Borja Tarrasó Hueso

Improving Security of Ericsson Cloud System

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

November 7, 2016

Helsinki Metropolia University of Applied Sciences

Preface

I would like to express my gratitude to my supervisor Ville Jääskeläinen for the useful comments, remarks, and engagement through the learning process of this master's thesis. Furthermore, I would like to thank my mentor Lauri Mikkola for introducing me to the topic as well for her support along the way.

I acknowledge my close friends for all the special moments I enjoy with you and for helping me to be who I am: Miguel Ángel Martín, Jon Diéguez, Italo Madalozo, Joan Yanini, Manuel Negre, Marcos Núñez, Germán Muñoz, Omar Benbouazza, Denis Karpov, Antti Halttunen, Antonio Salueña, Aleksi Aalto, Graeme Mallard, Adrian Yanes, David Fernandez, José Uceda, Ricardo Uceda, Pedro Ibañez, Marco Antonio Rubio, Xavier Gost, Martin J. Sánchez, Mikel Etxeberria, Antonio Puchol, Xavier Robles, Damian Rivera, David Vázquez, Guillem Martínez, Bruno Hernández, and Samuel Navarro.

In addition, I would like to thank my loved ones - my parents, José Manuel Tarrasó and Enriqueta Hueso who helped me in all things from the beginning of my life; my brothers and sister Raul Belinchón, Sergio Belinchón, and Silvana Tarrasó.

Finally, I would like to thank my wife, Olga Andreichik, who has supported me throughout the entire process, both by keeping me happy and helping me put the pieces together. I will be forever grateful for your love.

'Uno llega a ser grande por lo que lee, y no por lo que escribe.
Porque escribe lo que puede, y lee lo que quiere.'
Jose Luis Borges.

Abstract

Author(s)	Borja Tarrasó Hueso
Title	Improving Security of Ericsson Cloud System (ECS)
Number of Pages	128 (85 + 43 of Appendices)
Date	November 7, 2016
Degree	Master of Engineering, Metropolia University
Degree Programme	Information Technology
Specialization	Software Engineering
Instructor	Ville Jääskeläinen

Ericsson Cloud System (ECS) is an effort by Ericsson to provide a cloud solution product.

The solution provides distributed cloud capabilities, such as computing and storage in the network and more efficient utilization of network resources. In addition, it includes capabilities to control performance and decoupling of software from hardware. This enables automatic orchestration of predefined services. The Ericsson Cloud Execution Environment (CEE) is basically a Data Center (DC) within the ECS. The ECS allows hardware virtualization for efficient deployment of multiple applications sharing the same infrastructure.

From the services ECS provides, the most relevant ones are Open Platform Network Functions Virtualization (OPNFV), cloud storage as Platform as a Service (PaaS), and new-generation hyperscale data centre hardware, using optical interconnect and a new equipment manager for multi-vendor environments.

In order to provide this service effectively, the solution is trustable. Achieving that goal by a secure deployable solution. New challenges from the security point of view are covered, such as specific vector attacks for a cloud as well as conventional attacks.

Based on a general architectural solution and implementation, a set of security requirements are essential. To cover those requirements, a set of tests were needed that required several different specialized testing tools and different libraries to support them, as well as choosing a correct testing framework.

This document focusing on the two main and fundamental problems that ECS will expound once the cloud is deployed: hardening by removing the hardcoded credentials, and a scalable and highly available method to authenticate users in the cloud. Ericsson provided an elegant solution avoiding security threats using a secure and optimal method for authenticating and authorizing users in a distributed and virtual environment.

The final goal of this thesis was improving the security of the ECS by hardening and enabling Authentication, Authorization, and Accounting (AAA). The solution has been verified by a full set of tests which were automated in a Continuous Integration (CI) entity. Functional and non-functional tests for security features where implemented within the project.

Keywords	Cloud computing, security, configuration manager

List of Figures

List of Tables

Acronyms

3GPP Third-Generation Partnership Project

AAA Authentication, Authorization, and Accounting

ACK Acknowledgement flag

ACL Access Control List

AD Active Directory

ADSI Active Directory Service Interfaces

ANSI American National Standards Institute

API Application Programming Interface

AppArmor Application Armor

ARP Address Resolution Protocol

ASN.1 Abstract Syntax Notation One

ATDD Acceptance Test-Driven Development

BSD Berkeley Software Distribution

CCM Cloud Controls Matrix

CEE Cloud Execution Environment

CI Continuous Integration

CIDR Classless Inter-Domain Routing

CISSP Certified Information Systems Security Professional

CLI Command Line Interface

CM Configuration Manager

CPU Central Processing Unit

CRAM Challenge-Response Authentication

CRM Customer Relationship Management

CSA Cloud Security Assurance

CSP Cloud Security Provider

DBMS Database Management System

DC Data Center

DHCP Dynamic Host Control Protocol

DNS Domain Name Server

DDoS Distributed Denial of Service

DoD Department of Defense

DoS Denial of Service

DSL Domain-Specific Language

EC Electronic Communications

ECS Ericsson Cloud System

ERP Enterprise Resource Planning

ETA Estimated Time of Arrival

E2E End-to-End

FedRAMP Federal Risk and Authorization Management Program

FIN Finalization flag

FIPS Federal Information Processing Standards

FTP File Transfer Protocol

GDFL GNU Free Documentation License

GUI Graphical User Interface

GNU GNU's Not Unix

HA High Availability

HAVEGE HArdware Volatile Entropy Gathering and Expansion

HIPAA Health Insurance Portability and Accountability Act

HSM Hardware Security Module

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol over Secure Sockets Layer

IaaS Infrastructure as a Service

ICMP Internet Control Message Protocol

IdAM Identity and Access Management

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IGMP Internet Group Management Protocol

IDS Intrusion Detection System

IMAP Internet Message Access Protocol

IMPI Intelligent Platform Management Interface

IP Internet Protocol

IPID Internet Protocol ID

IPS Intrusion Prevention System

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

ISAKMP Internet Security Association and Key Management Protocol

ISC International Information Systems Security Certification Consortium

ISO International Organization for Standardization

ISO/EIC International Organization for Standardization/International Electrotechnical Commission

ISP Internet Service Provider

IT Information Technology

I&V Integration and Verification

IXOS IXIA Operating System

KPI Key Performance Indicators

L2 Layer 2

L3 Layer 3

LAN Local Area Network

LCE Log Correlation Engine

LDAP Lightweight Directory Access Protocol

LDAPS LDAP over SSL

LLC Logical Link Control

MAC Media Access Control

MDM Mobile Device Management

MD5 Message-Digest 5

MySQL My Structured Query Language

NAC Network Access Control

NAT Network Address Translation

NetBIOS Network Basic Input/Output System

NIST The National Institute of Standards and Technology

Nmap The Network Mapper

NSA National Security Agency

NSE Nmap Scripting Engine

NoSQL No Structured Query Language

NTP Network Time Protocol

NULL No flags

OAuth Open Standard to Authorization

OID OpenID

OpenID Open Identifier

OpenSSL Open-source Secure Sockets Layer

OPNFV Open Platform Network Functions Virtualization

OS Operating System

OVF Open Virtualization Format

OVFT Open Virtualization Format Tool

OWASP Open Web Application Security Project

PAM Pluggable Authentication Modules

PaaS Platform as a Service

PCI DSS Payment Card Industry Data Security Standard

PDF Portable Document Format

PDP Protection Detection Prevention

PDU Product Development Unit

PHI Protected Health Information

PII Personally Identifiable Information

PO Product Owner

POP Post Office Protocol

PPP Point-to-Point Protocol

PS Post Script

PSH Push flag

PVS Passive Vulnerability Scanner

RADIUS Remote Authentication Dial In User Service

RFC Request For Comments

RIP Routing Information Protocol

RPO Recovery Point Objective

RPS Requests Per Second

RST Reset flag

RTO Recovery Time Objective

SELinux Security Enhanced Linux

SQL Structured Query Language

SSAE Statement on Standards for Attestation Engagements

SaaS Software as a Service

SCM Secure Configuration Management

SDLC Systems Development Life Cycle

SFTP Simple File Transfer Protocol

SIEM Security Information and Event Management

SLA Service Level Agreements

SNAP SubNetwork Access Protocol

SNMP Simple Network Management Protocol

SOCKS Socket Secure

SSH Secure Shell

SSL Secure Sockets Layer

SSO Single Sign-On

STP Spanning Tree Protocol

SunRPC Sun's Remote Procedure Call

SYN Synchronization flag

TAB Tabulator

TCP Transmission Control Protocol

TLS Transport Layer Security

TPM Trusted Platform Module

TPS Transactions Per Second

TTL Time To Live

UDP User Datagram Protocol

URG Urgent flag

US United States

VIC Cloud controller

VLAN Virtual Local Area Network

VIM Virtual Infrastructure Managing

VM Virtual Machine

VPN Virtual Private Network

WoW Ways-of-Work

Xmas Christmas flags

XML Extensible Markup Language

XML-RPC XML-Remote Procedure Call

YAML YAML Ain't Markup Language

Table of contents

1. Introduction

The study focuses mainly on the cloud world and the different paradigms, including Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Software as a Service (SaaS). Services deployed within the system have also been studied; including OpenStack services, message services such as RabbitMQ, database services such as MySQL, and their security implications.

Centralized authentication with security options such as Identity and Access Management (IdAM) solutions and basic hardening are critical and fundamental topics that need to be covered, and they are be part of the solution.

Other studies concerning test frameworks and capabilities are also investigated.

1.1 Problem

After deploying the Ericsson Cloud System (ECS), Ericsson must be sure that no critical security issues occur, as this would lead to losses for the company in terms of incidents, money, and reputation.

Most of the critical data for several customers will be stored in the cloud. This includes a few million end customers. As the cloud is a complex system with several services, there are several vector attacks to break the system.

The main problem that needs to be solved from Ericsson's perspective is how to deploy in a secure way a complex service that contains a massive amount of critical data.

1.2 Solution

All the installed services should be properly configured, secured, and tested.

For the configuration part, to solve these issues, an appropriate configuration manager was selected. This covers implementation of the manifests and configuration rules, dependencies, and triggers.

Securing these services and the base Operating System (OS) includes hardening by removing not needed configurations, insecure default parameters, default accounts, hardcoded credentials, open ports, and services.

Testing forms the last part of the solution. Several health checks, port and vulner-

ability scannings, testing the implementation, and automation to avoid new regression issues have been performed.

1.3 Main Studies

Cloud computing is described in the first chapters to explain the nature of this new technology and group of services which this technology requires. The origin, need and evolution of cloud has been also described.

Next chapters explain security in general terms, its importance, and how it is applied to cloud services for the next topics of discussion. After that, test strategies, design and architecture of the system, and basic security requirements are the subsequent topics.

Finally, an explanation of hardening, the centralized login solution, and the results are presented. The implementation details are attached in the Appendix to show that this is not just a theoretical problem, in practice was solved. Implementation details are also presented while excluding the company's confidential information.

Two main studies were performed. These involved the Identity and Access Management (IdAM) solution and the Configuration Manager (CM) tool.

The IdAM provides the triple Authentication, Authorization, and Accounting (AAA), which gives security and a single entry point for users. Administrators from the system, operators, and end users use the service as such. Some policies are discussed and added as part of the solution. These include expiration accounts and password complexity.

The CM provides a smooth way of deploying the system. It also maintains the system—that is, it installs, configures, and updates or upgrades the system when security issues affect the system. It also enables periodical updates and supports upgrades when new features are introduced in later versions.

Security for these services are managed and handled by the CM, which can also be referred to as the Secure Configuration Management (SCM) when it includes, refers to, or focuses on the security part.

2. Cloud Computing

The cloud is considered the natural evolution of computers. It marks a shift from the administrators handling centralized servers to the distributed client-server world. Administrators in a centralized world were often the bottleneck, as nothing could be done without them. This gave rise to the new distributed client-server world where multiple administrators can handle the new systems. It brings flexibility and gains agility, but a drawback is that it decreases the effectiveness of security because of the deployment of more non-standard applications and services. Applications are more insecure, with more opportunities for malicious people to attack systems (B01, Kavis, 2014). Figure 1 illustratres the expectations of the new technologies within a given timeframe.

