini
		[ml2]
		type_drivers = vxlan,flat
		tenant_network_types = vxlan
		mechanism_drivers = openvswitch
		
		[ml2_type_vxlan]
		vni_ranges = 10:100
		vxlan_group = 224.0.0.1

	/etc/nova/nova.conf	[neutron] ovs_bridge = br-int
Compute	/etc/neutron/plugins/ ml2/openvswitch_agent.ini	[agent] tunnel_types = vxlan [ovs] enable_tunneling = true [securitygroup] firewall_driver = neutron.agent.linux.iptables_firewall. OVSHybridIptablesFirewallDriver
	/etc/nova/nova.conf	[neutron] ovs_bridge = br-int

Table A.2: OpenStack Mitaka Cloud Configuration Files Options.

Instance Details	**Test**		**T1user1**	**T1user2**
Name	KaliLight	my_first_instance	t1u1vm	t1u2vm
flavor	M1.medium	m1.tiny	m1.tiny	m1.tiny
image	KaliLight2	Cirros 0.3.4	Cirros 0.3.4	Cirros 0.3.4
security-group	default	default	-	default
net-id	Network1	Network1	-	T1user2net
IP address	10.190.0.7	10.190.0.6	192.168.10.3	192.168.20.3
floating IP	192.168.100.16	192.168.100.11	192.168.100.14	192.168.100.15

Table A.3: Tenant Instances and Their Configuration Details

Requirements	**Management**	**KVM Host**
Packages installed	NTP Server. Cloudstack-management. Mysql server. NFS Kernel Server.	NTP Server. Openvswitch-switch. Openvswitch-common. Cloudstack-agent.
Services running	SSH. Mysql. Nfs-kernel-server. Cloudstak-management. Tomcat6.	SSH. Cloudstack-agent.

Table A.4: CloudStack Packages Installed in Each Node.

Configuration	**Management Node**	**KVM Host Node**
[File] /etc/network/ interfaces	*auto eth0* *iface eth0 inet static* address 192.168.31.11 *netmask 255.255.255.0* *gateway 192.168.31.2* *dns-nameservers 192.168.31.2*	# Set eth0 for bridging *auto eth0* *iface eth0 inet manual* up ip link set dev $IFACE up down ip link set dev $IFACE down # The OVS bridge interface auto cloudbr0 iface cloudbr0 inet static address 192.168.31.12 netmask 255.255.255.0 *gateway 192.168.31.2* *dns-nameservers 192.168.31.2* # Set eth1 for bridging *auto eth1* *iface eth1 inet manual* up ip link set dev $IFACE up down ip link set dev $IFACE down
[File] /etc/hosts	192.168.31.11 cldstk-mgmt.mycloud.local cldstkmgmt 192.168.31.12 cld-stkkvm.mycloud.local cldstkkvm	192.168.31.11 cldstkmgmt.mycloud.local cldstkmgmt 192.168.31.12 cldstkkvm.mycloud.local cldstkkvm
[Command] OVS	None	sudo ovs-vsctl add-br cloudbr0 sudo ovs-vsctl add-port cloudbr0 eth0 sudo ovs-vsctl add-br cloudbr1 sudo ovs-vsctl add-port cloudbr1 eth1

[File] /etc/cloudstack/agent/agent.properties	None	network.bridge.type=openvswitch
		libvirt.vif.driver=com.cloud.hypervisor.kvm.resource.Ov

Table A.5: CloudStack Network Files and Commands Configuration.

# list	Name	MAC	IP
1	lo	00:00:00:00:00:00	127.0.0.1/8
2	eth0	00:0c:29:72:d6:cf	
3	eth1	00:0c:29:72:d6:d9	
4	ovs-system	1a:55:6e:0e:1d:dc	
5	cloudbr0	00:0c:29:72:d6:cf	192.168.31.12/24
6	cloudbr1	00:0c:29:72:d6:d9	
7	virbr0	7e:ed:2b:57:c3:05	192.168.122.1/24
8	cloud0	6e:19:aa:d7:57:4c	169.254.0.1/16
9	vnet0	fe:00:a9:fe:00:b5	
10	vnet1	fe:3d:d6:00:00:01	
11	vnet2	fe:e9:a6:00:00:15	
12	vnet3	fe:00:a9:fe:02:18	
13	vnet4	fe:d4:3a:00:00:05	
14	vnet5	fe:ee:34:00:00:16	
15	vnet6	fe:88:18:00:00:09	
16	vnet7	fe:00:a9:fe:00:71	
17	vnet8	fe:00:ca:00:00:17	
18	vnet9	fe:00:52:86:00:02	
19	vnet10	fe:00:1c:0f:00:01	
20	vnet11	fe:00:a9:fe:01:0e	
21	vnet12	fe:2f:60:00:00:18	
22	vnet13	fe:00:46:c8:00:02	
23	vnet14	fe:00:12:bc:00:01	
24	vnet15	fe:00:a9:fe:01:e9	
25	vnet16	fe:59:68:00:00:19	
26	vnet17	fe:00:63:79:00:02	
27	vnet18	fe:00:7f:e8:00:01	

Table A.6: KVM Host Node Port List.

APPENDIX B

APPENDIX-B PENETRATION TEST EXTRA DETAILS

B.1 Penetration Test Analyses Information

Table B.1 is the CloudStack Internal Penetation testing Zenmap information analysis.

IP	MAC	Port(s)	Service	Version
192.168.31.11	00:0C:29:C5:69:95 (VMware)	22/TCP	SSH	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.10 (Ubuntu Linux; protocol 2.0)
		111/TCP 111/UDP	rpcbind	2-4 (RPC #100000)
		2049/TCP 2049/UDP	nfs_acl	2-3 (RPC #100227)
		8080/TCP	HTTP	Apache Tomcat/Coyote JSP engine 1.1
		9090/TCP	zeus-admin?	
192.168.31.12	00:0C:29:72:D6:CF (VMware)	22/TCP	SSH	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.10 (Ubuntu Linux; protocol 2.0)
		111/TCP 111/UDP	rpcbind	2-4 (RPC #100000)
		5900/TCP 5901/TCP 5902/TCP 5903/TCP 5904/TCP 5905/TCP 5906/TCP 5907/TCP	VNC	VNC (protocol 3.8)
192.168.31.100	06:E9:A6:00:00:15	80/TCP	HTTP	JBoss Enterprise Application Platform
		443/TCP	HTTPS	
192.168.31.101	06:EE:34:00:00:16	80/TCP	HTTP	Apache HTTPD
		443/TCP	SSL/HTTP	Apache HTTPD
192.168.31.102	06:00:CA:00:00:17			
192.168.31.103	06:2F:60:00:00:18			
192.168.31.104	06:59:68:00:00:19			
192.168.31.121	06:3D:D6:00:00:01	8001/TCP	HTTP	JBoss Enterprise Application Platform
192.168.31.125	06:D4:3A:00:00:05			
192.168.31.129	06:88:18:00:00:09			

Table B.1: Internal Pentesting Zenmap; Analysis of the CloudStack Active Network 192.168.31.0/24.

Table B.2 is the CloudStack Internal Penetation testing Sparta information analysis.

IP address/ Ports per Stages	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
192.168.1.1	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060	53	
192.168.1.73	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.1.62				8834	
192.168.31.11			22, 111, 2049, 8080	7080, 8250, 9090, 20400	33405, 34829, 35900, 38462, 45671, 46270, 48158, 50883
192.168.31.12			22, 111,	5900, 5901, 5902, 5903, 5904, 5905, 5906, 5907, 5908, 16509	38589, 44706
192.168.31.100	80	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.101	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.102	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.103	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.104	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.121	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060	8001	
192.168.31.125	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		
192.168.31.129	80, 443	25, 135, 137 (TCP/UDP), 139, 445, 1433, 3306, 5432, 161, 162, 1434	21, 22, 23, 110, 111, 2049, 3389, 8080, 500, 5060		

Table B.2: Internal Pentesting Sparta analysis of CloudStack Active Network 192.168.31.0 and Test Private Network 192.168.1.0/24.

B.2 Nessus Targets, Vulnerabilities and Solutions

Vulnerability Name	**Solution**	**Targets**
Nessus UDP Scanner	Protect your target with an IP filter or implement ICMP rate limitation.	192.168.100.10 192.168.100.11 192.168.100.12 192.168.100.14 192.168.100.15 192.168.100.16
ICMP Timestamp Request Remote Date Disclosure	Filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).	192.168.100.10 192.168.100.11 192.168.100.12 192.168.100.13 192.168.100.14 192.168.100.15 192.168.100.16
Patch Report	Install the required security patches.	192.168.100.10 192.168.100.11 192.168.100.12 192.168.100.13 192.168.100.14 192.168.100.15 192.168.100.16
Nessus SYN scanner, Nessus TCP scanner	Protect your target with an IP filter.	192.168.100.11
Dropbear SSH Server < 2013.59 Multiple Vulnerabilities	Upgrade to the Dropbear SSH 2013.59 or later.	192.168.100.11 192.168.100.14 192.168.100.15
SSH Server CBC Ciphers Enabled	Contact the vendor or consult product documentation to disable CBC mode cipher encryption, and enable CTR or GCM cipher mode encryption	192.168.100.11 192.168.100.14 192.168.100.15
SSH Weak MAC Algorithms Enabled	Contact the vendor or consult product documentation to disable MD5 and 96-bit MAC algorithms	192.168.100.11 192.168.100.14

Continued on next page

Table B.3 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
		192.168.100.15
Dropbear SSH Server < 2016.72 Multiple Vulnerabilities	Upgrade to Dropbear SSH version 2016.74 or later	192.168.100.11 192.168.100.14 192.168.100.15
Dropbear SSH Server < 2016.72 xauth Command Injection	Upgrade to Dropbear SSH version 2016.72 or later	192.168.100.11 192.168.100.14 192.168.100.15
SSH Diffie-Hellman Modulus <= 1024 Bits (Logjam)	Reconfigure the service to use a unique Diffie-Hellman moduli of 2048 bits or greater	192.168.100.11 192.168.100.14 192.168.100.15
Firewall UDP Packet Source Port 53 Ruleset Bypass	Either contact the vendor for an update or review the firewall rules settings.	192.168.100.13
Nessus SYN scanner	Protect your target with an IP filter.	192.168.100.11 192.168.100.14 192.168.100.15

Table B.3: OpenStack 192.168.100.0/24 Target Vulnerabilities and Solutions suggested by Nessus.

Table B.3 represents the vulnerabilities discovered by Nessus on the OpenStack cloud targets within the 192.168.100.0/24 network. The table lists the target vulnerabilities and the solutions suggested by Nessus to overcome the security risks that may cause attacks on the targets. Note that some vulnerabilities occur in multiple targets, while others are unique to specific targets. The following list details some vulnerabilities and the potential attacks that might be launched through those vulnerabilities.

1. Nessus UDP Scanner: to determine the open UDP ports.
    a) The target machine must be protected by a firewall; therefore this technique will not distinguish open ports from filtered ports and will then fail.
    b) Almost all the targets from 192.168.100.0/24 are required to fix this issue.

2. Nessus TCP scanner: to determine open TCP ports.
   a) This is a reasonably quick test even against a firewalled target.
   b) Once a TCP connection is open, any available banners for the service identification plugins are copied.
   c) All the targets from 192.168.100.0/24 are required to fix this issue.
3. Nessus SYN scanner: to determine half-open TCP ports.
   a) This is reasonably quick even against a firewalled target but less intrusive than TCP (full connect) scans against broken services.
   b) This might cause problems for less robust firewalls and also leave unclosed connections on the remote target.
   c) Most of the tenants VMs are required to fix this issue.
4. ICMP Timestamp Request Remote Date Disclosure: to determine the exact time set on the remote host.
   a) This allows an attacker to know the date that is set on the targeted machine, which may assist an unauthenticated, remote attacker in defeating time-based authentication protocols.
   b) All the targets from 192.168.100.0/24 are required to fix this issue.
5. Patch Report: The remote host is missing security patches.
   a) All the targets from 192.168.100.0/24 are required to fix this issue.
6. Dropbear SSH Server < 2013.59 Multiple Vulnerabilities: The target SSH service is affected by multiple vulnerabilities.
   a) Causes a Denial of Service vulnerability. (CVE-2013-4421)
   b) User-enumeration is possible due to a timing error when authenticating users. (CVE-2013-4434)
   c) The Tenant VMs are required to fix this issue.
7. SSH Server CBC Mode Ciphers Enabled: The SSH server is configured to use Cipher Block Chaining.

   a) This may allow an attacker to recover the plaintext message from the ciphertext.

   b) The Tenants VMs are required to fix this issue.

8. SSH Weak MAC Algorithms Enabled: The remote SSH server is configured to allow MD5 and 96-bit MAC

   a) The SSH server is configured to allow either MD5 or 96-bit MAC algorithms, both of which are considered weak.

   b) The Tenants VMs are required to fix this issue.

9. Dropbear SSH Server < 2016.72 Multiple Vulnerabilities: The SSH service running on the remote host is affected by multiple vulnerabilities

   a) An unauthenticated, remote attacker can exploit improper handling of string format specifiers (e.g., %s and %x) in usernames and host arguments to execute arbitrary code with root privileges. (CVE-2016-7406)

   b) An unauthenticated, remote attacker can exploit improper handling of specially crafted OpenSSH key files to execute arbitrary code. (CVE-2016-7407)

   c) An unauthenticated, remote attacker can exploit flaws existing in dbclient when handling the -m or -c arguments in scripts, via a specially crafted script, to execute arbitrary code. (CVE-2016-7408)

   d) A local attacker can exploit flaws existing in dbclient or the DropBear server if they are compiled with the DEBUG_TRACE option and then run using the -v switch to disclose process memory. (CVE-2016-7409)

   e) The Tenants VMs are required to fix this issue.

10. Dropbear SSH Server < 2016.72 xauth Command Injection: The remote SSH service is affected by a command injection vulnerability.

   a) An authenticated, remote attacker can exploit improper sanitization of X11 authentication credentials to execute arbitrary xauth commands on the remote host. Note that

   b) Affected by command injection vulnerability when X11 Forwarding is enabled and X11 Forwarding is not enabled by default.

   c) The Tenants VMs are required to fix this issue.