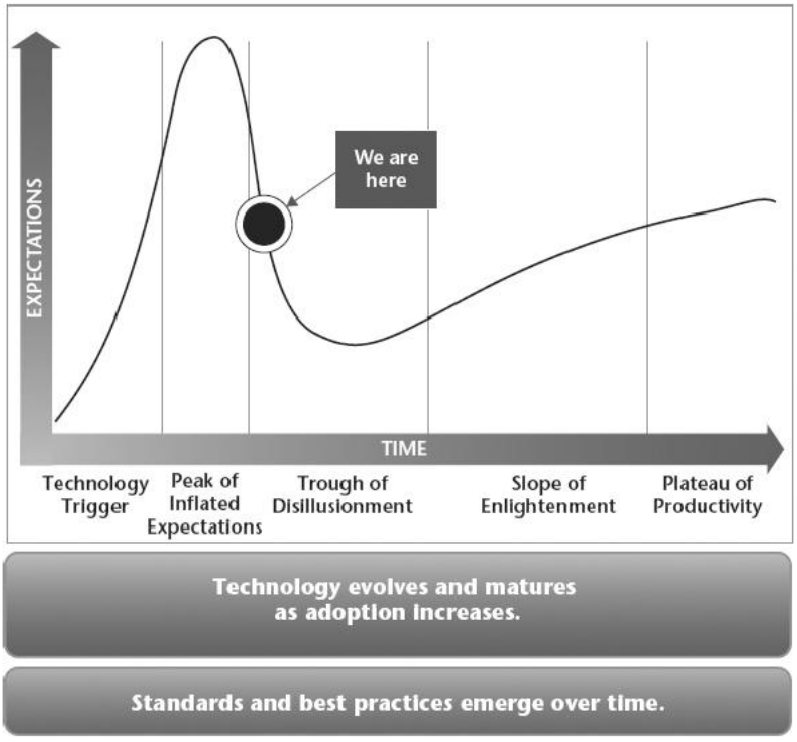


Figure 1: Expectations of new technologies during time (Architecting the cloud. p. 32).

As a consequence of the technology expectations, i ist increased the opportunity for a new business to consume resources. With demand rising, standards and security have become key topics for mass adoption, see figure 1. Cloud computing—such as other technologies that have emerged in the past—are where companies are moving. The difficult question is how this move should be carried out. Currently, cloud computing is widely accepted by small and medium-sized businesses but its adoption by large

corporations will take some time. That is because of the complexities of legacy architectures, existing infrastructures, and organizational challenges (B01, Kavis, 2014). The graph in Figure 2 illustrates the security maturity.

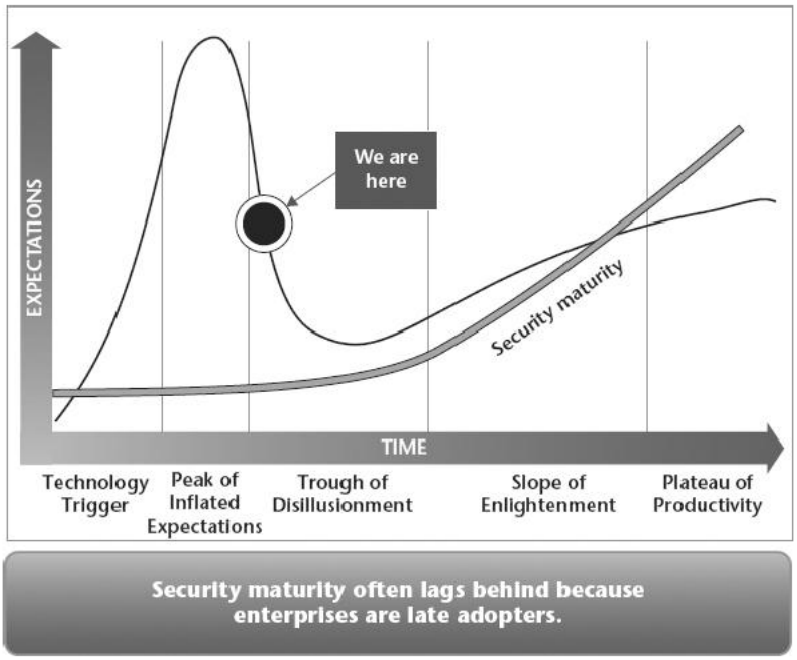


Figure 2: Security maturity evolution when adopting new technologies (Architecting the cloud, p. 34).

Cloud computing stills in a premature phase and, only in 2013 did it come to be accepted widely. The success or failure of these companies will depend on choosing the right cloud solutions for their business, which is a crucial factor governing whether these corporations make the right investments (B01, Kavis, 2014). In telecommunication companies like Ericsson, this adoption takes longer and it is still in implementation phase.

2.1 Service Models

In a traditional data center, Information Technology (IT) has the responsibility of building and managing everything. Cloud services can be deployed with different service models based on which layer the service providers and service consumers administrate the system (B02, Bento A. and Aggarwal A. 2012). Figure 3 visualizes different existing services models in the cloud.

The service models are typically divided into three categories: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) depending of stack components and the line where ends the customer and vendor responsibility from the cloud stack and stack components point of view.

2.1.1 Infrastructure as a Service (IaaS)

In IaaS, the service delivered is the computer infrastructure, which is often a platform virtualization environment together with control of storage and networking services ac-

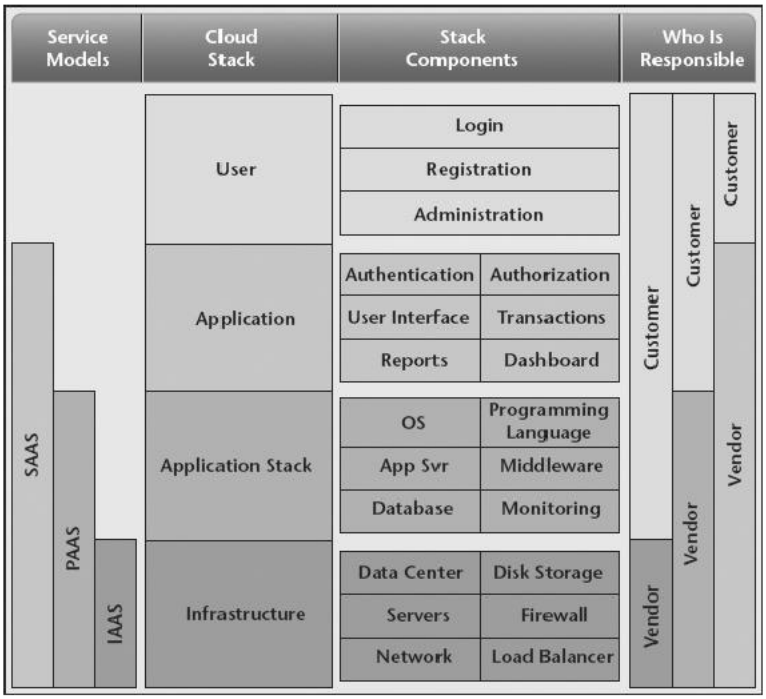


Figure 3: Different service models in cloud computing (Architecting the cloud, p. 45).

cessed using an Application Programming Interface (API). IaaS provides virtual data center capabilities. Consumers would focus on building and managing applications.

Some examples of IaaS are related to managing and maintaining a physical data center or physical infrastructure such as servers, disk storage, networking, and so on (A02, Archer J. Cullinane D. Puhlmann N. Boehme A. Kurtz P. Reavis J., 2011).

2.1.2 Platform as a Service (PaaS)

PaaS abstracts standard application functions, providing it as a service. Consumers are able to deploy cloud infrastructure, acquiring applications using programming languages, libraries, services, and tools supported by the service provider. Consumers will not control the cloud infrastructure such as networks, servers, operating systems, or storage.

Some examples of PaaS solutions are databases, monitoring, logging, caching, security, email, analytics, and payments (A02, Archer J. Cullinane D. Puhlmann N. Boehme A. Kurtz P. Reavis J., 2011).

2.1.3 Software as a Service (SaaS)

In SaaS, consumers only need to configure application-specific parameters and handling users.

Some examples of SaaS include Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), accounting, payroll, and some other business software (A02, Archer J. Cullinane D. Puhlmann N. Boehme A. Kurtz P. Reavis J., 2011).

Different service models provides different services for different needs for the end customers. Depending of which service model is used, it has different implications from the security point of view and impacts not only the development, but Service Level Agreements (SLA) and has a direct impact from the business point of view.

2.2 Deployment Models

Apart from service models introduced in the previous section, cloud has also different deployment models. The service models represent the scope and visibility of the cloud. This scope can be public, private or hybrid.

2.2.1 Public Cloud

The public cloud is a multi-tenant environment where the user pays for resources, sharing them alongside other customers. The physical location of these resources is abstracted for the end users, with the place of the physical hardware being hidden. Public clouds have several advantages:

- Utility pricing: End users are charged for used resources. They use hardware resources only when it is needed.
- Elasticity: End users have endless resources and are able to configure its software solutions to dynamically allocate the amount of resources needed, being able to react in real time for the spikes in traffic.
- Core competency: End users outsource their data center and the management of infrastructure to other companies with longer experience, spending less time on those areas and focusing on their own core competency.

There are several benefits of a public cloud. However, there are also a few drawbacks:

- Control: End users must rely on the public cloud vendor to meet their Service Level Agreements (SLA) for performance and uptime.
- Regulatory issues: Regulations regarding security of payments by card or data privacy can be challenging to deploy in the public cloud.
- Limited configurations: Public clouds have a standard set of infrastructure configurations to provide for general needs. The public cloud is not an option in case some specific hardware requirements are needed (B01, Kavis, 2014).

2.2.2 Private Cloud

Private clouds are provisioned for exclusive use of single organizations that have multiple consumers such as business units.

- The private cloud addresses the disadvantages of public clouds; control, regulatory issues, and configurations.

- Private cloud end users deploy on a single tenancy environment, which is not shared or commingled with other customers.
- Private clouds reduce regulatory risks with regard to data ownership, privacy and security given the single-tenancy nature of the model.

Referring to the disadvantages, private clouds have less elasticity, resource pooling, and higher cost. In private, cloud resources are limited as in the infrastructure, which has a limited amount of resources (B01, Kavis, 2014).

2.2.3 Hybrid Cloud

Many organizations try to take advantage of both public and private clouds. This new deployment model is called a hybrid cloud. Here, two or more distinct cloud infrastructures are bound together, enabling data and application portability.

It is good practice to use public clouds for rapid elasticity and resource pooling, and private clouds in case there are risk areas for data and privacy (B01, Kavis, 2014).

Different approaches from the security point of view requires

By using a public, private or hybrid cloud, in IaaS, PaaS or SaaS requires a proper analysis from the security perspective. Next chapter explains the different security domains and how this security is enforced in the cloud environment.

3. Security

When talking about IT security, it is necessary to distinguish between security domains in traditional models and in cloud-based environments. The following chapter introduces the different security domains categorized by the most well known IT organizations about standardization.

3.1 Security Domains

In IT, there are several different security domains depending on experts and organizations. The most accepted list is the Certified Information Systems Security Professional (CISSP) governed by the International Information Systems Security Certification Consortium (ISC)[2]. The CISSP is accredited by American National Standards Institute (ANSI) International Organization for Standardization/International Electrotechnical Commission (ISO/EIC) Standard 12024:2003. It is formally approved by the United States (US) Department of Defense (DoD) and adopted as a baseline in the US National Security Agency (NSA).

CISSP splits security into 10 domains:

- Access Control: This is defined as a collection of mechanisms that work together to create security architecture. It protects the assets of the information system.
- Telecommunications and Network Security: This includes network structures, transmission methods, transport formats, and security measures that are used to provide availability, integrity, and confidentiality.
- Information Security Governance and Risk Management: This involves identification of an organization's information assets, and the development, documentation, and implementation of policies, standards, procedures, and guidelines.
- Software Development Security: This covers the controls included within systems and applications software, and the steps used in their development.
- Cryptography: This includes principles, means, and methods of disguising information to ensure its integrity, confidentiality, and authenticity.
- Security Architecture and Design: This covers the concepts, principles, structures, and standards used to design, implement, monitor, and secure operating systems, equipment, networks, applications, and controls used to enforce different levels of confidentiality, integrity, and availability.
- Operations Security: This identifies controls over hardware, media, and the operators with access privileges of these resources.
- Business Continuity and Disaster Recovery Planning: This deals with safeguarding the business in the face of major disruptions to normal business operations.

- Legal, Regulations, Investigations, and Compliance: This includes computer crime laws and regulations, the measures and techniques that can be used to determine if a crime has been committed, and methods to gather evidence.
- Physical (Environmental) Security: This comprises threats, vulnerabilities, and countermeasures that can be utilized to physically protect resources and sensitive information. (B03, Gordon A. 2015)

All security domains are taken into account when providing cloud services.

3.2 Security Applied to Cloud

Several areas need to be checked from the security point of view in addition to the concerns from the conventional security domains. New vector attacks need to be taken into account for cloud computing.

For cloud-based solutions, specific security domains have been designed by the Cloud Security Assurance (CSA), which is currently the most widely accepted organization in the cloud world.

The Cloud Controls Matrix (CCM) is designed to provide fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider.

Some new vector attacks in the cloud are critical, especially in multi-tenancy. Some examples are given below:

- Guest Virtual Machine (VM) breaks out of VM: It gains access to another VM or attacks the hypervisor via kernel (if no Application Armor (AppArmor) or Security Enhanced Linux (SELinux) protection is active between VMs or between VM and the hypervisor).

- VM uses too many resources maliciously: Too much I/O or virtual Central Processing Unit (CPU)'s excessive usage, produces huge log files from the guest's console, fills up all available log space, or pushes gigabytes of logs per hour.

- VM gains access to wrong networks: It uses Virtual Local Area Network (VLAN) hopping, Address Resolution Protocol (ARP) poisoning, or any network hacks.

Many other vector attacks appear in the cloud. CCM provides a controls framework that gives a detailed understanding of security concepts and principles that are aligned to the CSA guidance in 13 domains:

- AIS: Application and Interface Security.
- AAC: Audit Assurance and Compliance.
- BCR: Business Continuity Management and Operations Resilience.
- CCC: Change Control and Configuration Management.
- DSI: Data Security and Information Lifecycle Management.
- DSC: Data center Security.
- EKM: Encryption and Key Management.
- GRM: Governance and Risk Management.

- HRS: Human Resources Security.
- IAM: Identity and Access Management.
- IVS: Infrastructure and Virtualization.
- IPY: Interoperability and Portability.
- MOS: Mobile Security.
- SEF: Security Incident Management, E-Disk and Cloud Forensics.
- STA: Supply Chain Management, Transparency and Accountability.
- TVM: Threat and Vulnerability Management.

In these 13 domains, there are currently 133 cloud controls. This number would change with cloud evolution (W01, Several authors. 2015).

3.2.1 Auditing Cloud

In traditional computing, the data was stored behind corporate firewalls. So, it was easy to secure the perimeter, harden the infrastructure, and secure the databases. Storing data in the cloud is different, given shared responsibility between the company, the Cloud Security Provider (CSP), and the chosen stack deployment. Nowadays, the CSP can be responsible for core competencies, including security and compliance, which includes encrypting data, hardening environments, and the backup and recovery processes.

The CSP could provide secure and compliant cloud services. Nonetheless, it is still up to the company to secure the overall application. Auditing the entire solution is a very complex task, as auditing occurs across multiple entities, such as the cloud service consumer and the cloud service provider (W02, Several authors, 2015).

Data and Cloud Security

Security is the most important concern for business and IT people in the cloud.

It is irrelevant where the data resides, as the threats are the same. Overall, there are some constraints around auditing, laws, compliance, customer requirements, and risks in terms of data stored in the cloud.

Auditing Cloud Applications

Auditors are responsible for validating the controls and processes for satisfying the requirements of a given set of constraints defined by a governing set of laws. There are many regulations today. Businesses need to determine which regulations must be applied. Companies must understand the standards of industry, business processes, and data requirements. In IT systems, the validation process is needed in different areas:

- Physical environment: perimeter security and data center controls.
- Systems and applications: security and controls of the network, databases, and software.

- Systems Development Life Cycle (SDLC): deployments and change management.
- Personnel: background checks, drug testing, and security clearance.

In traditional computing, an auditor would check the physical infrastructure against the different controls and processes that need to be audited. The auditors could concentrate on a physical machine and inspect the physical security of the data center. In cloud computing, this is not the case. Certain controls and processes map to a CSP. The auditor must rely on the auditing information produced and provided by the CSP. Without proof of compliance, the CSP could cause the customers to fail the audit. Companies do not accept that other companies have total control of their data and processes. They are reluctant to rely on another entity when it concerns critical aspects such as security, privacy, and regulations.

The IaaS provider in a public infrastructure will not allow an auditor of one of its tenants to access the infrastructure. Its own auditors for perimeter security, processes, and controls will be used instead. These auditors will not be given physical access.

In PaaS, the application stack is abstracted and managed by the CSP in addition to the infrastructure, and in some cases, database access and user's administration. The physical aspects of auditing are even more complex than in IaaS.

For SaaS, the provider is responsible for the entire application besides the infrastructure and the application stack. In these cases, consumers of SaaS have very limited responsibilities.

Regulations such as Health Insurance Portability and Accountability Act (HIPAA) or privacy levels such as Protected Health Information (PHI) are important because many customers will not do business with companies that offer cloud services that are not in compliance with those standards (B01, Kavis, 2014).

Regulations in Cloud

When building cloud services, standard and industry-specific regulations and controls need to be applied in cloud systems. The most important regulations relating to software are:

- ISO27001: International Standards for computer systems.
- Federal Information Processing Standards (FIPS): United States (US) Government standards for computer systems.

Security and privacy are more important when building cloud services:

- Statement on Standards for Attestation Engagements (SSAE)-16: Controls finance, security, and privacy.
- Directive 95/46/**ec!** (**ec!**): European security and privacy controls.
- Directive 2002/58/Electronic Communications (EC): European e-privacy controls.
- PCI DSS: Security and privacy of credit card information.

- Federal Risk and Authorization Management Program (FedRAMP): United States (US) Government security standards for cloud computing.

Industry-specific regulations need to be taken into account, especially when critical data or specific companies are using the cloud solution.

Audit Design Strategies

Before designing an audit design strategy in a new cloud application, one needs to identify all possible regulations that could be applied based on requirements from the customers and the industry.

Once the list of regulations is established, the next step is to create a work stream in the product roadmap for auditing based on data management, security management, centralized logging, SLA management, monitoring, disaster recovery, Systems Development Life Cycle (SDLC) and automation, operations and support, and organizational change management.

Another factor in audit strategy is the maturity of the consumer. For start-ups, getting to the markets quickly is much more important than passing the audits. So, less effort would be devoted to auditability in these cases. The start-ups cannot ignore auditing requirements, but efforts can be made a later stage (B01, Kavis, 2014).

3.2.2 Security Design in Cloud

In a traditional model, vendors provide security in enterprises by storing data in services such as Active Directory and providing Single Sign-On (SSO). The commercial software products run within the buyer's perimeter behind the corporate firewall.

In cloud computing, vendors have more responsibility to secure the software on behalf of the cloud consumers. Since consumers are giving up control and often allowing data to live outside of their firewall, vendors need to comply with various regulations.

For this reason, there is a common myth that critical data cannot be secure in the cloud. The reality is that security must be ensured regardless of where the data lives. It is not a matter of where the data resides, but of how much security is built into the cloud service.

Regulations such as Payment Card Industry Data Security Standard (PCI DSS) and Health Insurance Portability and Accountability Act (HIPAA) do not declare where the data may or may not reside. Regulations dictate that Personally Identifiable Information (PII) must be encrypted (A03, Several authors, 2013).

How much Security is Required

The level of security required for cloud services depends on different factors such as a target industry, data sensitivity, customer requirements, and risk tolerance.

The target industry often determines what regulations are in scope. Customer expectation is a factor that determines which security controls need to be in place.

The sensitivity of the data within cloud services has a major impact on the security requirements. Risk tolerance can drive security requirements.

Once companies consider these factors to determine how much security is required for their cloud service, security requirements will be evaluated to determine if a solution is already available in the marketplace or if the requirement should be implemented internally.

Responsibilities for each Cloud Security Model

For different service models, providers will be responsible for supplying security in different dimensions. In IaaS, the vendor supplies infrastructure security. In PaaS, the vendor supplies application stack security and infrastructure security. In SaaS, the vendor supplies application security, application stack security, and infrastructure security (B01, Kavis, 2014).

Security Strategies

There are three key strategies that manage security and an additional one for the areas to focus in a cloud-based application:

- Centralization: It refers to the practice of consolidating a set of processes, security controls, policies, and services, reducing the number of places where security needs to be managed and implemented. All the security controls in relation to the application stack should be administrated from one place. There should be limited paths for users and systems to gain access. Customers must enter in the same way so that they can be monitored. Appropriate controls and credentials are required to enter parts of the systems that other users are not allowed into. Policies should be centralized and configurable, so that changes are possible and tracked easily.
- Standardization: Security should be thought of as a shareable core service across the enterprise, instead of a solution for a specific application. Industry standards use Open Standard to Authorization (OAuth) and OpenID (OID) when connecting to third parties. Lightweight Directory Access Protocol (LDAP) for querying and modifying directory services such as Active Directory (AD) are highly recommended. Standardization applies to three areas, thus subscribing to industry best practices when implementing security solutions. Second, security should be implemented as a standalone set of services shared across applications. Third, all of the security data outputs should follow standard naming conventions and formats.
- Automation: Development and deployments will take too much time if automation is not used. Automation is important, for it is possible to scale automatically as demand increases or decreases; so, no human intervention is required. All cloud infrastructure resources should be automated in order to ensure the latest security patches and controls are automatically in place.
- Protection Detection Prevention (PDP): Protection is where all security controls, policies, and processes are implemented to protect the system and the company from security breaches. Detection is the process of mining logs, triggering events, and proactively trying to find security vulnerabilities across the

systems. Prevention arises if some security issues are detected, which is when actions need to be taken to prevent further damage (A04, Barker E. Barker W. Burr W. Polk W. Smid M. 2012).

In order to secure cloud-based systems, there are different areas where security controls need to focus:

- Policy enforcement: These are rules used to manage security in a system. These policies are maintained at every layer of the stack. The best practice is to create a template for each unique machine image that contains all of the security policies around access, port management, encryption, and so on.
- Encryption: Sensitive data processed in the cloud must be encrypted. Secure protocols must be used not only for data transit, but also where the data is stored—such as a database or a file system.
- Key management: This refers to the cryptographic keys (public and private keys). Objects are generally protected by a public key that can only be decrypted by the corresponding private key. This gives an advantage for an authorized person granted access to a system. The data cannot be decrypted without the corresponding key. Key management protects and stores these keys in a centralized place (never storing them on the same servers that they are protecting) and supplies the usage of the keys based on a single secure method of requests.
- Web security: One common way to compromise a system is by breaking web-based systems because web security is very dynamic. Attackers figure out new ways of attack: SQL injection, user session hijacking, and data in transit interception are some examples of these attacks. Using updated and patched web leverage frameworks is a good way to protect from web security threats. Another way is running proactively and continuously web vulnerability-scanning services.
- Application Programming Interface (API) management: Some of the advantages of the cloud-based architectures is how easily different services can be integrated by APIs. Moreover, this will create security challenges as each API in the system might be accessed over the web. In this case, it is expected that those APIs support OAuth or OpenID. In the worst case, some basic authentication over SSL could be used.
- Patch management: Some of the regulations require patching servers at least every 30 days. Auditors need to see proof that patching is applied properly and log what has been applied. Automation is desired for the patching process. For this purpose, a golden image method could be used. This will be an image of the system with the latest software upgraded and security patches applied. Applying patches in existing servers is not desired. It is suggested to create new servers and destroy the old ones, as this is simpler and less risky. If major issues occur when the new golden images are deployed, it will be safer to deploy the previous images than to revert patches in an existing image.
- Logging: This refers to the collection of all system logs from the infrastructure, application stack, and applications. As a good practice, it is recommended to write a log entry for every event that occurs in the system, especially the events related to users and systems requesting access.
- Monitoring: This refers to the process of watching over a system in order to provide information about activities and status on a system. Monitoring involves looking at real-time activity, but also mining log files.

- Auditing: This involves reviewing security processes and controls to check that the system is complying with the required controls of regulations, meeting security requirements, and SLA (A04, Barker E. Barker W. Burr W. Polk W. Smid M. 2012).

Controls in cloud computing are critical. Key management allows users and nodes to connect to other nodes or services and perform operations. For each operation, a policy enforcement is evaluated and logs this transaction, which is sent to a Cloud-centralized logging entity for monitoring and auditing the activity within the system.

3.2.3 Cloud-Centralized Logging

After moving from traditional client-server architectures into the distributed cloud-based architectures, the storing of logging information has to be separated from the servers where logs are created. The dynamism and elasticity of cloud applications ensures that information is not lost once cloud resources leave.

Log File uses

Log files are useful pieces of information about different behaviours of the systems—such as database activity, error and debugging information, user access, and so on. Log files have many uses: troubleshooting while debugging information and error messages, security when tracking user accesses, auditing by providing a trail of data, or monitoring that will help to identify trends, anomalies, and thresholds.

However, in a distributed environment with several servers, finding data in logs could not be trivial.

On the other hand, locking down access to production servers is necessary. By having a logging strategy that centrally maintains the logs on a separate server farm, the administrators will be able to remove access from all servers if needed (B01, Kavis, 2014).

Logging Requirements

In centralized architectures, there are two key requirements: Direct all logs to a redundant and isolated storage area, and standardize log formats.