11. SSH Diffie-Hellman Modulus <= 1024 Bits (Logjam): The remote host allows SSH connections with smaller Diffie-Hellman bit lengths.
    a) SSH server allows connections with one or more Diffie-Hellman moduli less than or equal to 1024 bits. Through cryptanalysis, a third party can find the shared secret in a short amount of time.
    b) This allows an attacker to recover the plaintext or potentially violate the integrity of connections.
    c) The Tenants VMs are required to fix this issue.
12. Firewall UDP Packet Source Port 53 Rule set Bypass: Firewall rule sets can be bypassed.
    a) It is possible to bypass the rules of the remote firewall by sending UDP packets with a source port equal to 53.
    b) An attacker may use this flaw to inject UDP packets to the remote hosts, in spite of the presence of a firewall.
    c) A tenant vRouter 192.168.100.13 (namespace) is required to fix this issue.

Vulnerability Name	**Solution**	**Targets**
Web Server Generic XSS	Contact the vendor for a patch or upgrade.	192.168.137.10
Nessus SYN scanner	Protect your target with an IP filter.	192.168.137.10 192.168.137.11
Apache Banner Linux Distribution Disclosure	If you do not wish to display this information, edit ’HTTPD.conf’ and set the directive ’ServerTokens Prod’ and restart Apache.	192.168.137.10
Web Server Generic Cookie Injection	Contact the vendor for a patch or upgrade.	192.168.137.10
Missing or Permissive Content-Security-Policy HTTP Response Header	Set a properly configured Content-Security-Policy header for all requested resources.	192.168.137.10

Continued on next page

Table B.4 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
Missing or Permissive X-Frame-Options HTTP Response Header	Set a properly configured X-Frame-Options header for all requested resources.	192.168.137.10
memcached Detection on 11211 port.	If memcached is deployed in untrusted networks, it's recommended that SASL be enabled to restrict access to authorized users.	192.168.137.10
Patch Report	Install the required security patches.	192.168.137.10 192.168.137.11
SSH Server CBC Mode Ciphers Enabled	Contact the vendor or consult product documentation to disable CBC mode cipher encryption, and enable CTR or GCM cipher mode	192.168.137.10 192.168.137.11
SSH Weak MAC Algorithms Enabled	Contact the vendor or consult product documentation to disable MD5 and 96-bit MAC algorithms.	192.168.137.10 192.168.137.11
AMQP Cleartext Authentication	Disable cleartext authentication mechanisms in the AMQP configuration.	192.168.137.10
SSH Weak Algorithms Supported	Contact the vendor or consult product documentation to remove the weak ciphers.	192.168.137.10 192.168.137.11
Multiple BSD ipfw / ip6fw ECE Bit Filtering Evasion	If you are running FreeBSD 3.X, 4.x, 3.5-STABLE, 4.2-STABLE, upgrade your firewall. If you are not running FreeBSD, contact your firewall vendor for a patch.	192.168.137.10
TCP/IP Sequence Prediction Blind Reset Spoofing DoS	Contact the vendor for a patch or mitigation advice.	192.168.137.10 192.168.137.11

Continued on next page

Table B.4 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
Apache 2.4.x < 2.4.8 Multiple Vulnerabilities	Upgrade to Apache version 2.4.9 or later. Alternatively, ensure that the affected modules are not in use. Note that the issues were addressed in 2.4.8, although that version was not released.	192.168.137.10
Apache 2.4.x < 2.4.10 Multiple Vulnerabilities	Upgrade to Apache version 2.4.10 or later. Alternatively, ensure that the affected modules are not in use.	192.168.137.10
OpenSSH SSHFP Record Verification Weakness	Update to version 6.7 or later or apply the vendor patch.	192.168.137.10 192.168.137.11
Apache 2.4.x < 2.4.12 Multiple Vulnerabilities	Upgrade to Apache version 2.4.12 or later. Alternatively, ensure that the affected modules are not in use.	192.168.137.10
Apache 2.4.x < 2.4.12 Multiple Vulnerabilities	Upgrade to Apache version 2.4.12 or later. Alternatively, ensure that the affected modules are not in use.	192.168.137.10
OpenSSH < 6.9 Multiple Vulnerabilities	Upgrade to OpenSSH 6.9 or later.	192.168.137.10 192.168.137.11
Apache 2.4.x < 2.4.16 Multiple Vulnerabilities	Upgrade to Apache version 2.4.16 or later. Alternatively, ensure that the affected modules are not in use.	192.168.137.10
SSH Diffie-Hellman Modulus <= 1024 Bits (Logjam)	Reconfigure the service to use a unique Diffie-Hellman moduli of 2048 bits or greater.	192.168.137.10 192.168.137.11

Continued on next page

Table B.4 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
Web Server HTTP Header Information Disclosure	Modify the HTTP headers of the web server to not disclose detailed information about the underlying web server.	192.168.137.10
OpenSSH < 7.2 Untrusted X11 Forwarding Fallback Security Bypass	Upgrade to OpenSSH version 7.2 or later.	192.168.137.10 192.168.137.11
OpenSSH < 7.2p2 X11Forwarding xauth Command Injection	Upgrade to OpenSSH version 7.2p2 or later.	192.168.137.10 192.168.137.11
OpenSSH < 7.3 Multiple Vulnerabilities	Upgrade to OpenSSH version 7.3 or later.	192.168.137.10 192.168.137.11
OpenSSH < 7.4 Multiple Vulnerabilities	Upgrade to OpenSSH version 7.4 or later.	192.168.137.10 192.168.137.11
OpenSSH < 7.5	Upgrade to OpenSSH version 7.5 or later.	192.168.137.10 192.168.137.11
Apache 2.4.x < 2.4.25 Multiple Vulnerabilities (httpoxy)	Upgrade to Apache version 2.4.25 or later. Note that the 'httpoxy' vulnerability can be mitigated by applying the workarounds or patches as referenced in the vendor advisory asf-httpoxy-response.txt. Furthermore, to mitigate the other vulnerabilities, ensure that the affected modules (mod_session_crypto, mod_auth_digest, and mod_http2) are not in use.	192.168.137.10

Continued on next page

Table B.4 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
Apache 2.2.x < 2.2.33-dev / 2.4.x < 2.4.26 Multiple Vulnerabilities	Upgrade to Apache version 2.2.33-dev / 2.4.26 or later.	
Apache 2.4.x < 2.4.27 Multiple Vulnerabilities	Upgrade to Apache version 2.4.27 or later.	
VNC Software Detection	Make sure use of this software is done in accordance with your organization's security policy and filter incoming traffic to this port.	192.168.137.11
VNC Server Unauthenticated Access	Disable the No Authentication security type.	
VNC Server Unauthenticated Access:Screenshot	Disable the 'No Authentication' security type.	

Table B.4: OpenStack 192.168.137.0/24 Target Vulnerabilities and Solutions suggested by Nessus.

Table B.4 shows the vulnerabilities and Nessus solutions to overcome the security risks occurring within the Cloud Management network (Cloud Nodes). Table B.4 shares some vulnerabilities listed in Table B.3. However, there are other vulnerabilities that did not show up in Table B.3. The following are the vulnerabilities occurring within the Management network targets without repeating those listed in Table B.3:

1. Web Server Generic XSS: a web server can be affected by a cross-site scripting vulnerability.
   a) A remote attacker can exploit a failure to adequately sanitize request strings of malicious JavaScript inputs, to execute arbitrary HTML and script code in a user's browser within the security context of the affected site.
   b) The Controller Cloud Node is required to fix this issue.
2. Apache Banner Linux Distribution Disclosure:

a) The name of the Linux distribution running on the remote host was found in the banner of the web server.

b) Controller Cloud Node is required to fix this issue.

3. Web Server Generic Cookie Injection.

   a) The remote host runs a web server that fails to adequately sanitize request strings of malicious JavaScript.

   b) By leveraging this issue, an attacker may be able to inject arbitrary cookies. Depending on the structure of the web application, it may be possible to launch a 'session fixation' attack using this mechanism.

   c) The Controller Cloud Node is required to fix this issue.

4. Missing or Permissive Content-Security-Policy and X-Frame-Options HTTP Response Header: a web server does not take steps to mitigate a class of web application vulnerabilities.

   a) The remote web server in some responses sets a permissive Content-Security-Policy (CSP) and X-Frame-Options response header or does not set one at all.

   b) The remote web server in some responses sets a permissive response header or does not set one at all.

   c) The CSP header and X-Frame-Options have been proposed by the W3C Web Application Security Working Group as a way to mitigate cross-site scripting and clickjacking attacks.

   d) The Controller Cloud Node is required to fix this issue.

5. Memcached Detection: memcached, a memory-based object store, is running on 11211 port.

   a) If memcached is deployed in untrusted networks, access to authorized users is possible.

   b) The Controller Cloud Node is required to fix this issue.

6. AMQP Cleartext Authentication: The remote host is running a service that allows cleartext authentication.

   a) Advanced Message Queuing Protocol (AMQP) service supports one or more authentication mechanisms that allow credentials to be sent in the clear.

   b) The Controller Cloud Node is required to fix this issue.

7. Multiple BSD ipfw / ip6fw ECE Bit Filtering Evasion.

    a) The host may be vulnerable to a bug wherein a remote attacker can circumvent the firewall by setting the ECE bit within the TCP flags field. At least one firewall (ipfw) is known to exhibit this sort of behaviour.

    b) The Controller Cloud Node is required to fix this issue.

8. TCP/IP Sequence Prediction Blind Reset Spoofing DoS: It was possible to send spoofed RST packets to the remote system.

    a) The host is affected by a sequence number approximation vulnerability that allows an attacker to send spoofed RST packets to the remote host and close established connections. This may cause problems for some dedicated services (BGP, a VPN over TCP, etc).

    b) The Controller and Compute Cloud Nodes are required to fix this issue.

9. Apache 2.4.x < 2.4.8, 2.4.10, 2.4.12, 2.4.16, 2.4.25, 2.4.26, 2.4.27 or 2.2.x < 2.2.33-dev Multiple Vulnerabilities: web server is affected by multiple vulnerabilities.

    a) A flaw exists:

        i. A flaw existswith the ’mod_dav’ module that is caused when tracking the length of CDATA that has leading white space. A remote attacker with a specially crafted DAV WRITE request can cause the service to stop responding. (CVE-2013-6438)

        ii. A flaw exists in ’mod_log_config’ module that is caused when logging a cookie that has an unassigned value. A remote attacker with a specially crafted request can cause the service to crash. (CVE-2014-0098)

        iii. A flaw exists in the ’mod_proxy’ module that may allow an attacker to send a specially crafted request to a server configured as a reverse proxy that may cause the child process to crash. This could potentially lead to a Denial of Service attack. (CVE-2014-0117)

        iv. A flaw exists in the ’mod_deflate’ module when request body decompression is configured. This could allow a remote attacker to cause the server to consume significant resources. (CVE-2014-0118)

        v. A flaw exists in the ’mod_status’ module when a publicly accessible server status page is in place.This could allow an attacker to send a specially crafted request designed to cause a heap buffer overflow. (CVE-2014-0226)

vi. A flaw exists in the 'mod_cgid' module in which CGI scripts that did not consume standard input may be manipulated in order to cause child processes to hang. A remote attacker may be able to abuse this in order to cause a Denial of Service. (CVE-2014-0231)

vii. A flaw exists in WinNT MPM versions 2.4.1 to 2.4.9 when using the default AcceptFilter. An attacker may be able to specially craft requests that create a memory leak in the application and may eventually lead to a Denial of Service attack. (CVE-2014-3523)

viii. A flaw exists in module mod_headers that can allow HTTP trailers to replace HTTP headers late during request processing, which a remote attacker can exploit to inject arbitrary headers. This can also cause some modules to function incorrectly or appear to function incorrectly. (CVE-2013-5704)

b) A NULL pointer dereference flaw exists in module mod_cache. A remote attacker, using an empty HTTP Content-Type header, can exploit this vulnerability to crash a caching forward proxy configuration, resulting in a Denial of Service attack if using a threaded MPM. (CVE-2014-3581)

c) A out-of-bounds memory read flaw exists in module mod_proxy_fcgi. An attacker, using a remote FastCGI server to send long response headers, can exploit this vulnerability to cause a Denial of Service by causing a buffer over-read. (CVE-2014-3583)

d) A flaw exists in module mod_lua when handling a LuaAuthzProvider used in multiple Require directives with different arguments. An attacker can exploit this vulnerability to bypass intended access restrictions. (CVE-2014-8109)

e) A flaw exists in the lua_websocket_read() function in the 'mod_lua' module due to incorrect handling of WebSocket PING frames. A remote attacker can exploit this, by sending a crafted WebSocket PING frame after a Lua script has called the wsupgrade() function to crash a child process, resulting in a Denial of Service condition. (CVE-2015-0228)

f) A NULL pointer de-reference flaw exists in the read_request_line() function due to a failure to initialize the protocol structure member. A remote attacker can exploit this flaw, on installations that enable the INCLUDES filter and has an ErrorDocument 400 directive specifying a local URI, by sending a request that lacks a method to cause a Denial of Service condition. (CVE-2015-0253)

g) A flaw exists in the chunked transfer coding implementation due to a failure to properly parse chunk headers. A remote attacker can exploit this to conduct HTTP request smuggling attacks. (CVE-2015-3183)

h) A flaw exists in the ap_some_auth_required() function due to a failure to consider that a Require directive may be associated with an authorization setting rather than an authentication setting. A remote attacker can exploit this, if a module that relies on the 2.2 API behaviour exists, to bypass intended access restrictions. (CVE-2015-3185)

i) A flaw exists in the RC4 algorithm due to an initial double-byte bias in the keystream generation. An attacker can exploit this, via Bayesian analysis that combines an a priori plaintext distribution with keystream distribution statistics, to conduct a plaintext recovery of the ciphertext. Note that RC4 cipher suites are prohibited under RFC 7465. This issue was fixed in Apache version 2.4.13; however, 2.4.13, 2.4.14, and 2.4.15 were never publicly released. (VulnDB 128186)

j) A flaw exists in the mod_session_crypto module due to encryption for data and cookies using the configured ciphers with possibly either CBC or ECB modes of operation (AES256-CBC by default). An unauthenticated, remote attacker can exploit this, via a padding oracle attack, to decrypt information without knowledge of the encryption key, resulting in the disclosure of potentially sensitive information. (CVE-2016-0736)

k) A Denial of Service vulnerability exists in the mod_auth_digest module during client entry allocation. An unauthenticated, remote attacker can exploit this, via specially crafted input, to exhaust shared memory resources, resulting in a server crash. (CVE-2016-2161)

l) The Apache HTTP Server is affected by a Man-in-the-Middle vulnerability known as ’httpoxy’ due to a failure to properly resolve namespace conflicts in accordance with RFC 3875 section 4.1.18. The HTTP_PROXY environment variable is set based on untrusted user data in the ’Proxy’ header of HTTP requests. The HTTP_PROXY environment variable is used by some web client libraries to specify a remote proxy server. An unauthenticated, remote attacker can exploit this, via a crafted ’Proxy’ header in an HTTP request, to redirect an application’s internal HTTP traffic to an arbitrary proxy server where it may be observed or manipulated. (CVE-2016-5387)

m) A Denial of Service vulnerability exists in the mod_http2 module due to improper handling of the LimitRequestFields directive. An unauthenticated, remote attacker can exploit this, via specially crafted CONTINUATION frames in an HTTP/2 request, to inject unlimited request headers into the server, resulting in the exhaustion of memory resources. (CVE-2016-8740)

n) A flaw exists due to improper handling of whitespace patterns in user-agent headers. An unauthenticated, remote attacker can exploit this, via a specially crafted user-agent header, to cause the program to incorrectly process sequences of requests, resulting in interpreting responses incorrectly, polluting the cache, or disclosing the content from one request to a second downstream user-agent. (CVE-2016-8743)

o) A Denial of Service vulnerability exists in HTTPD due to a failure to initialize or reset the value placeholder in [Proxy-]Authorization headers of type 'Digest' before or between successive key=value assignments by mod_auth_digest. An unauthenticated, remote attacker can exploit this, by providing an initial key with no '=' assignment, to disclose sensitive information or cause a Denial of Service condition. (CVE-2017-9788)

p) A read-after-free error exists in HTTPD that is triggered when closing a large number of connections. An unauthenticated, remote attacker can exploit this to have an unspecified impact. (CVE-2017-9789)

q) An authentication bypass vulnerability exists due to third-party modules using the ap_-get_basic_auth_pw() function outside of the authentication phase. An unauthenticated, remote attacker can exploit this to bypass authentication requirements. (CVE-2017-3167)

r) A NULL pointer dereference flaw exists due to third-party module calls to the mod_ssl ap_hook_process_connection() function during an HTTP request to an HTTPS port. An unauthenticated, remote attacker can exploit this to cause a Denial of Service condition. (CVE-2017-3169)

s) A NULL pointer de-reference flaw exists in mod_http2 that is triggered when handling a specially crafted HTTP/2 request. An unauthenticated, remote attacker can exploit this to cause a Denial of Service condition. Note that this vulnerability does not affect 2.2.x. (CVE-2017-7659)

t) An out-of-bounds read error exists in the ap_find_token() function due to improper handling of header sequences. An unauthenticated, remote attacker can exploit this, via a specially crafted header sequence, to cause a Denial of Service condition. (CVE-2017-7668)

u) An out-of-bounds read error exists in mod_mime due to improper handling of Content-

Type response headers. An unauthenticated, remote attacker can exploit this, via a specially crafted Content-Type response header, to cause a Denial of Service condition or the disclosure of sensitive information. (CVE-2017-7679)

v) The Controller Cloud Node is required to fix this issue.

10. OpenSSH SSHFP Record Verification Weakness: A secure shell client on the remote host could be used to bypass host verification methods.

    a) Affected by a host verification bypass vulnerability related to SSHFP and certificates that could allow a malicious SSH server to cause the supplied client to inappropriately trust the server.

    b) The Controller and Compute Cloud nodes are required to fix this issue.

11. OpenSSH < 6.9, 7.0, 7.3, 7.4 or 7.5 Multiple Vulnerabilities: The SSH server running on the remote host is affected by information disclosure vulnerability.

    a) A flaw exists within the x11_open_helper() function in the 'channels.c' file that allows connections to be permitted after 'ForwardX11Timeout' has expired. A remote attacker can exploit this to bypass timeout checks and XSECURITY restrictions. (CVE-2015-5352)

    b) Various issues were addressed by fixing the weakness in agent locking by increasing the failure delay, storing the salted hash of the password, and using a timing-safe comparison function.

    c) An out-of-bounds read error exists when handling incorrect pattern lengths. A remote attacker can exploit this to cause a Denial of Service or disclose sensitive information in the memory.

    d) An out-of-bounds read error exists when parsing the 'EscapeChar' configuration option.

    e) A security bypass vulnerability exists in the kbdint_next_device() function in file auth2-chall.c that allows the circumvention of MaxAuthTries during keyboard-interactive authentication. A remote attacker can exploit this issue to force the same authentication method to be tried thousands of times in a single pass by using a crafted keyboard-interactive 'devices' string, thus allowing a brute-force attack or causing a Denial of Service. (CVE-2015-5600)

    f) A security bypass vulnerability exists in SSHD due to improper handling of username data in MONITOR_REQ_PAM_INIT_CTX requests. A local attacker can exploit this, by sending a MONITOR_REQ_PWNAM request, to conduct an impersonation attack. Note that this issue only affects Portable OpenSSH. (CVE-2015-6563)

g) A privilege escalation vulnerability exists due to a use-after-free error in SSHD that is triggered when handling a MONITOR_REQ_PAM_FREE_CTX request. A local attacker can exploit this to gain elevated privileges. Note that this issue only affects Portable OpenSSH. (CVE-2015-6564)

h) A local command execution vulnerability exists in SSHD due to setting insecure world-writable permissions for TTYs. A local attacker can exploit this, by injecting crafted terminal escape sequences, to execute commands for logged-in users. (CVE-2015-6565)

i) A flaw exists that is due to the program returning shorter response times for authentication requests with overly long passwords for invalid users than for valid users. This may allow a remote attacker to conduct a timing attack and enumerate valid usernames. (CVE-2016-6210)

j) A Denial of Service vulnerability exists in the auth_password() function in auth-passwd.c due to a failure to limit password lengths for password authentication. An unauthenticated, remote attacker can exploit this, via a long string, to consume excessive CPU resources, resulting in a Denial of Service condition. (CVE-2016-6515)

k) An unspecified flaw exists in the CBC padding oracle countermeasures that allows an unauthenticated, remote attacker to conduct a timing attack. (VulnDB 142343)

l) A flaw exists due to improper operation ordering of MAC verification for Encrypt-then-MAC (EtM) mode transport MAC algorithms when verifying the MAC before decrypting any ciphertext. An unauthenticated, remote attacker can exploit this, via a timing attack, to disclose sensitive information. (VulnDB 142344)

m) A flaw exists in SSH-agent due to loading PKCS#11 modules from paths that are outside a trusted whitelist. A local attacker can exploit this by using a crafted request to load hostile modules via agent forwarding, to execute arbitrary code. To exploit this vulnerability, the attacker would need to control the forwarded agent-socket (on the host running the SSHD server) and the ability to write to the file system of the host running ssh-agent. (CVE-2016-10009)

n) A flaw exists in SSHD due to creating forwarded Unix-domain sockets with 'root' privileges whenever privilege separation is disabled. A local attacker can exploit this to gain elevated privileges. (CVE-2016-10010)

o) An information disclosure vulnerability exists in SSHD within the realloc() function due to leakage of key material to privilege-separated child processes when reading keys. A local attacker can possibly exploit this to disclose sensitive key material. Note that no such leak has been observed in practice for normal-sized keys, nor does a leak to the child processes directly expose key material to unprivileged users. (CVE-2016-10011)

p) A flaw exists in SSHD within the shared memory manager used by pre-authenticating compression support due to a bounds check being elided by some optimizing compiler and due to the memory manager being incorrectly accessible when pre-authenticating compression is disabled. A local attacker can exploit this to gain elevated privileges. (CVE-2016-10012)

q) A Denial of Service vulnerability exists in SSHD when handling KEXINIT messages. An unauthenticated, remote attacker can exploit this, by sending multiple KEXINIT messages, to consume up to 128MB per connection. (VulnDB 148976)

r) A flaw exists in SSHD due to improper validation of address ranges by the AllowUser and DenyUsers directives at configuration load time. A local attacker can exploit this, via an invalid CIDR address range, to gain access to restricted areas. (VulnDB 148977)

s) An unspecified timing flaw exists in the CBC padding oracle countermeasures, within the SSH and SSHD functions, that allows an unauthenticated, remote attacker to disclose potentially sensitive information. Note that the OpenSSH client disables CBC ciphers by default. However, SSHD offers them as lowest-preference options, which will be removed by default in a future release. (VulnDB 144000)

t) The Controller and Compute Cloud nodes are required to fix this issue.

12. Web Server HTTP Header Information Disclosure: The remote web server discloses information via HTTP headers.

    a) The HTTP headers sent by the remote web server disclose information that can aid an attacker, such as the server version and languages used by the web server.

    b) The Controller Cloud node is required to fix this issue.

13. OpenSSH < 7.2 Untrusted X11 Forwarding Fallback Security Bypass: The SSH server running on the remote host is affected by a security bypass vulnerability.

    a) affected by a security bypass vulnerability due to a flaw in SSH(1) that is triggered when it falls back from untrusted X11 forwarding to trusted forwarding when the SECURITY extension is disabled by the X server.

b) This can result in untrusted X11 connections that can be exploited by a remote attacker.

c) The Controller and Compute Cloud Nodes are required to fix this issue.

14. OpenSSH is earlier than 7.2p2 and X11Forwarding xauth Command Injection: The SSH server running on the remote host is affected by a security bypass vulnerability.

    a) affected by a security bypass vulnerability due to improper sanitization of X11 authentication credentials.

    b) An authenticated, remote attacker can exploit this, via crafted credentials, to inject arbitrary xauth commands, resulting in gaining read and write access to arbitrary files, connecting to local ports, or performing further attacks on xauth itself. Note that exploiting this vulnerability requires X11Forwarding to have been enabled.

    c) The Controller and Compute Cloud nodes are required to fix this issue.

15. VNC Software Detection:

    a) The remote host is running VNC (Virtual Network Computing), which uses the RFB (Remote FrameBuffer) protocol to provide remote access to graphical user interfaces and thus permits a console on the remote host to be displayed on another.

    b) The Compute Cloud node is required to fix this issue.

16. VNC Server Unauthenticated Access: VNC server does not require authentication.

    a) The VNC server installed on the remote host allows an attacker to connect to the remote host as no authentication is required to access this service. The VNC server sometimes sends the connected user to the XDM login screen. Unfortunately, Nessus cannot identify this situation. In such a case, it is not possible to go further without valid credentials and this alert may be ignored.

    b) The Compute Cloud node is required to fix this issue.

17. VNC Server Unauthenticated Access: Screenshot: VNC server does not require authentication

    a) The VNC server installed on the remote host allows an attacker to connect to the remote host as no authentication is required to access this service.

    b) It was possible to log into the remote service and take a screenshot.

   c) The Compute Cloud node is required to fix this issue.

18. TCP/IP Predictable ISN (Initial Sequence Number) Generation Weakness: It is possible to predict TCP/IP Initial Sequence Numbers for the remote host

    a) The remote host has predictable TCP sequence numbers.

    b) An attacker may use this flaw to establish spoofed TCP connections to this host.

    c) The Compute and Controller Cloud nodes are required to fix this issue.

The Nessus solutions suggested for CloudStack testbed are:

Vulnerability Name	**Solution**	**Targets**
Nessus SYN scanner, Nessus TCP scanner	Protect your target with an IP filter.	192.168.31.100 192.168.31.101 192.168.31.11 192.168.31.12 192.168.31.121
Nessus UDP Scanner	Protect your target with an IP filter and implement ICMP rate imitation.	192.168.31.100 192.168.31.101 192.168.31.102 192.168.31.103 192.168.31.104 192.168.31.11 192.168.31.12 192.168.31.121 192.168.31.125 192.168.31.129
Patch Report	Install the required security patches.	192.168.31.100 192.168.31.101 192.168.31.102 192.168.31.103 192.168.31.104 192.168.31.11 192.168.31.12 192.168.31.121 192.168.31.129 192.168.31.125
HSTS Missing From HTTPS Server	Configure the remote web server to use HSTS.	192.168.31.101
SSL Certificate Expiry	Purchase or generate a new SSL certificate to replace the existing one.	192.168.31.101

Continued on next page

Table B.5 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
SSL Version 2 and 3 Protocol Detection	Consult the application's documentation to disable SSL 2.0 and 3.0. Use TLS 1.1 (with approved cipher suites) or higher instead.	192.168.31.101
SSL Certificate Signed Using Weak Hashing Algorithm	Contact the Certificate Authority to have the certificate reissued.	192.168.31.101
SSL Medium Strength Cipher Suites Supported	Reconfigure the affected application if possible to avoid use of medium strength ciphers.	192.168.31.101 192.168.31.11
SSL Certificate Cannot Be Trusted	Purchase or generate a proper certificate for this service.	192.168.31.101 192.168.31.11
SSL/TLS Protocol Initialization Vector Implementation Information Disclosure Vulnerability (BEAST)	Configure SSL/TLS servers to only use TLS 1.1 or TLS 1.2 if supported. Configure SSL/TLS servers to only support cipher suites that do not use block ciphers. Apply patches if available. Note that additional configuration may be required after the installation of the MS12-006 security update in order to enable the split-record countermeasure.	192.168.31.101
SSL RC4 Cipher Suites Supported (Bar Mitzvah)	Reconfigure the affected application, if possible, to avoid use of RC4 ciphers. Consider using TLS 1.2 with AES-GCM suites subject to browser and web server support.	192.168.31.101

Continued on next page

Table B.5 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
SSLv3 Padding Oracle On Downgraded Legacy Encryption Vulnerability (POODLE)	Disable SSLv3. Services that must support SSLv3 should enable the TLS Fallback SCSV mechanism until SSLv3 can be disabled.	192.168.31.101
SSL Root Certification Authority Certificate Information	Ensure that use of this root Certification Authority certificate complies with your organization's acceptable use and security policies.	192.168.31.101
SSL Certificate Signed Using Weak Hashing Algorithm (Known CA)	Contact the Certificate Authority to have the certificate reissued.	192.168.31.101
TLS Version 1.0 Protocol Detection	Enable support for TLS 1.1 and 1.2, and disable support for TLS 1.0.	192.168.31.101 192.168.31.11
SSL/TLS Diffie-Hellman Modulus <=1024 Bits (Logjam)	Reconfigure the service to use a unique Diffie-Hellman moduli of 2048 bits or greater.	192.168.31.101 192.168.31.11
SSL 64-bit Block Size Cipher Suites Supported (SWEET32)	Reconfigure the affected application, if possible, to avoid use of all 64-bit block ciphers. Alternatively, place limitations on the number of requests that are allowed to be processed over the same TLS connection to mitigate this vulnerability.	192.168.31.101
ICMP Timestamp Request Remote Date Disclosure	Filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).	192.168.31.102 192.168.31.103 192.168.31.104 192.168.31.11 192.168.31.12
Web Server Transmits Cleartext Credentials	Make sure that every sensitive form transmits content over HTTPS.	192.168.31.11