First, all logs are directed to syslog instead of written directly to disk. Syslog on each server pipes to a dedicated logging server farm. Once the data arrives on the logging server farm, the data is transformed into a No Structured Query Language (NoSQL) database, being able to search the logs, schedule jobs, trigger alerts, and create reports for the end users. Figure 4 illustrates the centralized logging strategy.

This strategy has several benefits, allowing administrators to block access from all servers in production environments. Auditing becomes simpler since all logs are in one place. Data mining and trend analysis become feasible because of the NoSQL database. Implementing Intrusion Detection System (IDS) becomes simpler because the tools can run on top of the central logging database. Loss of log data is minimized as it is not stored in the local disk of servers that may be de-provisioned at some point.

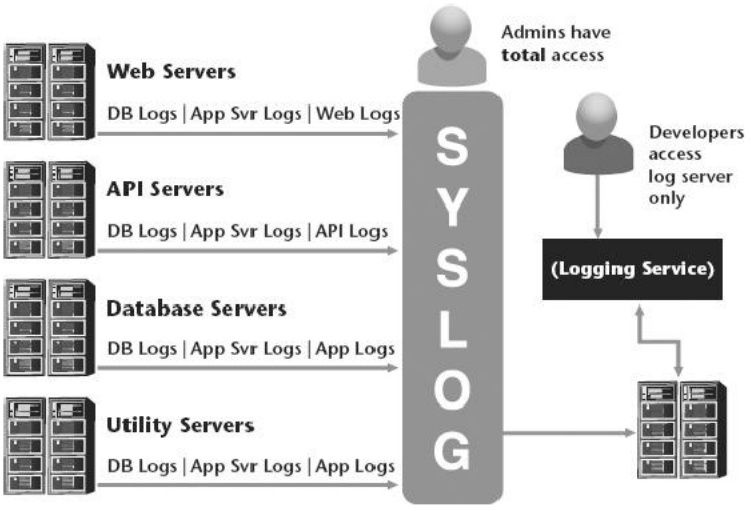


Figure 4: Centralized logging strategy (Architecting the cloud, p. 221).

Another option is to leverage an SaaS logging solution, sending the logs to a cloud-based centralized logging database as a service solution. In this particular case, the team no longer has to build, manage, and maintain logging functionality, and logs are maintained on an off-site scalable and reliable cloud infrastructure. In addition, if part of the data center goes down, the logs service will not be impacted.

The second action is standardizing all log formats, and naming conventions, severity levels, and error codes for all messages. This gives value to the data having a common log message format. Request For Comments (RFC) 5424 explains the severity codes. It will be necessary to create a common vocabulary for error descriptions, and including tracking attributes such as date, time, server, module, or API name is crucial for optimizing searches and producing consistent results.

Standardizing logs is a key strategy for increasing automation and proactive monitoring leading to a higher SLA (B01, Kavis, 2014).

3.2.4 Monitoring Cloud

In general, cloud services are built to always be enabled. The customer expects to be able to use the service 24 hours a day, 365 days a year. A huge amount of engineering is required in order to provide high levels of uptime, reliability, and scalability.

There are two flavours of monitoring: proactive and reactive.

Reactive

The goal of reactive monitoring is to detect failures.

Tracking the consumption of memory, CPU, and disk space of the servers, ping tools to check if websites are responding, and the throughput of the network to detect symptoms of possible failures are reactive types of monitoring.

Proactive

The goal of proactive monitoring is to prevent failures.

To prevent failures, first it is necessary to define baseline metrics for a healthy system. Then, patterns must be monitored to detect when data is trending towards an unhealthy system and fix the problem before reactive monitoring sounds warnings.

Reactive vs. Proactive

Combining reactive and proactive monitoring is the best practice when implementing cloud services. Both monitoring strategies are needed to find and resolve issues early, before system and customer are affected (B04, Henderson C. 2006).)

Monitoring Requirements

Monitoring helps to track the systems that are behaving with their expectations. These expectations are defined in the SLA between the cloud provider and the cloud consumer. In order to ensure the SLA is met, each SLA must be monitored, measured, and reported.

Apart from SLA, many cloud services are distributed systems composed of many parts, and all these parts of the system have a point of failure that needs to be monitored.

Roles in the organization would need different information from the system to ensure that the system functions properly: Front-end developers would need page-load times, API performance, network performance, and other information. Database architects may need metrics regarding database server, memory, cache, CPU utilization, and metrics of SQL statements and response times. Administrators would need to see metrics such as Requests Per Second (RPS), disk space, memory utilization, and CPU. Product Owner (PO) might need to see visits per day, new users, cost per user, and other business metrics.

These metrics will provide and determine if the system is behaving correctly and if it has the desired behaviour. It would also help to know the success of each deployment of software that could be compared against the baseline.

Different categories must be monitored. These include performance, quality, throughput, Key Performance Indicators (KPI), security, and compliance. However, not all categories are applied in each layer of a cloud-based solution (B04, Henderson C. 2006).

Strategies

There are different strategies of monitoring for the different categories based on the business model and target application.

Monitoring strategy metrics in user layer:

- Performance: This measures the behaviour of the customers using the system

(or in some cases, that of another system that interacts with the monitored one):

  - Number of new customers.
  - Number of unique visitors.
  - Number of page visits.
  - Average time spent on site.
  - Revenue per customer.
  - Bounce rate (users left without viewing pages).
  - Conversion rate (users who performed desired actions based on direct marketing).

- Throughput: This measures average rates of data moving through the system.
  - Concurrent users.
- Quality: This measures the accuracy and success of user registration and access.
  - Number of fails of process registration.
  - Number of fails of accessing the system.
- KPI: This shows if the system is meeting the business goals.
  - Revenue per customer.
  - Revenue per time.
  - Incoming customer calls in a certain period.
  - Jobs completed in a certain period.
  - Site traffic.
  - Shopping abandonment rate.
- Security: This focuses on mining log files, and discovering patterns of successful and unsuccessful attempts of attacking the system.
- Compliance: This provides alerts when parts of the system are falling out of compliance.
  - Track enforcement of policies mined in log files.

Monitoring strategy metrics in application layer:

- Performance: The goal is to measure how the system responds:
  - Time to load pages.
  - Uptime.
  - Response times for different API, reports, or queries.
- Throughput: The goal is to measure how much data the system can transmit to the end user.
  - Transactions Per Second (TPS).
  - RPS.
  - Clicks per second.
  - Page visits per second.
- Quality: This measures erroneous data.
  - Failed transactions.
  - Number of HTTP response codes between 400 and 500.

- KPI: The product team establishes what the metrics are, as each business model is unique.
- Security:
  - Failed authentication attempts for every component.

Monitoring strategy metrics in stack layer:

- Performance: The goal is to measure the components, such as the Operating System (OS), application server, database server, and so on:
  - Average response times between different systems.
- Throughput: This diagnoses issues within the system:
  - Nagios is used to gather various metrics.

Monitoring strategy metrics in infrastructure layer:

- Performance: This measures physical infrastructure, such as servers, networks, routers, or switches.
  - Summary health of their infrastructure with some basic indicators.
- Throughput: This measures the flow from physical hardware and network devices (B01, Kavis, 2014).

The metrics which are applied for monitoring the strategy depends in what layer the cloud requires more tracking; user, application, stack or infrastructure.

3.2.5 Disaster Recovery in Cloud Computing

In distributed environments, such as cloud-based solutions, there are many parts on the system that can fail. In cloud computing, systems should be designed expecting everything can fail, so that the system is prepared for these potential failures.

Costs

Strategies for disaster recovery in the cloud are similar to those for traditional data centers, but the implementation will be different. There are mainly three variables from the business perspective:

- Recovery Time Objective (RTO): This refers to the time within which the business requires that the services be back up and running.
- Recovery Point Objective (RPO): This is the amount of time in which data loss can be tolerated.
- Value of recovery: This is a measurement of how much it is worth to the company to mitigate disaster scenarios.

The business should determine the RTO, RPO, and value of recovery for each functional area of the system architecture (B01, Kavis, 2014).

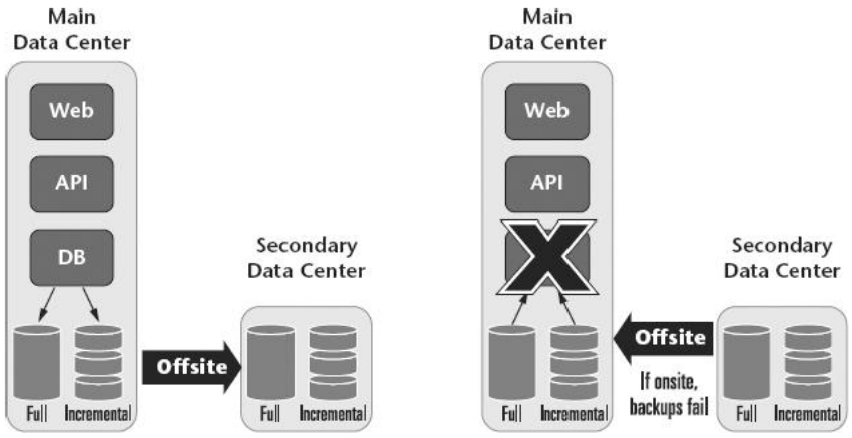


Figure 5: Classic backup and restore (Architecting the cloud, p. 268).

Strategies in IaaS

In IaaS, disaster recovery strategies are much more involved than other service models because the consumer is responsible for the application stack.

Having regions and availability zones is one strategy for IaaS, where the regions are located across the globe while the zones are independent virtual data centers in a region. Having redundancy across multiple zones will help to maintain uptime if there are any outages.

An alternative is to build redundancy across regions. However, this method is more complex and expensive, as moving data between zones incurs charges for the data transfer and introduces extra latency.

There is a standard set of best practices for recovering from a disaster in IaaS. Figure 5 shows the classic backup and restore: daily full and incremental backups are created and stored to a disk service provided by the vendor, and copied in a secondary data center and other third-party vendor.

Active-Passive cold: A secondary data center is waiting to take the primary data center in case of a disaster. However, the secondary data center is not running. Instead, it is waiting for a batch set of actions that will turn on in case of an emergency. Figures 6 and 7 show the Active-Passive cold scenario and its transition.

Active-Passive warm: A secondary data center is waiting to take the primary data center in case of a disaster. The secondary data center is running, decreasing the downtime if an outage occurs. This can be used to load balance using the secondary data center. Figures 8 and 9 show the Active-Passive warm scenario and its transition.

Active-Active hot: All the compute resources are being used at all times and the data center will not have any downtime at all, so that it does not miss any transaction. The database uses master-slave replication between data centers. If a primary data center fails, the database at the secondary data center becomes the master. Once all data in different data centers are in sync, the original primary data center will become the master again (B01, Kavis, 2014). Figures 10 and 11 show the Active-Passive hot scenario and its transition.

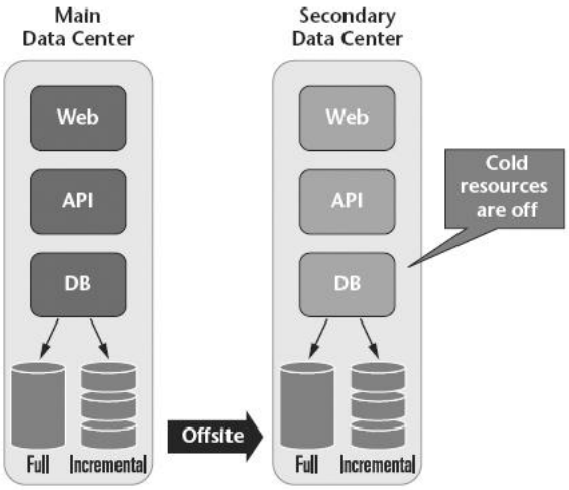


Figure 6: Active-passive cold (Architecting the cloud, p. 270).

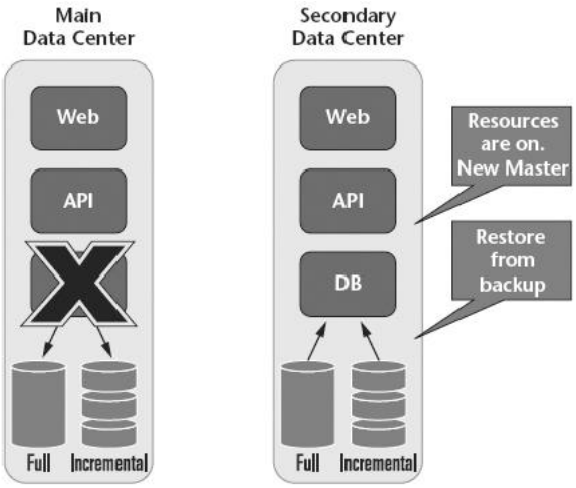


Figure 7: Active-passive cold (Architecting the cloud, p. 270).