Continued on next page

Table B.5 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
Web Application Potentially Sensitive CGI Parameter Detection	Ensure sensitive data is not disclosed by CGI parameters. In addition, do not use CGI parameters to control access to resources or privileges.	192.168.31.11
Missing or Permissive Content-Security-Policy frame-ancestors HTTP Response Header	Set a non-permissive Content-Security-Policy frame-ancestors header for all requested resources.	192.168.31.11
Missing or Permissive X-Frame-Options HTTP Response Header	Set a properly configured X-Frame-Options header for all requested resources.	192.168.31.11
Citrix CloudPlatform Default Credentials	Change the default 'admin' login credentials.	192.168.31.11
Web Application Potentially Vulnerable to Clickjacking	Return the X-Frame-Options or Content-Security-Policy (with the 'frame-ancestors' directive) HTTP header with the page's response. This prevents the page's content from being rendered by another site when using the frame or iframe HTML tags.	192.168.31.11
Web Application Cookies Not Marked Secure	Each cookie should be carefully reviewed to determine if it contains sensitive data or is relied upon for a security decision. If possible, ensure all communication occurs over an encrypted channel and add the 'secure' attribute to all session cookies or any cookies containing sensitive data.	192.168.31.11

Continued on next page

Table B.5 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
NFS Exported Share Information Disclosure	Configure NFS on the remote host so that only authorized hosts can mount its remote shares.	192.168.31.11
NFS Shares World Readable	Place the appropriate restrictions on all NFS shares.	192.168.31.11
SSL Self-Signed Certificate	Purchase or generate a proper certificate for this service.	192.168.31.11
SSH Server CBC Mode Ciphers Enabled	Contact the vendor or consult product documentation to disable CBC mode cipher encryption, and enable CTR or GCM cipher mode encryption.	192.168.31.11 192.168.31.12
SSH Weak MAC Algorithms Enabled	Contact the vendor or consult product documentation to disable MD5 and 96-bit MAC algorithms.	192.168.31.11 192.168.31.12
SSH Weak Algorithms Supported	Contact the vendor or consult product documentation to remove the weak ciphers.	192.168.31.11 192.168.31.12
OpenSSH SSHFP Record Verification Weakness	Update to version 6.7 or later or apply the vendor patch.	192.168.31.11 192.168.31.12
OpenSSH < 6.9 Multiple Vulnerabilities	Upgrade to OpenSSH 6.9 or later.	192.168.31.11 192.168.31.12
OpenSSH < 7.0 Multiple Vulnerabilities	Upgrade to OpenSSH 7.0 or later.	192.168.31.11 192.168.31.12
OpenSSH < 7.2 Untrusted X11 Forwarding Fallback Security Bypass	Upgrade to OpenSSH version 7.2 or later.	192.168.31.11 192.168.31.12
OpenSSH < 7.2p2 X11 Forwarding xauth Command Injection	Upgrade to OpenSSH version 7.2p2 or later.	192.168.31.11 192.168.31.12

Continued on next page

Table B.5 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
OpenSSH < 7.3 Multiple Vulnerabilities	Upgrade to OpenSSH version 7.3 or later.	192.168.31.11 192.168.31.12
OpenSSH < 7.4 Multiple Vulnerabilities	Upgrade to OpenSSH version 7.4 or later.	192.168.31.11 192.168.31.12
OpenSSH < 7.5	Upgrade to OpenSSH version 7.5 or later.	192.168.31.11 192.168.31.12
OpenSSH < 7.6	Upgrade to OpenSSH version 7.6 or later.	192.168.31.11 192.168.31.12
VNC Software Detection	Make sure use of this software is done in accordance with your organization's security policy and filter incoming traffic to this port.	192.168.31.12

Table B.5: CloudStack 192.168.31.0/24 Target Vulnerabilities and Solutions Suggested by Nessus.

Table B.5 consists of vulnerabilities detected in the CloudStack cloud. However, there are some vulnerabilities similar to both OpenStack and CloudStack clouds as is shown in Tables B.3, B.4 and B.5. The following are the newly displayed vulnerabilities that did not occur in OpenStack, but occurred in CloudStack:

1. HSTS Missing From HTTPS Server: web server is not enforcing HSTS.
    a) The remote HTTPS server is not enforcing HTTP Strict Transport Security (HSTS). The lack of HSTS allows downgrade attacks; SSL-stripping Man-in-the-Middle attacks, and weakens cookie-hijacking protections.
    b) The Secondary Storage System VM is required to fix this issue.
2. SSL Certificate Expiry:
    a) The remote server's SSL certificate has already expired.
    b) The Secondary Storage System VM is required to fix this issue.

3. SSL Version 2 and 3 Protocol Detection: The remote service encrypts traffic using a protocol with known weaknesses.

   a) The remote service accepts connections encrypted using SSL 2.0 and/or SSL 3.0. These versions of SSL are affected by several cryptographic flaws, including:

      i. An insecure padding scheme with CBC ciphers.

      ii. Insecure session renegotiation and resumption schemes.

   b) An attacker can exploit these flaws to conduct Man-in-the-Middle attacks or to decrypt communications between the affected service and clients.

   c) Although SSL/TLS has a secure means for choosing the highest supported version of the protocol (so that these versions will be used only if the client or server support nothing better), many web browsers implement this in an unsafe way that allows an attacker to downgrade a connection (such as in POODLE). Therefore, it is recommended that these protocols be disabled entirely.

   d) NIST has determined that SSL 3.0 is no longer acceptable for secure communications. As of the date of enforcement found in PCI DSS v3.1, any version of SSL will not meet the PCI SSC's definition of 'strong cryptography'.

   e) The Secondary Storage System VM is required to fix this issue.

4. SSL Certificate Signed Using Weak Hashing Algorithm: An SSL certificate in the certificate chain has been signed using a weak hash algorithm

   a) The remote service uses an SSL certificate chain that has been signed using a cryptographically weak hashing algorithm (e.g. MD2, MD4, MD5, or SHA1). These signature algorithms are known to be vulnerable to collision attacks. An attacker can exploit this to generate another certificate with the same digital signature, allowing an attacker to masquerade as the affected service.

   b) Note that this plugin reports all SSL certificate chains signed with SHA-1 that expire after January 1, 2017 as vulnerable. This is in accordance with Google's gradual sunsetting of the SHA-1 cryptographic hash algorithm.

   c) Note that certificates in the chain that are contained in the Nessus CA database (known_-CA.inc) have been ignored.

   d) The Secondary Storage System VM is required to fix this issue.

5. SSL Medium Strength Cipher Suites Supported: The remote service supports the use of medium strength SSL ciphers.
    a) The remote host supports the use of SSL ciphers that offer medium strength encryption. Nessus regards medium strength as any encryption that uses key lengths at least 64 bits and less than 112 bits, or else that uses the 3DES encryption suite.
    b) Note that it is considerably easier to circumvent medium strength encryption if the attacker is on the same physical network.
    c) The Secondary Storage System VM is required to fix this issue.
6. SSL Certificate Cannot Be Trusted: The SSL certificate for this service cannot be trusted.
    a) The server's X.509 certificate cannot be trusted. This situation can occur in three different ways, in which the chain of trust can be broken, as stated below:
        i. First, the top of the certificate chain sent by the server might not be descended from a known public certificate authority. This can occur either when the top of the chain is an unrecognized, self-signed certificate, or when intermediate certificates are missing and that would connect the top of the certificate chain to a known public certificate authority, but not the bottom.
        ii. Second, the certificate chain may contain a certificate that is not valid at the time of the scan. This can occur either when the scan occurs before one of the certificate's 'notBefore' dates, or after one of the certificate's 'notAfter' dates.
        iii. Third, the certificate chain may contain a signature that either did not match the certificate's information or could not be verified. Bad signatures can be fixed by getting the certificate with the bad signature to be re-signed by its issuer. Signatures that could not be verified are the result of the certificate's issuer using a signing algorithm that Nessus either does not support or does not recognize.
    b) If the remote host is a public host in production, any break in the chain makes it more difficult for users to verify the authenticity and identity of the web server. This could make it easier to carry out Man-in-the-Middle attacks against the remote host.
    c) The Secondary Storage System VM is required to fix this issue.
7. SSL/TLS Protocol Initialization Vector Implementation Information Disclosure Vulnerability (BEAST): It may be possible to obtain sensitive information from the remote host with SSL/TLS-enabled services.

a) A vulnerability exists in SSL 3.0 and TLS 1.0 that could allow information disclosure if an attacker intercepts encrypted traffic served from an affected system. TLS 1.1, TLS 1.2, and all cipher suites that do not use CBC mode are not affected.

b) This plugin tries to establish an SSL/TLS remote connection using an affected SSL version and cipher suite and then solicits return data. If the returned application data is not fragmented with an empty or one-byte record, it is likely vulnerable. OpenSSL uses empty fragments as a countermeasure unless the 'SSL_OP_DONT_INSERT_EMPTY_-FRAGMENTS' option is specified when OpenSSL is initialized.

c) Therefore, if multiple applications use the same SSL/TLS implementation, some may be vulnerable while others may not be, depending on whether or not a countermeasure has been enabled.

d) Note that this plugin detects the vulnerability in the SSLv3/TLSv1 protocol implemented in the server. It does not detect the BEAST attack where it exploits the vulnerability at the HTTPS client-side (i.e., Internet browser). The detection at the server-side does not necessarily mean your server is vulnerable to the BEAST attack, because the attack exploits the vulnerability at the client-side, and both SSL/TLS clients and servers can independently employ the split record countermeasure.

e) The Secondary Storage System VM is required to fix this issue.

8. SSL RC4 Cipher Suites Supported (Bar Mitzvah): The remote service supports the use of the RC4 cipher.

   a) The remote host supports the use of RC4 in one or more cipher suites. The RC4 cipher is flawed in its generation of a pseudo-random stream of bytes so that a wide variety of small biases are introduced into the stream, decreasing its randomness.

   b) If plaintext is repeatedly encrypted (e.g., HTTP cookies), and an attacker is able to obtain many (i.e., tens of millions) ciphertexts, the attacker may be able to derive the plaintext.

   c) The Secondary Storage System VM is required to fix this issue.

9. SSLv3 Padding Oracle On Downgraded Legacy Encryption Vulnerability (POODLE): It is possible to obtain sensitive information from the remote host with SSL/TLS-enabled services.

   a) The remote host is affected by a Man-in-the-Middle (MitM) information disclosure vulnerability known as POODLE. The vulnerability is due to the way SSL 3.0 handles padding bytes when decrypting messages encrypted using block ciphers in cipher block chaining (CBC) mode. MitM attackers can decrypt a selected byte of a cipher text in as few as 256 tries if they are able to force a victim application to repeatedly send the same data over newly created SSL 3.0 connections.

b) As long as a client and service both support SSLv3, a connection can be 'rolled back' to SSLv3, even if TLSv1 or newer is supported by the client and service.

c) The TLS Fallback SCSV mechanism prevents 'version rollback' attacks without impacting legacy clients; however, it can only protect connections when the client and service support the mechanism. Sites that cannot disable SSLv3 immediately should enable this mechanism.

d) This is a vulnerability in the SSLv3 specification, not in any particular SSL implementation. Disabling SSLv3 is the only way to completely mitigate the vulnerability.

e) The Secondary Storage System VM is required to fix this issue.

10. SSL Root Certification Authority Certificate Information: A root Certification Authority certificate was found at the top of the certificate chain.

    a) The remote service uses an SSL certificate chain that contains a self-signed root Certification Authority certificate at the top of the chain.

    b) The Secondary Storage System VM is required to fix this issue.

11. SSL Certificate Signed Using Weak Hashing Algorithm (Known CA): A known CA SSL certificate in the certificate chain has been signed using a weak hashing algorithm.

    a) The remote service uses a known CA certificate in the SSL certificate chain that has been signed using a cryptographically weak hashing algorithm (e.g., MD2, MD4, MD5, or SHA1). These signature algorithms are known to be vulnerable to collision attacks. An attacker can exploit this to generate another certificate with the same digital signature, allowing the attacker to masquerade as the affected service.

    b) Note that this plugin reports all SSL certificate chains signed with SHA-1 that expire after January 1, 2017 as vulnerable. This is in accordance with Google's gradual sunsetting of the SHA-1 cryptographic hash algorithm.

    c) The Secondary Storage System VM is required to fix this issue.

12. TLS Version 1.0 Protocol Detection: The remote service encrypts traffic using an older version of TLS.

    a) The remote service accepts connections encrypted using TLS 1.0. TLS 1.0 has a number of cryptographic design flaws. Modern implementations of TLS 1.0 mitigate these problems, but newer versions of TLS such as 1.1 and 1.2 are designed against these flaws and should be used whenever possible.

    b) PCI DSS v3.1 requires that TLS 1.0 be disabled entirely by June 2018, except for point-of-sale terminals and their termination points.

    c) The Secondary Storage System VM is required to fix this issue.

13. SSL 64-bit Block Size Cipher Suites Supported (SWEET32): The remote service supports the use of 64-bit block ciphers

    a) The remote host supports the use of a block cipher with 64-bit blocks in one or more cipher suites. It is affected by a vulnerability, known as SWEET32, due to the use of weak 64-bit blockciphers.

    b) A Man-in-the-Middle attacker who has sufficient resources can exploit this vulnerability, via a 'birthday' attack, to detect a collision that leaks the XOR between the fixed secret and a known plaintext, allowing the disclosure of the secret text, such as secure HTTPS cookies, and possibly resulting in the hijacking of an authenticated session. Proof-of-concepts have shown that attackers can recover authentication cookies from an HTTPS session in as little as 30 hours.

    c) Note that the ability to send a large number of requests over the same TLS connection between the client and server is an important requirement for carrying out this attack. If the number of requests allowed for a single connection were limited, this would mitigate the vulnerability. This plugin requires report paranoia as Nessus has not checked for such mitigation.

    d) The Secondary Storage System VM is required to fix this issue.

14. Web Server Transmits Cleartext Credentials: The remote web server might transmit credentials in cleartext.

    a) The remote web server contains several HTML form fields containing an input of type 'password', which transmit their information to a remote web server in cleartext.

    b) An attacker eavesdropping the traffic between web browser and server may obtain logins and passwords of valid users.

c) The Management cloud Node is required to fix this issue.

15. Citrix CloudPlatform Default Credentials: The application on the remote web server uses a default set of known credentials.

    a) The remote Citrix CloudPlatform web administration interface uses a known set of default credentials.

    b) The Management cloud Node is required to fix this issue.

16. Web Application Potentially Vulnerable to Clickjacking: The remote web server may fail to mitigate a class of web application vulnerabilities.

    a) The remote web server does not set an X-Frame-Options response header or a Content-Security-Policy 'frame-ancestors' response header in all content responses.

    b) This could potentially expose the site to a clickjacking or UI redress attack, in which an attacker can trick a user into clicking an area of the vulnerable page that is different than what the user perceives the page to be. This can result in a user performing fraudulent or malicious transactions.

    c) X-Frame-Options have been proposed by Microsoft as a way to mitigate clickjacking attacks and is currently supported by all major browser vendors.

    d) A Content-Security-Policy (CSP) has been proposed by the W3C Web Application Security Working Group, with increasing support among all major browser vendors, as a way to mitigate clickjacking and other attacks. The 'frame-ancestors' policy directive restricts which sources can embed the protected resource.

    e) Note that while the X-Frame-Options and Content-Security-Policy response headers are not the only mitigations for clickjacking, they are currently the most reliable methods that can be detected throughautomation. Therefore, this plugin may produce false positives if other mitigation strategies (e.g., frame-busting JavaScript) are deployed or if the page does not perform any security-sensitive transactions. The Management cloud Node is required to fix this issue.

17. Web Application Cookies Not Marked Secure: HTTP session cookies might be transmitted in cleartext.

    a) The remote web application sets various cookies throughout a user's unauthenticated and authenticated session. However, there are instances where the application is running over unencrypted HTTP or the cookies are not marked 'secure', meaning the browser could send them back over an unencrypted link under certain circumstances. As a result, it may be possible for a remote attacker to intercept these cookies.

   b) Note that this plugin detects all general cookies missing the 'secure' cookie flag, whereas plugin 49218 (Web Application Session Cookies Not Marked Secure) will only detect session cookies from an authenticated session missing the secure cookie flag.

   c) The Management cloud Node is required to fix this issue.

18. Web Application Potentially Sensitive CGI Parameter Detection: An application was found that may use CGI parameters to control sensitive information.

   a) According to their names, some CGI parameters may control sensitive data (e.g., ID, privileges, commands, prices, credit card data, etc.). In the course of using an application, these variables may disclose sensitive data or be prone to tampering that could result in privilege escalation.

   b) These parameters should be examined to determine what type of data is controlled and if it poses a security risk.

   c) The Management cloud Node is required to fix this issue.

19. NFS Exported Share Information Disclosure: It is possible to access NFS shares on the remote host.

   a) At least one of the NFS shares exported by the remote server could be mounted by the scanning host.

   b) An attacker may be able to leverage this to read (and possibly write) files on remote host.

   c) The Management cloud Node is required to fix this issue.

20. NFS Shares World Readable: The remote NFS server exports world-readable shares.

   a) The remote NFS server is exporting one or more shares without restricting access (based on hostname, IP, or IP range).

   b) The Management cloud Node issue.

21. OpenSSH < 7.6: The SSH server running on the remote host is affected by file creation restriction bypass vulnerability.

    a) According to its banner, the version of OpenSSH running on the remote host is prior to 7.6. It is, therefore, affected by a file creation restriction bypass vulnerability related to the 'process_open' function in the file 'sftp-server.c' that allows authenticated users to create zero-length files regardless of configuration.

    b) Note that Nessus has not tested for these issues but has instead relied only on the application's self-reported version number.

    c) The Management and Host cloud Nodes are required to fix this issue.

Vulnerability Name	**Solution**	**Targets**
Nessus SYN scanner, Nessus TCP scanner	Protect your target with an IP filter.	192.168.1.1 192.168.1.62 192.168.1.73
Nessus UDP Scanner	Protect your target with an IP filter implement ICMP rate imitation.	192.168.1.1 192.168.1.73
Patch Report	Install the required security patches.	192.168.1.1 192.168.1.62 192.168.1.73
ICMP Timestamp Request Remote Date Disclosure	Filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).	192.168.1.1 192.168.1.73
DHCP Server Detection	Apply filtering to keep this information off the network and remove any options that are not in use.	192.168.1.1
DNS Server Detection	Disable this service if it is not needed or restrict access to internal hosts only if the service is available externally.	192.168.1.1
DNS Server Cache Snooping Remote Information Disclosure	Contact the vendor of the DNS software for a fix.	192.168.1.1

Continued on next page

Table B.6 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
DNS Server Spoofed Request Amplification DDoS	Restrict access to your DNS server from public network or reconfigure it to reject such queries.	192.168.1.1
dnsmasq < 2.63test1 libvirtd TCP Network Packet Parsing Response DNS Amplification DoS	Upgrade to DNSMASQ 2.63test1 or later.	192.168.1.1
dnsmasq < 2.66test2 libvirtd TCP Network Packet Parsing Response DNS Amplification DoS	Upgrade to dnsmasq 2.66test2 or later.	192.168.1.1
dnsmasq < 2.73rc4 setup_reply() Function Return Value Checking Information Disclosure	Upgrade to dnsmasq 2.73rc4 or later.	192.168.1.1
dnsmasq < 2.78 Multiple Remote Vulnerabilities	Upgrade to dnsmasq 2.78 or later.	192.168.1.1
HSTS Missing From HTTPS Server	Configure the remote web server to use HSTS.	192.168.1.62
Missing or Permissive Content-Security-Policy frame-ancestors HTTP Response Header	Set a non-permissive Content-Security-Policy frame-ancestors header for all requested resources.	192.168.1.62
Enumerate IPv6 Interfaces via SSH	Disable IPv6 if you are not actually using it. Otherwise, disable any unused IPv6 interfaces.	192.168.1.62

Continued on next page

Table B.6 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
Enumerate IPv4 Interfaces via SSH	Disable any unused IPv4 interfaces.	192.168.1.62
Enumerate MAC Addresses via SSH	Disable any unused interfaces.	192.168.1.62
SSL Certificate 'commonName' Mismatch	If the machine has several names, make sure that users connect to the service through the DNS hostname that matches the common name in the certificate.	192.168.1.62
SSL Certificate with Wrong Hostname	Purchase or generate a proper certificate for this service.	192.168.1.62
SSL Certificate Cannot Be Trusted	Purchase or generate a proper certificate for this service.	192.168.1.62
KVM / QEMU Guest Detection (credentialed check)	Ensure that the host's configuration agrees with your organization's acceptable use and security policies.	192.168.1.62
Unix Operating System Unsupported Version Detection	Upgrade to a version of the Unix operating system that is currently supported.	192.168.1.73
TCP/IP Sequence Prediction Blind Reset Spoofing DoS	Contact the vendor for a patch or mitigation advice.	192.168.1.73
SSH Server CBC Mode Ciphers Enabled	Contact the vendor or consult product documentation to disable CBC mode cipher encryption, and enable CTR or GCM cipher mode encryption.	192.168.1.73
SSH Weak MAC Algorithms Enabled	Contact the vendor or consult product documentation to disable MD5 and 96-bit MAC algorithms.	192.168.1.73

Continued on next page

Table B.6 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
SSH Weak Algorithms Supported	Contact the vendor or consult product documentation to remove the weak ciphers.	192.168.1.73
OpenSSH SSHFP Record Verification Weakness	Update to version 6.7 or later or apply the vendor patch.	192.168.1.73
OpenSSH < 6.9 Multiple Vulnerabilities	Upgrade to OpenSSH 6.9 or later.	192.168.1.73
OpenSSH < 7.0 Multiple Vulnerabilities	Upgrade to OpenSSH 7.0 or later.	192.168.1.73
OpenSSH < 7.2 Untrusted X11 Forwarding Fallback Security Bypass	Upgrade to OpenSSH version 7.2 or later.	192.168.1.73
OpenSSH < 7.2p2 X11Forwarding xauth Command Injection	Upgrade to OpenSSH version 7.2p2 or later.	192.168.1.73
OpenSSH < 7.3 Multiple Vulnerabilities	Upgrade to OpenSSH version 7.3 or later.	192.168.1.73
OpenSSH < 7.4 Multiple Vulnerabilities	Upgrade to OpenSSH version 7.4 or later.	192.168.1.73
OpenSSH < 7.5	Upgrade to OpenSSH version 7.5 or later.	192.168.1.73
OpenSSH < 7.6	Upgrade to OpenSSH version 7.6 or later.	192.168.1.73
OpenSSH < 4.4 Multiple Vulnerabilities	Upgrade to OpenSSH 4.4 or later.	192.168.1.73
OpenSSH X11 Forwarding Session Hijacking	Upgrade to OpenSSH version 5.0 or later.	192.168.1.73

Continued on next page

Table B.6 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
OpenSSH < 5.2 CBC Plaintext Disclosure	Upgrade to OpenSSH 5.2 or later.	192.168.1.73
OpenSSH < 4.5 Multiple Vulnerabilities	Upgrade to OpenSSH 4.5 or later.	192.168.1.73
OpenSSH < 4.7 Trusted X11 Cookie Connection Policy Bypass	Upgrade to OpenSSH 4.7 or later.	192.168.1.73
OpenSSH < 4.9 'ForceCommand' Directive Bypass	Upgrade to OpenSSH version 4.9 or later.	192.168.1.73
OpenSSH X11UseLocalhost X11 Forwarding Port Hijacking	Upgrade to OpenSSH version 5.1 or later.	192.168.1.73
OpenSSH < 5.7 Multiple Vulnerabilities	Upgrade to OpenSSH 5.7 or later.	192.168.1.73
Portable OpenSSH ssh-keysign ssh-rand-helper Utility File Descriptor Leak Local Information Disclosure	Upgrade to Portable OpenSSH 5.8p2 or later.	192.168.1.73
Linux Kernel TCP Sequence Number Generation Security Weakness	Contact the OS vendor for a Linux kernel update / patch.	192.168.1.73
OpenSSH LoginGraceTime / MaxStartups DoS	Upgrade to OpenSSH 6.2 and review the associated server configuration settings.	192.168.1.73

Continued on next page

Table B.6 – *Continued from previous page*

Vulnerability Name	**Solution**	**Targets**
OpenSSH < 6.6 Multiple Vulnerabilities	Upgrade to OpenSSH version 6.6 or later.	192.168.1.73
CentOS 3 / 4 / 5 /6 / 7 : missing security updates	Update the affected following packages: pango, nss, gnutls, krb5 nss_db, sudo, kernel, postgresql, mysql, perl, samba/samba3x, cups, perl-Archive-Tar, avahi, pcsc-lite,,pam, php,lftp, libpng/libpng10, openldap, freetype,bind,dhcp, NetworkManager/dbus-glib, openssl(Logjam) bzip2, poppler , glibc openssl, libuser,libtiff, vsftpd, logwatch, dbus, python, apr, curl, dovecot, bash, libpng, ecryptfs-utils, nspr/nss , HTTPD, libxml2, libpng/libpng10, expat, libxslt, sos, quota, tcl, gnome-vfs2, autofs, gtk2, elinks, mesa, gdm/initscripts, xinetd, libgcrypt, libjpeg, gnupg, yum-updatesd, squid, procmail,ccid, bash(Shellshock), openssl(POODLE), rpm, nss(POODLE), openssl(FREAK),firefox, openssl(DROWN), (Badlock), openssl (httpoxy), (DirtyCOW), nss/nss-util,(GHOST)	192.168.1.73

Table B.6: CloudStack 192.168. 1.0/24 Targets Vulnerabilities and Solutions suggested by Nessus.

Although the tenant network resource in Table B.6 belongs to the attacker, however, it was important to display those vulnerabilities because one of the resources is a vRouter (192.168.1.1) implemented by the cloud system. The security is lacking because of a weakness in the cloud platform, not the tenant. Every tenant that uses the same resource will suffer from similar vulnerabilities. Moreover, one of the VM (192.168.1.73) was created from an image provided by the cloud system and that resource was security poor considering the number and level of vulnerabilities. Hence, any tenant will be at risk ifthey chose this resource. Finally, the other VM (192.168.1.62) was created from a custom image and yet it still displayed some vulnerabilities. Therefore every tenant should test every resource they create in a cloud. The following are the vulnerabilities from Table B.6 are newly discovered by Nessus beside the one displayed in the previous Tables B.3, B.3 and B.5:

1. DHCP Server Detection: The remote DHCP server may expose information about the associated network
   a) This script contacts the remote DHCP server (if any) and attempts to retrieve information about the network layout.
   b) Some DHCP servers provide sensitive information such as the NIS domain name, or network layout information such as the list of the network web servers, and so on.
   c) It does not demonstrate any vulnerability, but a local attacker may use DHCP to become intimately familiar with the associated network.
   d) The Tenant vRouter and Host cloud nodes are required to fix this issue.
2. DNS Server Detection: A DNS server is listening on the remote host
   a) The remote service is a Domain Name System (DNS) server, which provides a mapping between hostnames and IP addresses.
   b) The Tenant vRouter and Host cloud nodes are required to fix this issue.
3. DNS Server Cache Snooping Remote Information Disclosure: The remote DNS server is vulnerable to cache snooping attacks.
   a) The remote DNS server responds to queries for third-party domains that do not have the recursion bit set.
   b) This may allow a remote attacker to determine which domains have recently been resolved via this name server, and therefore which hosts have been recently visited.
   c) For instance, if an attacker was interested in whether your company utilizes the online services of a particular financial institution, they would be able to use this attack to build a statistical model regarding company usage of that financial institution. Of course, the attack can also be used to find B2B partners, web-surfing patterns, external mail servers, and more.
   d) Note: If this is an internal DNS server not accessible to outside networks, attacks would be limited to the internal network. This may include employees, consultants and potentially users on a guest network or WiFi connection if supported.
   e) The Tenant vRouter and Host cloud nodes are required to fix this issue.