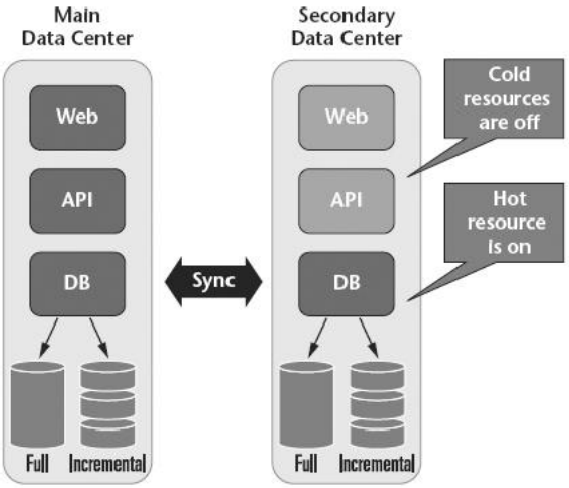


Figure 8: Active-passive warm (Architecting the cloud, p. 272).

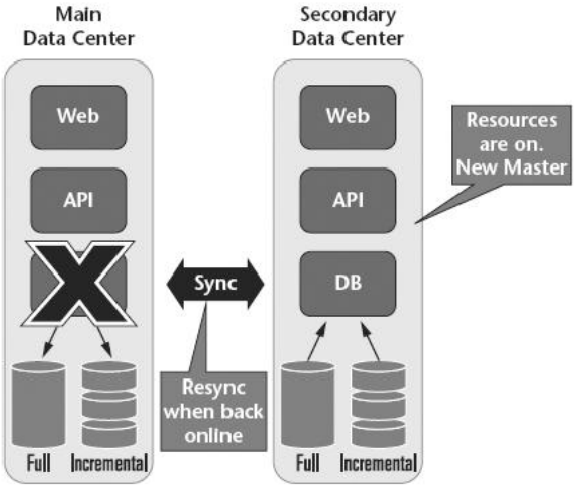


Figure 9: Active-passive warm (Architecting the cloud, p. 272).

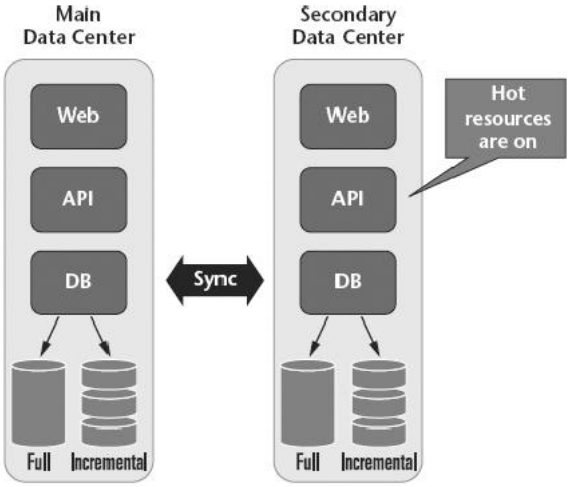


Figure 10: Active-passive hot (Architecting the cloud, p. 274).

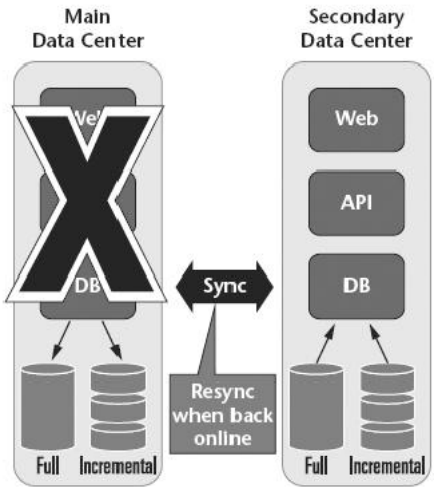


Figure 11: Active-passive hot (Architecting the cloud, p. 274).

Strategies in PaaS

In public PaaS, the vendor is responsible for applications built on top of the platform. When a disaster occurs, the consumer depends on the recovery plan from the PaaS provider.

Private PaaS is a less risky alternative because the vendor abstracts the development platform by installing and managing the application stack being simpler and automated; however, the consumer has to manage the infrastructure. When a disaster occurs, it is the consumer who controls the situation, as he manages the infrastructure (B01, Kavis, 2014).

Strategies in SaaS

For SaaS, there should be a plan if the service is not available for a long time. At least a software escrow is recommended, holding the Internet Protocol (IP) from the vendor in an independent area hold of a third party, being able to release the vendor from the buyer point of view in case of business impact. In other words, the buy is ownership of the data.

Having two different SaaS vendors to protect against outages could be feasible in some cases, depending on the business impact.

From the risk mitigation point of view, it is highly recommended to extract data regularly (B01, Kavis, 2014)).

Strategies in Hybrid

In a hybrid clouds, a company can split workloads between the public and private clouds. For the workloads that run in the public cloud, the private cloud could be configured to fail over data center; as for the workloads that run in the private cloud, the public cloud could be used as the fail over data center. For this strategy, both public and private clouds must run similar services (B01, Kavis, 2014).

The above strategies are applied over a cloud installation. Currently there is an existing platform called Openstack which act as a base OS for providing cloud services.

3.3 Openstack Components

As OpenStack is a complex software, it is not only split in the VIC, VIM and Fuel. There are additional required components. See Figure 12 for an architectural overview of Openstack fuel component.

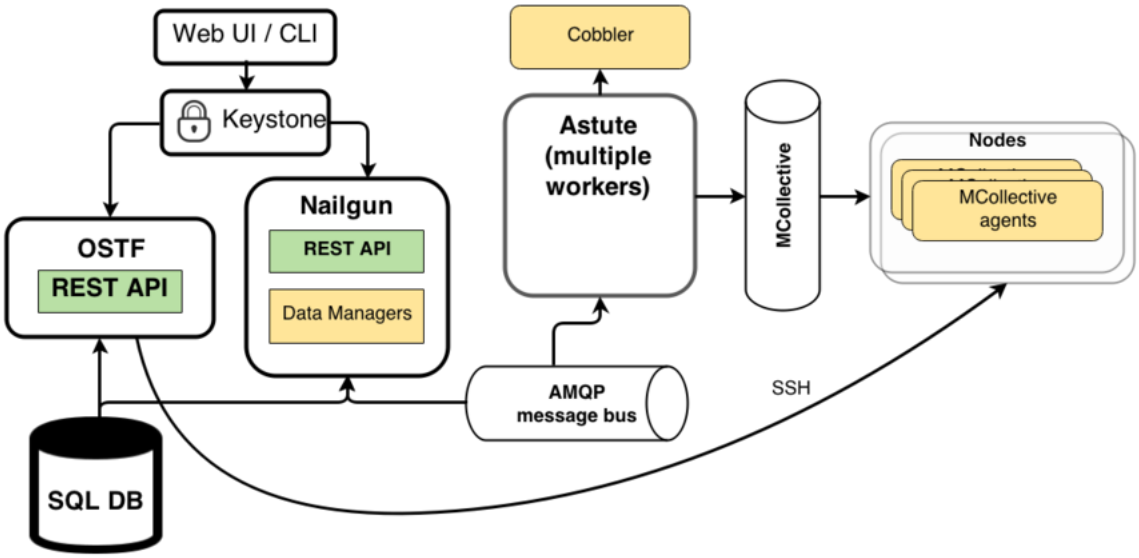


Figure 12: Fuel architectural overview (Introduction to Fuel, openstack.org)

OpenStack components are as follows:

- **Neutron**: This is an L2 and L3 networking environment for a VM.
- **Nova**: This represents computing resources of a VM, CPU, and memory.
- **Cinder**: This provides block storage devices to the VM.
- **Glance**: This provides image storage of boot images for VM.
- **Ceilometer**: This collects measurements of the utilization of the physical and virtual resources.
- **Heat**: This orchestrates the installation of applications.
- **Swift**: This replicates storage of files for internal use.
- **Keystone**: This provides identity, token catalogue, and policy services (W12. Several authors. 2015).

The Openstack components and its interaction are shown in Figure 13.

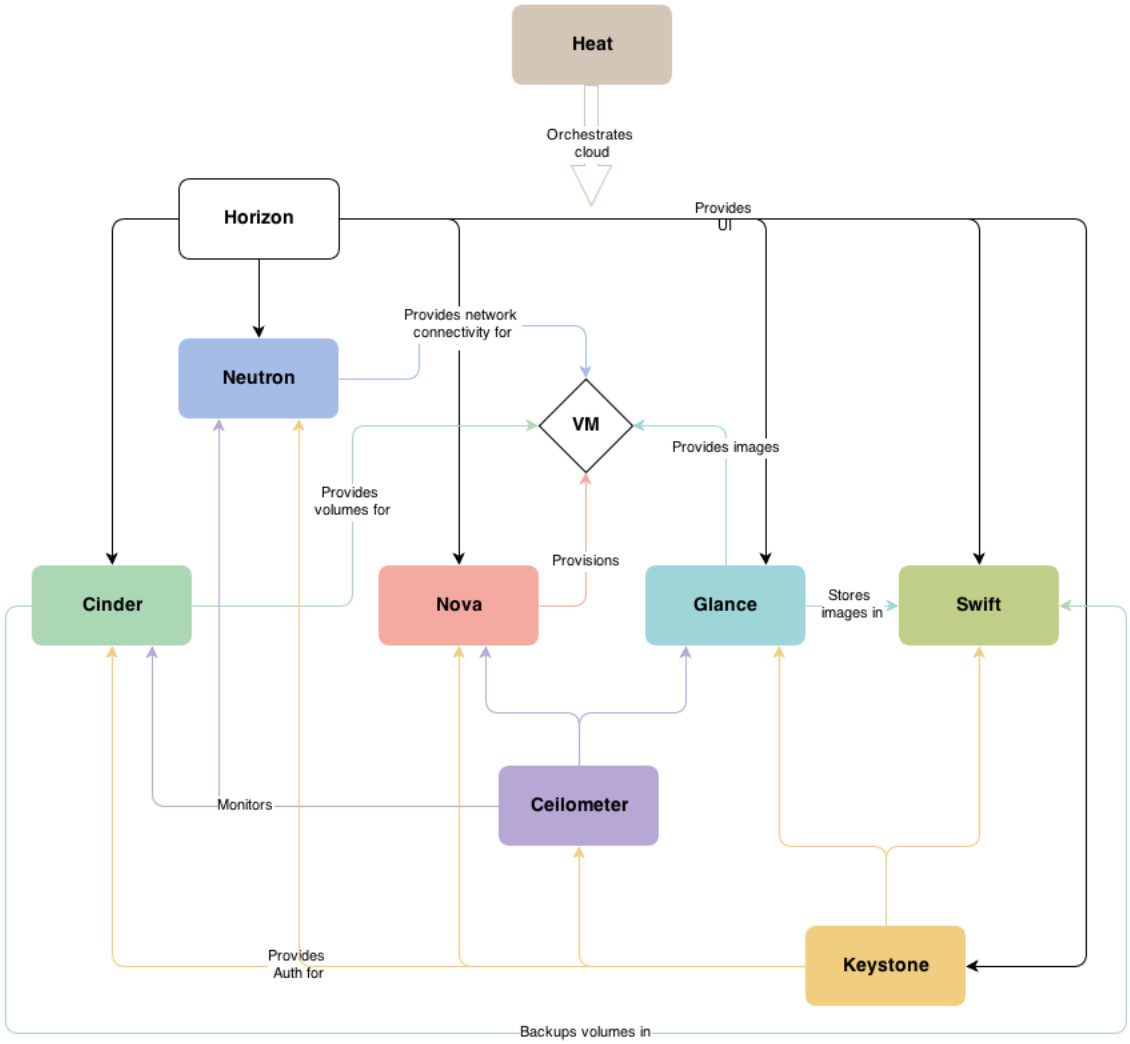


Figure 13: OpenStack architectural overview (Introduction to OpenStack, openstack.org)

Openstack base will run the Cloud Execution Environment (CEE) for the ECS solution.

3.4 CEE Components

CEE consists of three software components. The first part is the VIC, which provides the needed infrastructure support for running a cloud environment. This manages all activities in the CEE. All OpenStack services are included in the VIC, but also functionality is included for maintaining and monitoring the CEE infrastructure.

The second part is Virtual Infrastructure Managing (VIM), which provides the management interfaces, Command Line Interface (CLI) and Graphical User Interface (GUI), for managing the virtual infrastructure. It also manages the system for the CEE by HTTP access.

The last part is Fuel, which adds installation, upgrade, and management support for an OpenStack region.

Fuel is an open-source development and management tool for OpenStack. Fuel is

delivered as an image that is used to boot a Fuel master image and docker containers are used to run the Fuel logic. Fuel provides GUI, CLI, and API interfaces for management (W11. Several authors. 2015). See Figure 14 for CEE architectural overview.