4. DNS Server Spoofed Request Amplification DDoS: The remote DNS server could be used in a distributed Denial of Service attack
   a) The remote DNS server answers to any request. It is possible to query the name servers (NS) of the root zone ('.') and get an answer that is bigger than the original request.
   b) By spoofing the source IP address, a remote attacker can leverage this 'amplification' to launch a Denial of Service attack against a third-party host using the remote DNS server.
   c) The Tenant vRouter and Host cloud Nodes issue.
5. dnsmasq < 2.63test1 & 2.63test2 libvirtd TCP Network Packet Parsing Response DNS Amplification DoS: The remote DNS / DHCP service is affected by a Denial of Service vulnerability.
   a) The remote dnsmasq server is running a version prior to 2.63test1 and 2.66test2. It is, therefore, affected by a Denial of Service vulnerability in libvirtd due to improper parsing of malformed network packets. An unauthenticated, remote attacker can exploit this to cause an amplification of a large amount of DNS queries, resulting in a Denial of Service condition.
   b) Note that this vulnerability exists due to an incomplete fix for CVE-2012-3411.
   c) The Tenant vRouter and Host cloud nodes are required to fix this issue.
6. dnsmasq < 2.73rc4 setup_reply() Function Return Value Checking Information Disclosure: The remote DNS / DHCP service is affected by an information disclosure vulnerability.
   a) The remote dnsmasq server is running a version prior to 2.73rc4. It is, therefore, affected by an information disclosure vulnerability due not properly checking the return value from the setup_reply() function during TCP connections.
   b) An unauthenticated, remote attacker can exploit this to disclose sensitive information.
   c) The Tenant vRouter and Host cloud nodes are required to fix this issue.
7. dnsmasq < 2.78 Multiple Remote Vulnerabilities: The remote DNS / DHCP service is affected by multiple vulnerabilities.
   a) The version of dnsmasq installed on the remote host is prior to 2.78, and thus, is affected by the following vulnerabilities :

i. Denial of Service related to handling DNS queries exceeding 512 bytes. (CVE-2017-13704)

ii. Heap overflow related to handling DNS requests. (CVE-2017-14491)

iii. Heap overflow related to IPv6 router advertisement handling. (CVE-2017-14492)

iv. Stack overflow related to DHCPv6 request handling. (CVE-2017-14493)

v. Memory disclosure related to DHCPv6 packet handling. (CVE-2017-14494)

vi. Denial of Service related to handling DNS queries. (CVE-2017-14495)

vii. Denial of Service related to handling DNS queries. (CVE-2017-14496)

b) The Tenant vRouter and Host cloud nodes are required to fix this issue.

8. Enumerate IPv6 and IPv4 interfaces via SSH: Nessus was able to enumerate the IPv6 interfaces on the remote host.

   a) Nessus was able to enumerate the network interfaces configured with IPv6 and IPv4 addresses by connecting to the remote host via SSH using the supplied credentials.

   b) The Tenant customed VM and Host cloud nodes are required to fix this issue.

9. SSL Certificate ’commonName’ Mismatch: The ’commonName’ (CN) attribute in the SSL certificate does not match the hostname.

   a) The service running on the remote host presents an SSL certificate for which the ’commonName’ (CN) attribute does not match the hostname on which the service listens.

   b) The Tenant customed VM and Host cloud nodes are required to fix this issue.

10. SSL Certificate with Wrong Hostname: The SSL certificate for this service is for a different host.

    a) The ’commonName’ (CN) attribute of the SSL certificate presented for this service is for a different machine.

    b) The Tenant customed VM and Host cloud nodes are required to fix this issue.

11. KVM / QEMU Guest Detection (credentialed check): The remote host seems to be a KVM / QEMU virtual machine.

   a) According to its model name, the machine is running on a QEMU virtual processor.

   b) The Tenant customed VM and Host cloud Nodes issue.

12. OpenSSH < 4.4 Multiple Vulnerabilities: The remote SSH server is affected by multiple vulnerabilities.

    a) According to its banner, the version of OpenSSH installed on the remote host is affected by multiple vulnerabilities :

       i. A race condition exists that may allow an unauthenticated, remote attacker to crash the service or, on portable OpenSSH, possibly execute code on the affected host. Note that successful exploitation requires that GSSAPI authentication be enabled.

       ii. A flaw exists that may allow an attacker to determine the validity of usernames on some platforms. Note that this issue requires that GSSAPI authentication be enabled.

       iii. When SSH version 1 is used, an issue can be triggered via an SSH packet that contains duplicate blocks that could result in a loss of availability for the service.

       iv. On Fedora Core 6 (and possibly other systems), an unspecified vulnerability in the linux_audit_record_event() function allows remote attackers to inject incorrect information into audit logs.

    b) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

13. OpenSSH X11 Forwarding Session Hijacking: The remote SSH service is prone to an X11 session hijacking vulnerability.

    a) According to its banner, the version of SSH installed on the remote host is older than 5.0.

    b) Such versions may allow a local user to hijack X11 sessions because it improperly binds TCP ports on the local IPv6 interface if the corresponding ports on the IPv4 interface are in use.

    c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

14. OpenSSH < 5.2 CBC Plaintext Disclosure: The SSH service running on the remote host has an information disclosure vulnerability.

    a) The version of OpenSSH running on the remote host has an information disclosure vulnerability. A design flaw in the SSH specification could allow a Man-in-the-Middle attacker to recover up to 32 bits of plaintext from an SSH-protected connection in the standard configuration.

    b) An attacker could exploit this to gain access to sensitive information.

    c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

15. OpenSSH < 4.5 Multiple Vulnerabilities: The remote SSH service is affected by multiple vulnerabilities.

    a) If, according to its banner, the remote host runs a version of OpenSSH prior to 4.5. Versions before 4.5 are affected by the following vulnerabilities :

        i. A client-side NULL pointer dereference, caused by a protocol error from a malicious server, which could cause the client to crash. (CVE-2006-4925)

        ii. A privilege separation vulnerability exists, which could allow attackers to bypass authentication. The vulnerability is caused by a design error between privileged processes and their child processes. Note that this particular issue is only exploitable when other vulnerabilities are present. (CVE-2006-5794)

        iii. An attacker that connects to the service before it has finished creating keys could force the keys to be recreated. This could result in a Denial of Service for any processes that relies on a trust relationship with the server. Note that this particular issue only affects the Apple implementation of OpenSSH on Mac OS X. (CVE-2007-0726)

    b) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

16. . OpenSSH is earlier than 4.7 Trusted X11 Cookie Connection Policy Bypass: Remote attackers may be able to bypass authentication.

    a) According to the banner, OpenSSH earlier than 4.7 is running on the remote host. Such versions contain an authentication bypass vulnerability. In the event that OpenSSH cannot create an untrusted cookie for client X, for example due to the temporary partition being full, it will use a trusted cookie instead.

b) This allows attackers to violate intended policy and gain privileges by causing their client X to be treated as trusted.

c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

17. OpenSSH is earlier than 4.9 'ForceCommand' Directive Bypass: The remote SSH service is affected by a security bypass vulnerability

    a) According to its banner, the version of OpenSSH installed on the remote host is earlier than 4.9.

    b) It may allow a remote, authenticated user to bypass the 'sshd_config' 'ForceCommand' directive by modifying the '.ssh/rc' session file.

    c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

18. OpenSSH X11UseLocalhost X11 Forwarding Port Hijacking: The remote SSH service may be affected by an X11 forwarding port hijacking vulnerability.

    a) According to its banner, the version of SSH installed on the remote host is older than 5.1 and may allow a local user to hijack the X11 forwarding port. The application improperly sets the 'SO_REUSEADDR' socket option when the 'X11UseLocalhost' configuration option is disabled.

    b) Note that most operating systems, when attempting to bind to a port that has previously been bound with the 'SO_REUSEADDR' option, will check that either the effective user-id matches the previous bind (common BSD-derived systems) or that the bind addresses do not overlap (Linux and Solaris). This is not the case with other operating systems such as HP-UX.

    c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

19. OpenSSH < 5.7 Multiple Vulnerabilities: The remote SSH service may be affected by multiple vulnerabilities.

a) According to its banner, the version of OpenSSH running on the remote host is earlier than 5.7. Versions before 5.7 may be affected by the following vulnerabilities :

   i. A security bypass vulnerability because OpenSSH does not properly validate the public parameters in the J-PAKE protocol. This could allow an attacker to authenticate without the shared secret. Note that this issue is only exploitable when OpenSSH is built with J-PAKE support, which is currently experimental and disabled by default, and that Nessus has not checked whether J-PAKE support is indeed enabled. (CVE-2010-4478)

   ii. The auth_parse_options function in auth-options.c in sshd provides debug messages containing authorized_keys command options, which allows remote, authenticated users to obtain potentially sensitive information by reading these messages. (CVE-2012-0814)

b) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

20. Portable OpenSSH ssh-keysign ssh-rand-helper Utility File Descriptor Leak Local Information Disclosure: Local attackers may be able to access sensitive information.

    a) According to its banner, the version of OpenSSH running on the remote host is earlier than 5.8p2. Such versions may be affected by a local information disclosure vulnerability that could allow the contents of the host's private key to be accessible by locally tracing the execution of the ssh-keysign utility.

    b) Having the host's private key may allow the impersonation of the host.

    c) Note that installations are only vulnerable if ssh-rand-helper was enabled during the build process, which is not the case for *BSD, OS X, Cygwin and Linux.

    d) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

21. Linux Kernel TCP Sequence Number Generation Security Weakness: It may be possible to predict the TCP/IP Initial Sequence Numbers for the remote host.

    a) The Linux kernel is prone to a security weakness related to TCP sequence number generation. Attackers can exploit this issue to inject arbitrary packets into TCP sessions using a brute-force attack.

    b) An attacker may use this vulnerability to create a Denial of Service condition or a

Man-in-the-Middle attack.

c) Note that this plugin may fire as a result of a network device (such as a load balancer, VPN, IPS, transparent proxy, etc.) that is vulnerable and that re-writes TCP sequence numbers, rather than the host itself being vulnerable.

d) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

22. OpenSSH LoginGraceTime / MaxStartups DoS: The remote SSH service is susceptible to a remote Denial of Service attack.

    a) According to its banner, a version of OpenSSH earlier than version 6.2 is listening on this port. The default configuration of OpenSSH installs before 6.2 could allow a remote attacker to bypass the LoginGraceTime and MaxStartups thresholds by periodically making a large number of new TCP connections and thereby prevent legitimate users from gaining access to the service.

    b) Note that this plugin has not tried to exploit the issue or detect whether the remote service uses a vulnerable configuration. Instead, it has simply checked the version of OpenSSH running on the remote host.

    c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

23. OpenSSH is earlier than 6.6 Multiple Vulnerabilities: The SSH server on the remote host is affected by multiple vulnerabilities.

    a) According to its banner, the version of OpenSSH running on the remote host is prior to 6.6. It is, therefore, affected by the following vulnerabilities:

        i. A flaw exists due to a failure to initialize certain data structures when makefile.inc is modified to enable the J-PAKE protocol. An unauthenticated, remote attacker can exploit this to corrupt memory, resulting in a Denial of Service condition and potentially the execution of arbitrary code. (CVE-2014-1692)

        ii. An error exists related to the 'AcceptEnv' configuration setting in sshd_config due to improper processing of wildcard characters. An unauthenticated, remote attacker can exploit this, via a specially crafted request, to bypass intended environment restrictions. (CVE-2014-2532)

b) Note that Nessus has not tested for these issues but has instead relied only on the application's self-reported version number.

c) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

24. CentOS 3 / 4 / 5 /6 / 7 : missing security updates:

    a) Updated the packages listed in table B.5 that fix the security issues are available for Red Hat Enterprise Linux 3, 4, 5, 6 and 7.

    b) The Tenant -cloud provided image- VM and Host cloud nodes are required to fix this issue.

REFERENCES

[1] B. Loganayagi and S. Sujatha, “Enhanced Cloud Security by Combining Virtualization and Policy Monitoring Techniques,” *Procedia Engineering*, vol. 30, pp. 654–661, Jan 2012.

[2] C. Argyropoulos, B. A. FAU, J. Aznar, and K. Baumann, “Deliverable d13. 1 (dj2. 1.1) specialised applications support utilising openflow/sdn,” *GEANT, March*, 2015.

[3] R.-C. W. J.-P. Lin and T.-C. Wang, “Assessment Criteria of IaaS Solution in Cloud Computing,” *International Journal*, vol. 3, no. 1, pp. 1493–2305, 2013.

[4] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[5] C. Rong, S. T. Nguyen, and M. G. Jaatun, “Beyond lightning: A survey on security challenges in cloud computing,” *Computers & Electrical Engineering*, vol. 39, no. 1, pp. 47–54, 2013.

[6] Virtualization Special Interest Group and PCI Security Standards Council 2.0, “Information Supplement : PCI DSS Virtualization Guidelines,” no. June, 2011.

[7] F. Baroncelli, B. Martini, and P. Castoldi, “Network virtualization for cloud computing,” *annals of telecommunications-annales des télécommunications*, vol. 65, no. 11-12, pp. 713–721, 2010.

[8] D. Vaile, K. Kalinich, P. Fair, and A. Lawrence, “Data sovereignty and the cloud: A board and executive officer’s guide,” 2013.

[9] A. Kannan, “Performance evaluation of security mechanisms in Cloud Networks,” 2012.

[10] Z. Xiao and Y. Xiao, “Security and privacy in cloud computing,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.

[11] S. A. Baset, “Open source cloud technologies,” in *Proceedings of the Third ACM Symposium on Cloud Computing*, p. 28, ACM, 2012.

[12] Openstack.org, “OPENSTACK USER SURVEY A snapshot of OpenStack users ’ attitudes and deployments,” Tech. Rep. April, 2016.

[13] D. Schlosser and M. Jarschel, “Network Virtualization : Isolation Problems and Scalability Issues Scalability of virtualized Net- works,” 2010.

[14] Z. Bozakov and P. Papadimitriou, “Towards a scalable software-defined network virtualization platform,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–8, IEEE, 2014.

[15] N. Vengurlekar, B. Clouse, and R. Kammend, “Network Isolation in Private Database Clouds,” Tech. Rep. April, Oracle Corporation, 2012.

[16] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, G. M. Parulkar, *et al.*, “OpenVirteX: A Network Hypervisor.,” in *Open Networking Summit 2014 (ONS 2014)*, 2014.