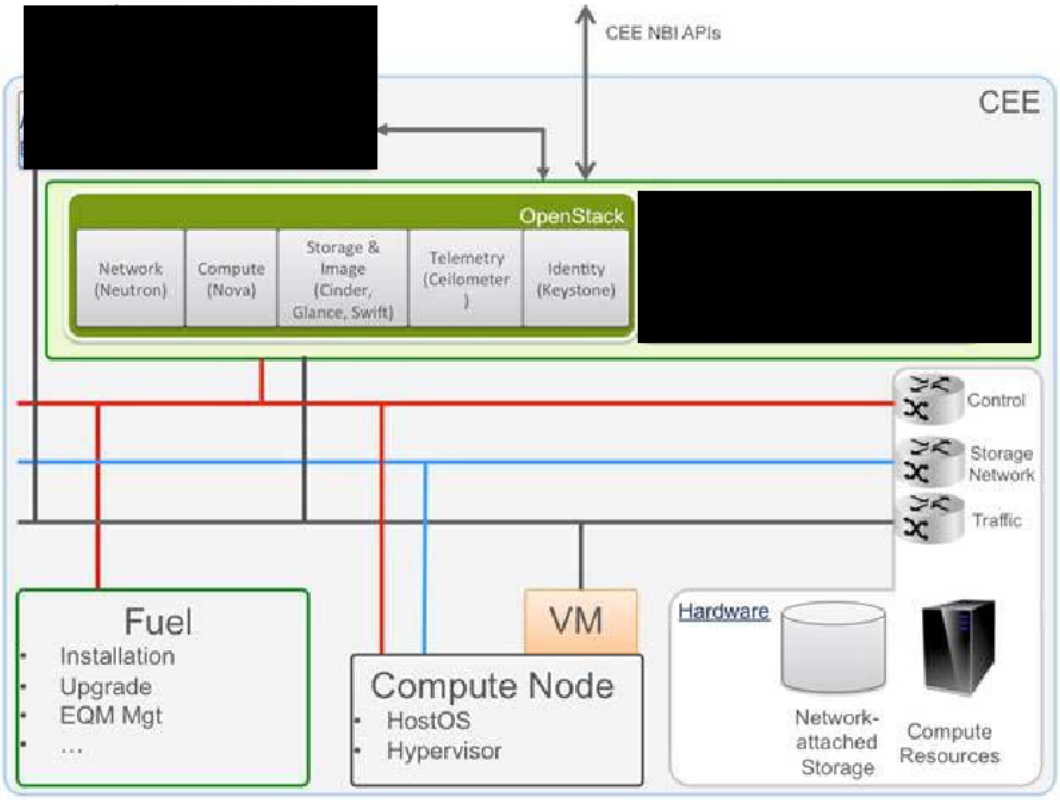


Figure 14: CEE architectural overview (Cloud Execution Environment, ericsson.com)

ECS provides external API allowing for north-bound applications to interact and control the whole or parts of the infrastructure depending on whether the tenant or the cloud operator interacts.

Openstack is interacting with the rest of cloud entities, and acts as an interface for the installation, integration, and management of the cloud.

Understood the nature of the cloud and its principles; including the different services models, scope and components, it is time to elaborate the requirements concerning the security.

3.5 Security Requirements

The implementation of the ECS is required to follow the company's security requirements and focus on the biggest possible threats. Being compliant within security requirements requires proper testing to achieve this goal.

3.5.1 Security Baseline Requirements

Ericsson has generic baseline security requirements for every product that is delivered. This guideline defines the level of security in different areas to reach an acceptable level of security.

The baseline security requirements document covers risk management, hardening, access control, security event logging, traffic protection, data protection, and packet filtering. Technical specifications are also defined based on the input material: Third-Generation Partnership Project (3GPP), Internet Engineering Task Force (IETF), American National Standards Institute (ANSI), The National Institute of Standards and Technology (NIST), Common Criteria, International Organization for Standardization (ISO), and Open Web Application Security Project (OWASP).

The risk management area refers to the risks for each product that should be investigated and documented in the risk assessment. Unacceptable risks must be mitigated with risk treatment actions.

Hardening refers to the process of securing a system by reducing its surface of vulnerability. It is a design, a configuration, and a deployment issue. Hardening includes removal of unnecessary software, installing the latest patches, disabling insecure devices, and replacing default passwords.

Access control covers the protection of the system resources against unauthorized access. Access control is regulated according to a security policy.

Security event logging requirements involve recognizing, recording, and storing information related to relevant events. The resulting event log can be examined to determine which relevant events took place and who is responsible for them. It should be possible to trace and log all the security-related events in a given node.

Traffic protection contains requirements for security mechanisms of protecting IP-based communication within a node.

Data protection covers requirements related to the protection of data at rest. Data is an important corporate asset that needs to be safeguarded.

Packet filtering includes the access control to and from a network by analysing the incoming and outgoing packets and letting them pass or blocking them.

3.5.2 ECS Requirements

The two main areas that need to be covered from the security perspective in the ECS project are: first, disabling and randomizing in real-time all the possible default passwords for different deployed systems in order to prevent unauthorized access; and second, setting a proper system of authenticating efficiently different users for different roles and capabilities in a data center with a massive amount of nodes with different services and levels.

Other project-specific requirements regarding security are not evaluated in this thesis, as other teams are responsible. These areas are not part of the purpose of this thesis.

4. Test Strategy

In order to fulfil all the security requirements for the functionality provided in the Ericsson cloud, a set of tests is required. First, it was necessary to choose the test strategy.

The test strategy describes the testing approach of the software development cycle. It helps to inform project managers, testers, and developers about some issues of the testing process. This includes the testing objective, methods of testing new functionalities, total time and resources required for the project, and the testing environment (B05, Kaner C. Falk J. Nguyen H. 2006).

In this particular case, the test strategy is narrowed to the security scope.

4.1 Test Coverage

From the perspective in ECS, the following areas need to be covered in order to provide a competent cloud product that is secure enough for the customers. The test coverage is an outcome from the 'Security requirements' chapter of this thesis.

- Services: This involves focusing on port scanning, listening services, expected transport used protocols, and interface binding. Pre-requisites identify which nodes need to be scanned as well as what services and protocols must be used.
  - Verify services using netstat utility to see active connections.
  - List services using The Network Mapper (Nmap) port scanner and see how those ports appear.
  - Document which ports and services are filtered by a firewall or system configuration.
  - Check transport protocols used for these services (Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).
  - Run version scanning using Nmap port scanner.
  - Run OS fingerprint using Nmap port scanner.
  - Verify interface binding where services are mapped.
  - Verify traffic separation and ensure no leakage happens by using fuzzing tools.
  - Run dictionary and brute force attacks against services and verify Pluggable Authentication Modules (PAM), Access Control List (ACL), and other security modules, features, or configuration options, which ensure protection against password cracking.

- Vulnerabilities and Denial of Service (DoS): This is a vulnerability scan based on services version, fuzzing, and robustness protocols. As a pre-requisite, it is necessary to identify possible DoS attacks that will affect the system.

  - Run vulnerability scanners such as Nessus to not allow any critical bug active in the system.
  - Run fuzzing tools such as codenomics to check leakages and robustness of protocols.
  - Run traffic generators such as IXIA to flood the networks. This might detect possible DoS and provides an output of performance of the networks.
  - Generate malformed attacks using scapy or hping utilities that would help to ensure the system is behaving properly and services are responsive.
  - Malware testing that generates a fake virus could also be performed to ensure Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) are responding as expected, as well as some alarms to isolate the threat.
  - Running additional tools regarding security in web services such as XSS.

- Hardening: This includes default passwords, services running by default, users running services, configuration and file permissions, and administrator accounts disabled. It is needed to identify which users and groups are allowed to do different actions in each system and to identify the list of default services that systems run as a prerequisite. For each node, it is needed to verify:

  - No default passwords are used.
  - Administrator accounts are not enabled by default.
  - There are no unexpected services by default.
  - Services are not run as privileged accounts like root.
  - Configuration and file permissions are set properly, as is the storage area.
  - Password policies are followed (no weak passwords are allowed).
  - Every account can be disabled.
  - SELinux policy configurations are verified after performing different transactions.

- Encryption: This includes encrypted traffic, encrypted passwords, and usage of secure protocols. To perform, the following checklist is needed to identify every traffic flow between two components for the whole system.

  - Not allowing plain text data to be transferred between two components of the system.
  - Passwords are not sent in plain text.
  - Certificates for authentication are used if needed for some components.
  - Critical data such as passwords are properly encrypted and not accessible to other users.
  - No unsecure protocol is used, (for example, Lightweight Directory Access Protocol (LDAP) over Transport Layer Security (TLS) or LDAP over SSL (LDAPS) instead of plain LDAP).

- Logging: Logging relates to security issues (local and remote logging). It is a prerequisite to identify security incidences that will trigger the logs.

  - Verify security incidences are reported to the log on the machine on which it happens.
  - Check security incidences are transferred to a remote centralized node, using a log collector.

- Verify if it is possible to identify which security incident or action has been performed by which user (needed for forensics).
- Verify how accurate logs based on incidents are and if they give valuable information (quality and error standardized errors and incidents).

When performing these test operations, it is important to take into account:

- Tests should also be performed with traffic in order to simulate having real CPU, memory, and network usage from the different systems. - SQL injection or XSS attacks do not always need to be performed, as these services are not always provided via a web interface. - Some tests could be difficult to perform, such as hardware specific for Hardware Security Module (HSM) or Trusted Platform Module (TPM) related with Intelligent Platform Management Interface (IMPI).

These tests are applied to the following identified entities:

- Hosts OS.
- Cloud controller (VIC) OS.
- Ethernet switches.
- Dashboard.
- LDAP Authentication system.
- Network protocols.
- EMC (storage).

Defined the strategy and test coverage, it is a matter of priority to implement them. Next chapter defines these priorities.

4.2 Priorities

Based on the input information from the Test Coverage, Test Strategy, and the Security Requirements, the functionality is implemented first. Integration and Verification (I&V) team verifies releases. Finally, test coordinators align with the rest of the organization. The following priorities has been decided for the testing part:

Testing services in the different nodes:

- Services listened to on the controllers and compute nodes in VIC.
- Services listened to in VIM Virtual Machine (VM).
- Services filtered by the firewall.
- Encryption between components.
- Auditability for incidences in different services.

Testing the access control:

- Identity and Access Management (IdAM) as a centralized access control.
- Lightweight Directory Access Protocol (LDAP) databases that store system users.

- Identity and Access Management (IdAM) tools used for administrate users.
- Group managing permissions is verified.
- Remote Authentication Dial In User Service (RADIUS) for administrate switch users.

Testing the hardening of the system:

- Password policies' strength.
- Password policies for history.
- Robustness of randomized passwords.
- Administrative and default accounts are disabled in different nodes.
- Verification of services running with their own user and domain.
- Verification of default accounts has been disabled.

Given a list of services, access controls and which systems requires hardening, it is time to describe the tools used to perform this goal. The following section describes the tools which have been being used for that purpose.

4.3 Testing Tools

Once Test Coverage and Priorities have been decided on and described, it is necessary to select which tools are powerful enough and most suitable to achieve the goal of testing the ECS solution from the security point of view.

After some analysis and discussions with other members from cloud security and I&V from the Product Development Unit (PDU) cloud, the following technologies and tools have been selected to perform security tests:

- **Nmap**: Port and network scanning.
- **Hydra**: Password cracking tool for different services.
- **Nessus**: Vulnerability scanner.
- **IXIA**: Packet generator, which could be used for traffic leakage control and DoS attacks due to the high performance.
- **Scapy** and **hping**: Interactive packet manipulation programs that could be used for DoS attacks involving malformed and unexpected packets.
- **Tcpdump** and **wireshark**: Packet sniffer.
- **Defensics codenomics**: For fuzzing.
- **Burp**: Web vulnerability scanner.
- **W3at**: Analyses applications that communicate using the HTTP and HTTPS protocols.
- Small built-in utilities: **netstat**, **socklist**, **lsof**, **nc**, **iptraf**, **iperf**, etc.
- **Iptables**: Not as a test tool but used and affecting results.

Some additional tools can be used in development phases or in Continuous Integration (CI):

- **Lint**, **clint**, or **splint**: Static code analysis.
- **Valgrind** and **callgrind**: Run-time code analysis.
- **Puppet validate**: Puppet syntax validation for manifests.

These tools can be used to prevent or indicate possible errors, mistakes, or bugs in development phases rather than in testing phases. Some of them could be security issues.

From the above highlighted testing tools, some of them are critical in terms of testing and will be widely used. So, it is necessary to describe in detail the powerfulness of these utilities:

4.3.1 Nmap

Nmap is not just a simple port scanner. It is a utility for network discovery and security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime.

Nmap has several features, such as host discovery for identifying hosts on a network, port scanning that enumerates open ports on a target host, version detection by interrogating network services on remote devices, OS detection that determines the system and hardware characteristics of network devices, and scriptable interaction with the target by using Nmap Scripting Engine (NSE) and Lua programming language. Nmap follows a sequence described below:

1. Converts the target from a hostname into an Internet Protocol Version 4 (IPv4) address using Domain Name Server (DNS).
2. Pings the host, by default with an Internet Control Message Protocol (ICMP) echo request packet and a Transmission Control Protocol (TCP) Acknowledgement flag (ACK) packet to port 80 to determine if it is up and running.
3. Converts the target IP address back to the name using reverse DNS query.
4. Launches a TCP port scan of the 1,000 most popular ports listed in nmap-services. A Synchronization flag (SYN) stealth scan is usually used, but connect in case for no privileged users.
5. Prints the results to standard output.

Nmap is divided into different phases while performing the above actions:

1. **Script pre-scanning**: NSE uses a collection of special-purpose scripts to gain more information about remote systems.
2. **Target enumeration**: Nmap researches the host specifiers provided by the user. Nmap resolves these specifiers into a list of Internet Protocol Version 4 (IPv4) or Internet Protocol Version 6 (IPv6) addresses for scanning.
3. **Host discovery**: This discovers which targets on the network are online and, thus, worth deeper investigation.

4. **Reverse DNS resolution**: This looks up the reverse DNS names of all hosts found online by the ping scan. Sometimes, a host's name provides clues to its function, and names make reports more readable than providing only IP numbers.
5. **Port scanning**: Probes are sent and the responses (or non-responses) to those probes are used to classify remote ports into states such as open, closed, or filtered.
6. **Version detection**: If any ports are found to be open, Nmap may be able to determine what server software is running on the remote system. It does this by sending a variety of probes to the open ports and matching any responses against a database.
7. **OS detection**: On measuring these differences, it is often possible to determine the operating system running on a remote host.
8. **Traceroute**: Nmap contains an optimized traceroute implementation. It can find the network routes to many hosts in parallel.
9. **Script scanning**: Scripts running during this phase generally run once for each target host and port number that they interact with. They commonly perform tasks such as detecting service vulnerabilities, malware discovery, collecting more information from databases and other network services, and advanced version detection.
10. **Output**: Nmap collects all the information it has gathered and writes it to the screen or to a file.
11. **Script post-scanning**: The scripts in this phase can process results and deliver final reports and statistics.

Host discovery techniques used by Nmap:

- **TCP SYN ping (-PS portlist)**: This sends an empty TCP packet with the SYN flag set. The packet attempts to establish a connection. If the port closed, an Reset flag (RST) packet is sent back. If the port is open, a TCP three-way handshake begins. A TCP SYN/ACK packet will be the response from our machine instead of an ACK packet. This is sent by the kernel and not by Nmap, as a response to unexpected SYN/ACK. On UNIX boxes, sending raw TCP packets requires privileges. So, a workaround is to use syscall connect() again target port, if connect returns success or ECONNREFUSED the TCP stack must receive a SYN/ACK or RST and host is marked available. If hanging until timeout, host marked as down. The default port is 80.
- **TCP ACK ping (-PA portlist)**: This sends a TCP ACK packet when no connection exists, so that the remote host always responds with an RST packet, disclosing its existence in the process. If an unprivileged user or IPv6 target is specified, the connect workaround will be used. This workaround is imperfect because connect is actually sending a SYN packet rather than an ACK. The reason for offering SYN and ACK probes is to maximize the chances of bypassing firewalls, as many admins just block incoming SYN. The default port is 80.
- **UDP ping (-PU portlist)**: This sends an empty UDP packet to the given ports (except if data-length is provided). A UDP probe should elicit an ICMP port unreachable packet in return, which means the host is up and available. Other ICMP errors, such as host/network unreachable or TTL exceeded, are indicative of a down or unreachable host. If a port is reachable, most services

simply ignore it. This is why a high default port is chosen. But some services such as chargen could reply to an empty UDP packet disclosing that the machine is available. The default port is 31338.

- **ICMP ping types (-PE,-PP,-PM)**: These send ICMP type 8 (echo request) to the target, expecting a type 0 (echo reply) as a reply. An echo request can be sent with a -PE option. The -PP option will send an ICMP with a timestamp request (that could receive a timestamp reply that it is ICMP code 14). -PM will send an ICMP with an address mask query (which could receive an address mask reply that it is ICMP code 18).
- **IP protocol ping (-PO protocollist)**: This sends IP packets with a specified protocol number set in their IP header. If no protocols are specified, the default is to send multiple IP packets for ICMP (protocol 1), IGMP (protocol 2), and IP-in-IP (protocol 4). Note that for ICMP, IGMP, TCP (protocol 6), and UDP (protocol 17), the packets are sent with the proper protocol headers, while other protocols are sent with no additional data beyond the IP header (unless the data-length option is specified).
- **ARP scan (-PR)**: In an ethernet Local Area Network (LAN), when Nmap tries to send a raw IP packet, such as an ICMP echo request, the OS must determine the destination hardware (ARP) address corresponding to the target IP, so it can address the ethernet frame properly. This requires it to issue a series of ARP requests. The –send-ip option tells Nmap to send IP-level packets (rather than raw ethernet) even though it is a local network. So, performing an ARP scan could help with regard to the question of time and avoiding adding an incomplete ARP entry on the kernel ARP cache. You can spoof your MAC address with –spoof-mac if you are using this scan in some conference room.

Netfilter/iptables state seen with the option –state, which categorizes packets based on connection state. Different packet states are display in Table 1:

Connection state	Description
INVALID	packet is associated with no known connection.
ESTABLISHED	packet is associated with a connection that has seen packets in both directions.
NEW	packet has started a new connection or otherwise is associated with a connection that has not seen packets in both directions.
RELATED	packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer or ICMP error.

Table 1: Netfilter and iptables states representation

Port scan techniques used by Nmap:

- **TCP SYN stealth scan (-sS)**: This scan never completes TCP connections. It just sends TCP SYN segments.

  Figures 15, 16 and 17 show an example of ports open, closed and filtered respectively.

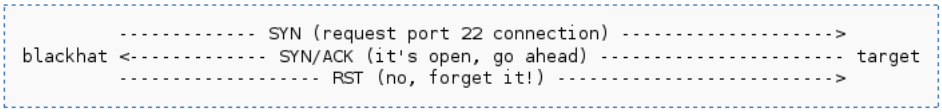


Figure 15: Example of port open discovered by TCP SYN stealth

Figure 16: Example of port closed discovered by TCP SYN stealth

Figure 17: Example of port filtered discovered by TCP SYN stealth

Nmap interpretation in TCP SYN stealth scans are shown in Figure 2:

Probe response	State
TCP SYN/ACK response	open
TCP RST response	closed
No response	filtered
ICMP unreachable error (type 3, code 1, 2, 3, 9, 10 or 13)	filtered

Table 2: Nmap interpretation for TCP scans

- **TCP connect scan (-sT)**: It uses connect syscall as this does not require root privileges. However, Nmap has less control over the syscall connect than with raw packets. TCP connections usually end with another handshake involving the Finalization flag (FIN) flag, but Nmap asks the host OS to terminate the connection immediately with an RST packet.

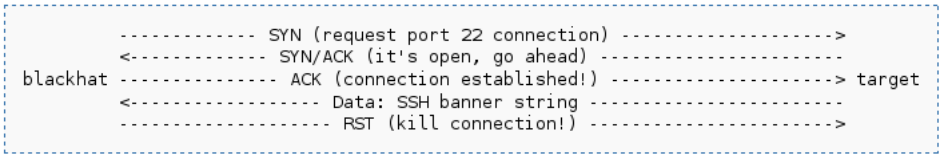


Figure 18: Example of TCP connect scan

- **UDP scan (-sU)**: UDP scanning is generally slower and more difficult than TCP. A UDP scan works by sending an empty (no data) UDP header to every targeted port. Based on the response or lack thereof, the port is assigned to one of four states.

  Nmap interpretation in UDP scans are shown in Figure 3:

Probe response	State
Any UDP response from target port	open
No response received	open or filtered
ICMP port unreachable error (type 3, code 3)	closed
Other ICMP unreachable errors (type 3, code 1, 2, 9, 10, or 13)	filtered

Table 3: Nmap interpretation for UDP scans

Open ports rarely respond to these probes, as the TCP/IP stack simply passes the empty packet up to the listening application, which usually discards it immediately as invalid. That is why Nmap cannot determine whether the port is open or filtered when there is no response.

UDP services generally define their own packet structure rather than adhere to some common general format that Nmap could always send. So, an Simple Network Management Protocol (SNMP) packet looks completely different than a SunRPC, Dynamic Host Control Protocol (DHCP), or DNS request packet. So, to send a proper packet for every popular UDP service, Nmap would need a large database defining their probe formats (located at nmap-service-probes). So, if the version scanning with -sV is performed, Nmap will send UDP probes to every open|filtered or open ports.

To speed up UDP scans (they are extremely slow because normally probes are not replied to and GNU/Linux is rate-limiting to avoid flooding the network), it is suggested to use options such as increase host parallelism (with –min-hostgroup 100), scan popular ports first (with –F option to scan the 100 most common UDP ports), add version detection scan against a given port number (with –version-intensity 0), scan from behind the firewall if possible (as they can slow down scans dramatically), skip slow hosts (with –host-timeout 900000), and use verbosity level enabled (with -v) to provide ETA for scan completion of each host.

- **TCP FIN, NULL, and Xmas scans (-sF, -sN, -sX)**: Based on the Request For Comments (RFC) 793, which says *'if the [destination] port is CLOSED... an incoming segment not containing an RST causes an RST to be sent in response'* and the next page, which discusses packets sent to open ports without the SYN, RST, or ACK bits set, saying *'you are unlikely to get here, but if you do, drop the segment, and return'*. So, when scanning systems are compliant with this RFC, any packet not containing SYN, RST, or ACK bits will result in a returned RST if the port is closed and no response at all if the port is open.

  The following operations are performed to execute this scan:

  – NULL scan will not set any bits (TCP flag header is 0).
  – FIN scan sets the TCP FIN bit.
  – Xmas scan sets the FIN, PSH, and URG flags, lighting the packet such as a Christmas tree.

  Nmap interpretation in the TCP FIN, NULL, and Xmas scans (see Figure 4):

Probe response	State
No response received	open or filtered
TCP RST packet	closed
ICMP unreachable errors (type 3, code 1, 2, 9, 10 or 13)	filtered

Table 4: Nmap interpretation for specific TCP SYN flags scans

The advantage of this kind of scan is that they can sneak certain non-stateful firewalls and packet filtering routers (because they are trying to block TCP connections with an SYN bit set and ACK cleared).

- **Custom scan types (–scanflags)**: These are scans that allow the user to specify any TCP flag combinations.

- **Custom SYN/FIN scan**: This allows you to specify arbitrary TCP flags in order to bypass some firewalls. Possible flags can be used in mashing together any of those combinations: URG, ACK, PSH, RST, SYN, and FIN. For example, this one sets everything (being the order irrelevant). In addition, it is possible to set the TCP scan type, telling Nmap how to interpret responses.
- **PSH scan**: By trying a customized PSH/URG or FIN/PSH scan, it gets a small chance of evading scan detection systems. So, the idea is to choose the FIN scan as a base type -sF and specify one of those flags.

- **TCP ACK scan (-sA)**: This scan is used only to map out firewall rules, determining if they are stateful or stateless and which ports are filtered. It is a probe packet that has only an ACK set, and when scanning systems:

  - Open and closed ports returning an RST packet: Nmap will label them as unfiltered (meaning they are reachable but it is undetermined).
  - Ports that do not respond or send some ICMP error back are labelled as filtered.

  Nmap interpretation in TCP ACK scans (see Figure 5):

Probe response	State
TCP RST response	unfiltered
No response received (even after retransmissions)	filtered
ICMP unreachable errors (type 3, code 1, 2, 9, 10, or 13)	filtered

Table 5: Nmap interpretation for TCP ACK scans

- **TCP window scan (-sW)**: This scan exploits an implementation detail of certain systems to differentiate an open port from closed ones, rather than always printing unfiltered when an RST is returned (as it happens with an ACK scan). This scan relies on an implementation detail of a minority of systems. So, it is not always trustable. However, it could help in cases where the FIN scan cannot distinguish between open and filtered ports.