[17] H. Moraes, R. V. Nunes, and D. Guedes, “DCPortalsNg : Efficient Isolation of Tenant Networks in Virtualized Datacenters,” *Proc. 13th ICN*, 2014.

[18] R. Riggio, F. De Pellegrini, and D. Siracusa, “The price of virtualization: Performance isolation in multi-tenants networks,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–7, IEEE, 2014.

[19] Y. Kanada, K. Shiraishi, and A. Nakao, “Network-resource isolation for virtualization nodes,” *IEICE transactions on communications*, vol. 96, no. 1, pp. 20–30, 2013.

[20] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, “Virtualizing the network forwarding plane,” in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, p. 8, ACM, 2010.

[21] F. Hsu, M. S. Malik, and S. Ghorbani, “Openflow as a service,” *Retrieved April*, vol. 21, 2014.

[22] S. Gutz, A. Story, C. Schlesinger, and N. Foster, “Splendid Isolation : A Slice Abstraction for Software-Defined Networks,” in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 79–84, ACM, 2012.

[23] F. Hao, T. Lakshman, S. Mukherjee, and H. Song, “Secure cloud computing with a virtualized network infrastructure.,” in *HotCloud*, 2010.

[24] F. Aderholdt, B. Caldwell, S. Hicks, S. Koch, T. Naughton, D. Pelfrey, J. Pogge, S. L. Scott, G. Shipman, and L. Sorrillo, "Multi-tenant isolation via reconfigurable networks," tech. rep., Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2014.

[25] A. Bechtsoudis and N. Sklavos, "Aiming at higher network security through extensive penetration tests," *IEEE latin america transactions*, vol. 10, no. 3, pp. 1752–1756, 2012.

[26] M. Okuhara, T. Shiozaki, and T. Suzuki, "Security architecture for cloud computing," *Fujitsu Sci. Tech. J*, vol. 46, no. 4, pp. 397–402, 2010.

[27] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[28] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation computer systems*, vol. 28, no. 3, pp. 583–592, 2012.

[29] D. Catteddu and G. Hogben, "Cloud computing: benefits, risks and recommendations for information security. european network and information security agency," *ENISA, Heraklion, Crete, Greece, Educational and Information Report*, vol. 460, p. 2004, 2009.

[30] R. Ken, D. Harris, J. Meegan, B. Pardee, Y. Le Roux, C. Dotson, E. Cohen, M. Edwards, and J. Gershater, "Security for Cloud Computing: 10 Steps to Ensure Success," *Cloud Standards Customer Council (CSCC), Tech. Rep., August*, 2012.

[31] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 17, 2013.

[32] J. Archer, D. Cullinane, N. Puhlmann, A. Boehme, P. Kurtz, and J. Reavis, "Security Guidance for Critical Areas of Focus in Cloud Computing v3. 0," *Cloud Security Alliance*, 2011.

[33] W. Streitberger and A. Ruppel, "Cloud computing security-protection goals, taxonomy, market review," *Institute for Secure Information Technology SIT, Tech. Rep*, 2010.

[34] A. Nakao, "Network virtualization as foundation for enabling new network architectures and applications," *IEICE transactions on communications*, vol. 93, no. 3, pp. 454–457, 2010.

[35] S. Natarajan and T. Wolf, "Security issues in network virtualization for the future internet," in *Computing, Networking and Communications (ICNC), 2012 international conference on*, pp. 537–543, IEEE, 2012.

[36] N. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[37] N. M. K. Chowdhury and R. Boutaba, “Network virtualization: state of the art and research challenges,” *IEEE Communications magazine*, vol. 47, no. 7, 2009.

[38] J. Carapinha, P. Feil, P. Weissmann, S. E. Thorsteinsson, Ç. Etemoğlu, Ó. Ingþórsson, S. Çiftçi, and M. Melo, “Network virtualization-opportunities and challenges for operators,” in *Future Internet Symposium*, pp. 138–147, Springer, 2010.

[39] A. Bouayad, A. Blilat, N. E. H. Mejhed, and M. El Ghazi, “Cloud computing: security challenges,” in *Information Science and Technology (CIST), 2012 Colloquium in*, pp. 26–31, IEEE, 2012.

[40] P. R. Srivastava and S. Saurav, “Networking agent for overlay l2 routing and overlay to underlay external networks l3 routing using openflow and open vswitch,” in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pp. 291–296, IEEE, 2015.

[41] M. Anisetti, C. A. Ardagna, E. Damiani, F. Gaudenzi, and R. Veca, “Toward security and performance certification of open stack,” in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pp. 564–571, IEEE, 2015.

[42] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds,” in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199–212, ACM, 2009.

[43] Mirantis, “Mirantis OpenStack: Security Best Practices V1.0.2,” Tech. Rep. 888, 2016. Also available as `https://docs.mirantis.com/mcp/q3-18/mcp-security-best-practices/index.html`.

[44] R. LaBarge and T. McGuire, “Cloud penetration testing,” *arXiv preprint arXiv:1301.1912*, 2013.

[45] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, “Estinet openflow network simulator and emulator,” *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, 2013.

[46] A. S. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy, “Cloudsec: a Security Monitoring Appliance for Virtual Machines in the IaaS Cloud Model,” in *Network and System Security (NSS), 2011 5th International Conference on*, pp. 113–120, IEEE, 2011.

[47] Y. Wei, *Monitoring, Configuration and Resource Management of Service Workflows in Virtualized Clusters and Clouds*. University of Notre Dame, 2013.

[48] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Comparison of Multiple IaaS Cloud Platform Solutions," in *Proceedings of the 7th WSEAS International Conference on Computer Engineering and Applications,(Milan-CEA 13). ISBN*, pp. 978–1, 2012.

[49] S. A. Baset, C. Tang, B.-C. Tak, and L. Wang, "Dissecting open source cloud evolution: An openstack case study.," in *HotCloud*, 2013.

[50] P. Rad, R. V. Boppana, P. Lama, G. Berman, and M. Jamshidi, "Low-latency software defined network for high performance clouds," in *System of Systems Engineering Conference (SoSE), 2015 10th*, pp. 486–491, IEEE, 2015.

[51] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of multi-tenant virtual networks in openstack-based cloud infrastructures," in *Globecom Workshops (GC Wkshps), 2014*, pp. 81–85, IEEE, 2014.

[52] Ł. Podleski, E. JACOB, J. AZNAR-BARANDA, X. JEANNIN, K. BAUMANN, and C. ARGYROPOULOS, "Multi-domain software defined network: exploring possibilities," *TNC*, 2014.

[53] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy, "Network virtualization architecture: Proposal and initial prototype," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 63–72, ACM, 2009.

[54] V. Fusenig and A. Sharma, "Security architecture for cloud networking," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pp. 45–49, IEEE, 2012.

[55] P. Murray *et al.*, "Dd. 1 cloud network architecture description," *EU FP7 Project Deliverable*, 2011.

[56] T. Lin, J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Enabling SDN applications on software-defined infrastructure," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–7, IEEE, 2014.

[57] P. Schoo, V. Fusenig, V. Souza, M. Melo, P. Murray, H. Debar, H. Medhioub, and D. Zeghlache, "Challenges for cloud networking security," in *International Conference on Mobile Networks and Management*, pp. 298–313, Springer, 2010.

[58] R. Kloti, V. Kotronis, and P. Smith, “Openflow: A security analysis,” in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pp. 1–6, IEEE, 2013.

[59] O. N. Fundation, “SDN Architecture Overview,” *ONF White Paper*, vol. 1.1, pp. 1–8, 2014.

[60] R. V. Nunes, R. L. Pontes, and D. Guedes, “Virtualized network isolation using software defined networks,” in *2013 IEEE 38th Conference on Local Computer Networks (LCN 2013)*, pp. 683–686, IEEE, 2013.

[61] F. Hu, Q. Hao, and K. Bao, “A survey on software-defined network and openflow: From concept to implementation,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[62] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, “Performance of network virtualization in cloud computing infrastructures: The openstack case,” in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pp. 132–137, IEEE, 2014.

[63] M. B. Gebreyohannes, “Network Performance Study on OpenStack Cloud Computing,” Master’s thesis, 2014.

[64] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[65] N. C. Fernandes, M. D. Moreira, I. M. Moraes, L. H. G. Ferraz, R. S. Couto, H. E. Carvalho, M. E. M. Campista, L. H. M. Costa, and O. C. M. Duarte, “Virtual networks: Isolation, performance, and trends,” *Annals of telecommunications-annales des télécommunications*, vol. 66, no. 5-6, pp. 339–355, 2011.

[66] A. Edwards, A. Fischer, and A. Lain, “Diverter: A new approach to networking within virtualized infrastructures,” in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pp. 103–110, ACM, 2009.

[67] R. Housley and S. Hollenbeck, “EtherIP: Tunneling Ethernet Frames in IP Datagrams,” tech. rep., 2002.

[68] IEEE Computer Society, “IEEE Standards for Local and metropolitan area networks: Virtual Bridged Local Area Networks,” tech. rep., 2003.

[69] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, *Resilient Overlay Networks*, vol. 35. ACM, 2001.

[70] A. C. Bavier, M. Bowman, B. N. Chun, D. E. Culler, S. Karlin, S. Muir, L. L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating systems support for planetary-scale network services.," in *NSDI*, vol. 4, pp. 19–19, 2004.

[71] S. Miura, T. Okamoto, T. Boku, M. Sato, and D. Takahashi, "Low-cost High-bandwidth Tree Network for PC Clusters Based on Tagged-VLAN Technology," in *Parallel Architectures, Algorithms and Networks, 2005. ISPAN 2005. Proceedings. 8th International Symposium on*, pp. 8–pp, IEEE, 2005.

[72] H. Wu, Y. Ding, C. Winer, and L. Yao, "Network security for virtual machine in cloud computing," in *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*, pp. 18–21, IEEE, 2010.

[73] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, vol. 1, p. 132, 2009.

[74] W. Wu, G. Wang, A. Akella, and A. Shaikh, "Virtual network diagnosis as a service," in *Proceedings of the 4th annual Symposium on Cloud Computing*, p. 9, ACM, 2013.

[75] C. R. Senna, M. A. Soares, L. F. Bittencourt, and E. R. Madeira, "An architecture for adaptation of virtual networks on clouds," in *Network Operations and Management Symposium (LANOMS), 2011 7th Latin American*, pp. 1–8, IEEE, 2011.

[76] S. Ahn, S. Lee, S. Yoo, D. Park, D. Kim, and C. Yoo, "Isolation Schemes of Virtual Network Platform for Cloud Computing.," *KSII Transactions on Internet & Information Systems*, vol. 6, no. 11, 2012.

[77] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "Cloudnaas: a cloud networking platform for enterprise applications," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, p. 8, ACM, 2011.

[78] M. Mechtri, D. Zeghlache, E. Zekri, and I. J. Marshall, "Inter and Intra Cloud Networking Gateway as a Service," in *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*, pp. 156–163, IEEE, 2013.

[79] C. De Cusatis, R. Cannista, and L. Hazard, "Managing multi-tenant services for software defined cloud data center networks," in *Adaptive Science & Technology (ICAST), 2014 IEEE 6th International Conference on*, pp. 1–5, IEEE, 2014.

[80] J. Brassil, "Physical layer network isolation in multi-tenant clouds," in *Distributed computing systems workshops (icdcsw), 2010 IEEE 30th international conference on*, pp. 77–81, IEEE, 2010.

[81] Cloud Security Alliance, "SecaaS Implementation Guidance, Category 10:Network Security," September 2012. Available at: https://cloudsecurityalliance.org/research/secaas/#_-downloads.

[82] S. J. Vaughan-Nichols, "Openflow: The next generation of the network?," *Computer*, no. 8, pp. 13–15, 2011.

[83] M. Suñé, L. Bergesio, H. Woesner, T. Rothe, A. Köpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, *et al.*, "Design and implementation of the ofelia fp7 facility: The european openflow testbed," *Computer Networks*, vol. 61, pp. 132–150, 2014.

[84] A. Blenk, A. Basta, and W. Kellerer, "HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 397–405, IEEE, 2015.

[85] *Openstack Networking Guide*. OpenStack Foundation, 2015.

[86] A. Nguyen-Ngoc, S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, and M. Jarschel, "Investigating isolation between virtual networks in case of congestion for a pronto 3290 switch," in *Networked Systems (NetSys), 2015 International Conference and Workshops on*, pp. 1–5, IEEE, 2015.

[87] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, 2005.

[88] E. Keller and J. Rexford, "The" platform as a service" model for networking.," *INM/WREN*, vol. 10, pp. 95–108, 2010.

[89] V. Rathore, J. Yoo, J. Lee, and S. Hong, "Providing network performance isolation in VDE-based cloud computing systems," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 721–725, IEEE, 2011.

[90] N. C. Fernandes and O. C. M. B. Duarte, "Xnetmon: A network monitor for securing virtual networks," in *Communications (ICC), 2011 IEEE International Conference on*, pp. 1–5, IEEE, 2011.

[91] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, “Software-defined networking: Challenges and research opportunities for future internet,” *Computer Networks*, vol. 75, pp. 453–471, 2014.

[92] D. Salomoni and M. Caberletti, “A dynamic virtual networks solution for cloud computing,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*, pp. 526–534, IEEE, 2012.

[93] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, “OpenVirteX: Make your virtual SDNs programmable,” in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 25–30, ACM, 2014.

[94] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[95] J. Reich, C. Monsanto, N. Foster, J. Rexford, D. Walker, and P. Cornell, “Composing software defined networks,” in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Citeseer, 2013.

[96] M. El-Azzab, I. L. Bedhiaf, Y. Lemieux, and O. Cherkaoui, “Slices isolator for a virtualized openflow node,” in *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pp. 121–126, IEEE, 2011.

[97] G. Kontesidou and K. Zarifis, “Openflow virtual networking: A flow-based network virtualizationarchitecture,” 2009.

[98] N. Foster, M. J. Freedman, A. Guha, R. Harrison, N. P. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, and C. Schlesinger, “Languages for software-defined networks,” tech. rep., PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE, 2013.

[99] S. Shetty, N. Luna, and K. Xiong, “Assessing Network Path Vulnerabilities for Secure Cloud Computing,” in *Communications (ICC), 2012 IEEE International Conference on*, pp. 5548–5552, IEEE, 2012.

[100] J. Carapinha and J. Jiménez, “Network virtualization: a view from the bottom,” in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 73–80, ACM, 2009.

[101] P. H. V. Guimaraes, L. H. G. Ferraz, J. V. Torres, D. M. Mattos, I. D. Alvarenga, C. S. Rodrigues, O. C. M. Duarte, *et al.*, “Experimenting Content-centric Networks in the

Future Internet Testbed Environment," in *Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 1383–1387, IEEE, 2013.

[102] X. Jin, E. Keller, and J. Rexford, "Virtual switching without a hypervisor for a more secure cloud.," in *Hot-ICE*, 2012.

[103] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication - SIGCOMM '08*, ACM Press, 2008.

[104] R. H. E. L. OpenStack, "Mellanox Reference Architecture for Red Hat Enterprise Linux OpenStack Platform 4.0," 2014.

[105] W. Shi, J. Lee, T. Suh, D. H. Woo, and X. Zhang, "Architectural support of multiple hypervisors over single platform for enhancing cloud computing security," in *Proceedings of the 9th conference on Computing Frontiers*, pp. 75–84, ACM, 2012.

[106] K. Sloan, "Security in a virtualised world," *Network Security*, vol. 2009, no. 8, pp. 15–18, 2009.

[107] P. Chau and Y. Wang, "Security-awareness in Network Virtualization: A Classified Overview," in *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pp. 545–550, IEEE, 2014.

[108] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 408–415, IEEE, 2010.

[109] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 127–132, ACM, 2012.

[110] H. Zhou, C. Wu, M. Jiang, B. Zhou, W. Gao, T. Pan, and M. Huang, "Evolving defense mechanism for future network security," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 45–51, 2015.

[111] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, ACM Press, 2013.

[112] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 198–211, 2013.

[113] V. T. Costa and L. Costa, "Vulnerability study of flowvisor-based virtualized network environments," in *2nd Workshop on Network Virtualization and Intelligence for the Future Internet*, 2013.

[114] C. Modi and D. Patel, "A feasible approach to intrusion detection in virtual network layer of cloud computing," *Sādhanā*, vol. 43, no. 7, p. 114, 2018.

[115] M. A. Lopez, A. G. P. Lobato, O. C. M. Duarte, and G. Pujolle, "An Evaluation of a Virtual Network Function for Real-time Threat Detection Using Stream Processing," in *Mobile and Secure Services (MobiSecServ), 2018 Fourth International Conference on*, pp. 1–5, IEEE, 2018.

[116] M. Pourzandi, M. F. Ahmed, M. Cheriet, and C. Talhi, "Multi-tenant Isolation in a Cloud Environment Using Software Defined Networking," Mar 2018. US Patent 9,912,582.

[117] L. W. Paczkowski, J. P. Sisul, and M. Balmakhtar, "Virtual Network Function (VNF) Hardware Trust in a Network Function Virtualization (NFV) Software Defined Network (SDN)," Jan 2018. US Patent App. 15/216,677.

[118] S. K. Adolph, M. A. Gazier, I. H. Duncan, J. A. Frodsham, and D. Fluet, "Method and Apparatus for Provisioning Virtual Network Functions from a Network Service Provider," Apr 2018. US Patent 9,936,047.

[119] A. V. Nimkar and S. K. Ghosh, "Towards Full Network Virtualization in Horizontal IaaS Federation: Security Issues," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, p. 19, 2013.

[120] D. Shackleford, *Virtualization security: protecting virtualized environments*. John Wiley & Sons, 2012.

[121] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," *IEEE Security & Privacy*, vol. 9, no. 2, pp. 50–57, 2011.

[122] A. Saeed, P. Garraghan, B. Craggs, D. van der Linden, A. Rashid, and S. A. Hussain, "A cross-virtual machine network channel attack via mirroring and tap impersonation," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 606–613, IEEE, 2018.

[123] R. W. Beggs, *Mastering Kali Linux for advanced penetration testing*. Packt Publishing Ltd, 2014.

[124] K. Thimmaraju, G. Rétvári, and S. Schmid, “Virtual Network Isolation: Are We There Yet?,” 2018.

[125] L. Allen, T. Heriyanto, and S. Ali, *Kali Linux - Assuring security by penetration testing*. Packt Publishing Ltd, 2014.

[126] J. Broad and A. Bindner, “Introduction to the penetration test lifecycle,” in *Hacking with Kali: practical penetration testing techniques*, pp. 85–88, Elsevier, 2014.

[127] N. C. L. Hess, D. J. Carlson, J. D. Inder, E. Jesulola, J. R. Mcfarlane, and N. A. Smart, *The Hacker Playbook*, vol. 65. 2016.

[128] P. Engebretson, “What is penetration testing,” in *The Basics of Hacking and Penetration Testing*, pp. 1–14, Elsevier, 2011.

[129] K. Barker and S. Morris, *CCNA Security 640-554 Official Cert Guide*. Pearson Education, 2012.

[130] D. R. Tobergte and S. Curtis, *The Hacker Playbook*, vol. 53. 2013.

[131] J. Broad and A. Bindner, “Software, patches, and upgrades,” in *Hacking with Kali: practical penetration testing techniques*, Newnes, 2014.

[132] M. Rouse, “What is Nessus?.” `http://searchnetworking.techtarget.com/definition/Nessus`, 2006. [Online; Accessed 2017-06-30].

[133] Kamalb, “NESSUS Vulnerability Scanner - Basics.” `http://www.securitylearn.net/2013/02/27/nessus-vulnerability-scanner-basics/`, 2013. [Online; Accessed 2017-06-30].

[134] T. Klosowski, “How to Use Nessus To Scan a Network for Vulnerabilities.” `https://lifehacker.com/how-to-use-nessus-to-scan-a-network-for-vulnerabilities-1788261156`, 2016. [Online; Accessed 2017-06-30].

[135] TechTarget Network, Search Security, “Nessus 3 Tutorial: How to use Nessus to identify network vulnerabilities.” `http://searchsecurity.techtarget.com/tutorial/Nessus-3-Tutorial`, 2008. [Online; Accessed 2017-06-30].

[136] J. Hutchens, *Kali Linux network scanning cookbook*. Packt Publishing Ltd, 2014.

[137] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[138] M. J. R. P., “There’s Real Magic behind OpenStack Neutron - Tricky Deadline.” `https://pinrojas.com/2014/07/29/theres-real-magic-behind-openstack-neutron/`, 2014. [Online; accessed 2017-04-11].

[139] OpenStack, “OpenStack Docs: Network Troubleshooting.” `https://docs.openstack.org/ops-guide/ops-network-troubleshooting.html`, 2017. [Online; accessed 2017-04-11].

[140] M. Kashin, “Network Engineering Analysis of OpenStack SDN - Network-oriented programming.” `http://networkop.co.uk/blog/2016/04/22/neutron-native/`, 2016. [Online; accessed 2017-04-12].

[141] OpenStack, “OpenStack Docs: OpenStack Installation Guide for Ubuntu.” `https://docs.openstack.org/mitaka/install-guide-ubuntu/index.html`. [Online; accessed 2017-03-04].

[142] E. K. Joseph and M. Fischer, *Common Openstack deployments: real world examples for systems adminstrators and engineers*. Prentice Hall, 2017.

[143] “Openstack Neutron using VXLAN - Open Cloud Blog.” `http://www.opencloudblog.com/?p=300`, 2014. [Online; accessed 2017-04-12].

[144] R. Kofman, “Diving into OpenStack Network Architecture - Part 2 - Basic Use Cases (Ronen Kofman’s Blog).” `https://blogs.oracle.com/ronen/entry/diving_into_openstack_network_architecture1`, 2014. [Online; accessed 2017-04-11].

[145] R. Kofman, “Diving into OpenStack Network Architecture - Part 1 (Ronen Kofman’s Blog).” `https://blogs.oracle.com/ronen/entry/diving_into_openstack_network_architecture`, 2014. [Online; accessed 2017-04-11].

[146] RDO, “Networking in too much detail - RDO.” `https://www.rdoproject.org/networking/networking-in-too-much-detail/`. [Online; accessed 2017-04-12].

[147] OpenStack Foundation, “OpenStack Installation Guide for Ubuntu 14.04 - Kilo,” tech. rep., Mar 2016.

[148] N. Sabharwal and R. Shankar, *Apache Cloudstack cloud computing*. Packt Publishing Ltd, 2013.

[149] Apache Software Foundation, “CloudStack Installation Documentation, Release 4.9.0,” tech. rep., Feb 2017.

[150] OpenStack Basement, “CloudStack Install).” https://sites.google.com/site/openstackinthebasement3/cloud-stack-install?pli=1. [Online; Accessed 2017-07-12].

[151] M. Koster, “CloudStack 4.4 Single Server on Ubuntu 14.04.1 with KVM).” http://www.greenhills.co.uk/2015/02/23/cloudstack-4.4-single-server-on-ubuntu-14.04.1-with-kvm.html, Feb 2015. [Online; Accessed 2017-07-12].

[152] T. Team, M. Aggarwal, and H. S. Tetra, “Cloud Computing with Apache CloudStack: Run your own cloud.” https://www.udemy.com/apache-cloudstack-install-build-and-run-iaas-cloud/, Jul 2017. [Online; Accessed 2017-12].

[153] C. Bunch, “KVM and OVS on Ubuntu 16.04.” https://blog.codybunch.com/2016/10/14/KVM-and-OVS-on-Ubuntu-1604/, Oct 2016. [Online; Accessed 2017-12-18].

[154] D. Sonstebo, “Networking KVM for CloudStack.” https://www.shapeblue.com/networking-kvm-for-CloudStack/, Oct 2016. [Online; Accessed 2017-12-18].

[155] A. Heyward, “Apache CloudStack 4.4 with Advanced Networking Setup Pt. 2 – Open vSwitch.” http://www.thehyperadvisor.com/CloudStack-2/CloudStack-with-advanced-networking-setup-pt-2-installing-open-vswitch-on-cento Sep 2014. [Online; Accessed 2017-12-18].

[156] Apache Software Foundation, “Administration Guide 4.8: Managing Accounts, Users and Domains.” http://docs.cloudstack.apache.org/projects/cloudstack-administration/en/4.8/accounts.html, 2016. [Online; Accessed 2017-12-18].

[157] V. Kimlaychuk and E. Weber, “KVM and advanced network. Routing and IP assignment details.” http://users.cloudstack.apache.narkive.com/bI8vLvn2/kvm-and-advanced-network-routing-and-ip-assignment-details, 2014. [Online; Accessed 2018-02-07].

[158] Apache CloudStack, “ Configuring a Virtual Private Cloud - Edition 1.” https://svn.apache.org/repos/asf/cloudstack/docsite/html/docs/en-US/Apache_CloudStack/4.1.1/html/Admin_Guide/configure-vpc.html. [Online; Accessed 2018-05-04].

[159] Apache Software Foundation, " Working with System Virtual Machines." http://docs.cloudstack.apache.org/projects/cloudstack-administration/en/4.9/systemvm.html, 2016. [Online; Accessed 2018-05-04].

[160] G. Serra, T. Lordotter, G. Nunes, and Candlerb, "nested virtualization (virtualbox>kvm)." https://groups.google.com/forum/#\protect\kern-.1667em\relaxtopic/ganeti/pdBD8p18irk, Jul 2014. [Online; Accessed 2017-07-12].

[161] rhtyd, "agent.properties." https://github.com/apache/cloudstack/blob/master/agent/conf/agent.properties, 2015. [Online; Accessed 2017-07-12].

[162] P. Santhanam and N. Mehta, " SSVM, templates, Secondary storage troubleshooting." https://cwiki.apache.org/confluence/display/CLOUDSTACK/SSVM%2C+templates%2C+Secondary+storage+troubleshooting, 2014. [Online; Accessed 2017-07-26].

[163] Apache Software Foundation, "CloudStack Documentation, Release 4.11.0," tech. rep., Apr 2018.

[164] Apache CloudStack, "The OVS Plugin." http://docs.cloudstack.apache.org/en/latest/networking/ovs-plugin.html, 2017. [Online; Accessed 2018-02-07].

[165] Ubuntu, "KVM/Managing." https://help.ubuntu.com/community/KVM/Managing, 2017. [Online; Accessed 2018-02-07].

[166] OpenStack, "Configuration — DevStack 0.0.1.dev6136 documentation." http://docs.openstack.org/developer/devstack/configuration.html. [Online; Accessed 2015-06-05].

[167] "Openstack Juno Install using Devstack." https://sreeninet.wordpress.com/2015/02/21/openstack-juno-install-using-devstack/, 2015. [Online; Accessed 2015-06-05].

[168] Apache Software Foundation, "Working with Virtual Machines." http://docs.cloudstack.apache.org/projects/cloudstack-administration/en/latest/virtual_machines.html, 2016. [Online; Accessed 2018-5-14].

[169] Cloudify, "Configuring CloudStack Cloud." https://cloudify.co/guide/2.7/setup/configuring_cloudstack.html. [Online; Accessed 2018-5-14].

[170] Apache CloudStack, "Apache CloudStack 4.1.0 CloudStack Administrator's Guide." https://svn.apache.org/repos/asf/cloudstack/docsite/html/docs/en-US/

Apache_CloudStack/4.1.0/html/Admin_Guide/configure-vpc.html, 2012. [Online; Accessed 2018-5-14].

[171] S. Sadhu, "ASF CS 4.0 Test Execution:Static NAT and Source NAT." https://cwiki.apache.org/confluence/display/CLOUDSTACK/Static+NAT+and+Source+NAT, 2012. [Online; Accessed 2018-5-14].

[172] Y. Liao, D. Yin, and L. Gao, "Network virtualization substrate with parallelized data plane," *Computer Communications*, vol. 34, no. 13, pp. 1549–1558, 2011.

[173] A. Squicciarini, S. Sundareswaran, and D. Lin, "Preventing information leakage from indexing in the cloud," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 188–195, IEEE, 2010.

[174] H. Raj, D. Robinson, T. B. Tariq, P. England, S. Saroiu, and A. Wolman, "Credo: Trusted computing for guest VMs with a commodity hypervisor," *Technical Report MSR-TR-2011-130, Microsoft Research*, 2011.

[175] Y. Mundada, A. Ramachandran, and N. Feamster, "Silverline: Data and network isolation for cloud services.," in *HotCloud*, 2011.

[176] A. Nasseri, H. M. Khamis, and I. M. M. Duncan, "Investigation of Virtual Network Isolation Security in Cloud Computing: Data Leakage Issues," in *International Conference of Big Data in Cybersecurity*, 2016.