  Nmap interpretation in TCP window scans (see Figure 6):

Probe response	State
TCP RST response with non-zero window field	open
TCP RST response with zero window field	closed
No response received (even after retransmissions)	filtered
ICMP unreachable errors (type 3, code 1, 2, 9, 10, or 13)	filtered

Table 6: Nmap interpretation for TCP window scans

- **TCP maimon scan (-sM)**: This described a technique in Phrack magazine 49 when it was invented by Uriel Maimon. The technique is exactly the same as for the Xmas, NULL, or FIN scan except that the probe is FIN/ACK. According to the RFC 793 (TCP), an RST packet should be generated in response to such as a probe whether the port is open or closed. However, many BSD-derived systems simply drop the packet if the port is open. So, Nmap is looking at exactly this.

  Nmap interpretation in TCP maimon scans (see Figure 7):

Probe response	State
No response received (even after retransmissions)	open or filtered
TCP RST packet	closed
ICMP unreachable errors (type 3, code 1,2,9,10 or 13)	filtered

Table 7: Nmap interpretation for TCP maimon scans

- **TCP idle scan (-sI)**: Antirez (author of hping2) published in Bugtraq a list of ingenious port scanning techniques: a way that scans a target without sending a single packet to the target from their own IP address, using a zombie host and having IDS detect this one as an attacker.

  1. Probe the zombie's IPID and record it (every IP packet on the internet has a fragment identification number called Internet Protocol ID (IPID)).
  2. Create a SYN packet from the zombie and send it to the desired port on target. Depending on the port state, the target would increase the zombie's IPID or not.
  3. Probe the zombie's IPID again. Compare the new IPID with the one recorded in step 1.

  This will determine the port state:

  1. An increase of one indicates that the zombie has not sent out any packets, except for its reply to the attacker's probe, which means that the port is not open. This is because the target must have sent either an RST packet (which was ignored) or nothing at all.
  2. An increase of two indicates that the zombie has sent out a packet between two probes. This extra packet usually means that the port is open. This is because the target presumably sent the zombie a SYN/ACK packet in response, which introduced an RST packet from the zombie.
  3. An increase of more than two means it is not a good zombie host, as it has no predictable IPID.

  This type of scan cannot distinguish between closed and filtered ports. So, when Nmap sees an increase of one, it marks the port as closed or filtered. The Figure 19 shows and example of TCP idle scan.

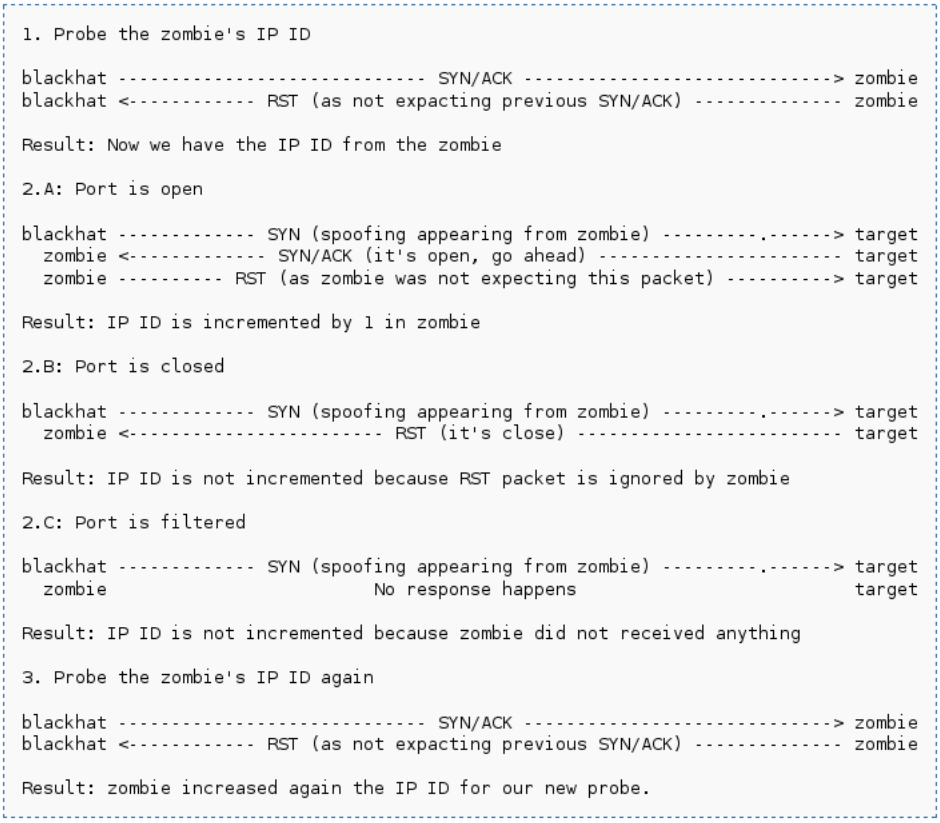


Figure 19: Example of TCP idle scan

To find a working idle scan zombie host, it is necessary to find one that assigns IPID packets incrementally on a global basis (instead of per-host communication). It should be idle (it is why is call Nmap scan idle). Printer devices could be great in acting as zombies.

It is also suggested to use -PN in order to avoid the first initial ping packet from our machine against the target. That would reveal our IP address. The -sV should never be specified as it will also show the real IP address.

It is possible to specify the port on the zombie machine using a semicolon:

```
nmap -PN -p- -sI zombie.machine.com:113 target.machine.com
```

Nmap can parallelize an idle scan by sending probes in groups of up to 100 ports and check IPID is increased N times. Later, it uses a binary search in order to find the port state.

There are some cases in which IPID is increased by 256 instead of by one. This is because some systems use little-endian when they do not convert IPID to network byte order (big-endian).

- **IP protocol scan (-sO)**: This scan allows you to determine which IP protocols (TCP, ICMP, IGMP, etc) are supported by the target machines. It sends IP packet headers and iterates through the 8 bit IP protocol field, without any data. Protocol scan is looking for ICMP protocol unreachable messages.

  Nmap interpretation in IP protocol scans (see Figure 8):

Probe response	State
Any response in any protocol from target host	open (for protocol used by response, not necessarily probe protocol)
ICMP protocol unreachable error (type 3, code 2)	closed
ICMP unreachable errors (type 3, code 1, 3, 9, 10, or 13)	filtered (through they prove ICMP is open if sent from the target machine)
No response received (even after retransmissions)	open or filtered

Table 8: Nmap interpretation for IP protocol scans

- **TCP FTP bounce scan (-b)**: FTP protocol specified by RFC 959 has a feature called proxy FTP connections that allow a user to connect to one FTP server, then ask that files be sent to a third-party server. This is exploitable as asking the FTP server to send a file to each port of a target. In turn, the error message will describe whether the port is open or not. The syntax for TCP FTP bounce scan is (B06, Fyodor G. 2009):

```
nmap -PN -b login:password@server:port.servertarget
```

Discovering hosts and performing scans against the services are tasks that Nmap perform well. However, attacks against user account services are achieved by other tools such as Hydra.

4.3.2 Hydra

Hydra is a fast network logon cracker that supports many different services. This tool has many advantages in comparison to other similar network logon and password cracker tools. It supports IPv6, HTTP proxy, and Socket Secure (SOCKS) proxy. It also supports a wide number of protocols for performing the tests—47 currently.

For Ericsson's purpose of testing part of the ECS, Hydra is convenient as it supports versions 2 and 3 of Lightweight Directory Access Protocol (LDAP) in combination with AUTH Challenge-Response Authentication (CRAM)-Message-Digest 5 (MD5) and AUTH DIGEST-MD5. It also supports MySQL, which will be the integration point of puppet modules such as Open Virtualization Format Tool (OVFT) and heat for configuration management that will perform the action of removing the hardcoded credentials in VIM VM. Additional protocols such as Secure Shell (SSH) and SSH with keys that should be tested are also supported by this tool.

Usage of this tool it is quite simple, but powerful. The following example shows how to perform a network logon attack against the localhost by SSH protocol at port 22 using *overdrive* as a username and *foobar* as a password. The -t option will use a maximum of four threads to avoid congestion and massive resources usage. This example will fail as the password is incorrect:

```
hydra -l overdrive -p 12345 ssh://localhost:22 -t 4
```

```
Hydra v8.2-dev (c) 2014 by van Hauser/THC - Please do not use in
   military or secret service organizations, or for illegal
   purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2015-03-08
   20:31:30
[WARNING] Restorefile (./hydra.restore) from a previous session
   found, to prevent overwriting, you have 10 seconds to abort...
[DATA] max 1 task per 1 server, overall 64 tasks, 1 login try
   (l:1/p:1), ~0 tries per task
[DATA] attacking service ssh on port 22
1 of 1 target completed, 0 valid passwords found
Hydra (http://www.thc.org/thc-hydra) finished at 2015-03-08
   20:31:42
```

The same example but performing a successful attack would be:

```
hydra -l overdrive -p foobar ssh://localhost:22 -t 4
Hydra v8.2-dev (c) 2014 by van Hauser/THC - Please do not use in
   military or secret service organizations, or for illegal
   purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2015-03-08
   20:37:17
[DATA] max 1 task per 1 server, overall 64 tasks, 1 login try
   (l:1/p:1), ~0 tries per task
[DATA] attacking service ssh on port 22
[22][ssh] host: localhost login: overdrive password: foobar
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2015-03-08
   20:37:17
```

The tool also allows passing a list of logins and passwords from a file by using the options -L for logins and -P for passwords. A full set of options would be described if no parameter is passed to the binary:

```
hydra
Hydra v8.2-dev (c) 2014 by van Hauser/THC - Please do not use in
   military or secret service organizations, or for illegal
   purposes.

Syntax: hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C
   FILE]] [-e nsr] [-o FILE] [-t TASKS] [-M FILE [-T TASKS]] [-w
   TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET]
   [-SuvVd46] [service://server[:PORT][/OPT]]

Options:
  -l LOGIN or -L FILE login with LOGIN name, or load several
     logins from FILE
  -p PASS or -P FILE try password PASS, or load several passwords
     from FILE
  -C FILE colon separated "login:pass" format, instead of -L/-P
     options
```

```
  -M FILE list of servers to attack, one entry per line, ':' to
     specify port
  -t TASKS run TASKS number of connects in parallel (per host,
     default: 16)
  -U      service module usage details
  -h      more command line options (COMPLETE HELP)
  server  the target: DNS, IP or 192.168.0.0/24 (this OR the -M
     option)
  service the service to crack (see below for supported protocols)
  OPT     some service modules support additional input (-U for
     module help)

Supported services: asterisk cisco cisco-enable cvs ftp ftps
    http[s]-{head|get} http[s]-{get|post}-form http-proxy
    http-proxy-urlenum icq imap[s] irc ldap2[s]
    ldap3[-{cram|digest}md5][s] mssql mysql(v4) nntp
    oracle-listener oracle-sid pcanywhere pcnfs pop3[s] rdp redis
    rexec rlogin rsh s7-300 sip smb smtp[s] smtp-enum snmp socks5
    ssh sshkey teamspeak telnet[s] vmauthd vnc xmpp

Hydra is a tool to guess/crack valid login/password pairs.
    Licensed under AGPL
v3.0. The newest version is always available at
    http://www.thc.org/thc-hydra
Don't use in military or secret service organizations, or for
    illegal purposes.

Example: hydra -l user -P passlist.txt ftp://192.168.0.1
```

Hydra works really well for user account service attacks. Moreover, this is not enough to verify the services are secured. Verification of the services by using advanced buffers overflows, wrong configurations or DoS attacks is fulfill by Nessus.

4.3.3 Nessus

Nessus is a vulnerability scanner developed by Tenable Network Security. Nessus allows scans for the following types of vulnerabilities: vulnerabilities that allow a remote hacker to control or access sensitive data on a system, misconfigurations, default or common non-strength passwords, black or absent passwords for accounts, DoS attacks against TCP/IP suite, and Payment Card Industry Data Security Standard (PCI DSS) audits.

There are different Nessus products from Tenable.

- Nessus®
- Nessus Enterprise
- Nessus Enterprise Cloud
- Nessus Auditor Bundles
- Nessus Home

The Security Center is a comprehensive vulnerability, threat, and compliance management platform that alleviates the arduous and time-consuming tasks of security forensic analysis, complex threat mitigation, and compliance management. It discovers assets in the environment by monitoring them. Security Center combines vulnerability scanning, network sniffing, and event log correction into a single platform.

The Nessus product platform provides the Passive Vulnerability Scanner (PVS), which identifies hosts and services as they connect to the network and monitors the network continuously for vulnerabilities.

Log Correlation Engine (LCE) is a component of the Security Center that provides incident response and forensics functionality with SIEM technology that aggregates, normalizes, correlates, and analyses event log data from raw network traffic, IDS, system, application logs, and user activity. Figure 20 shows the different entities in the Nessus product platform.

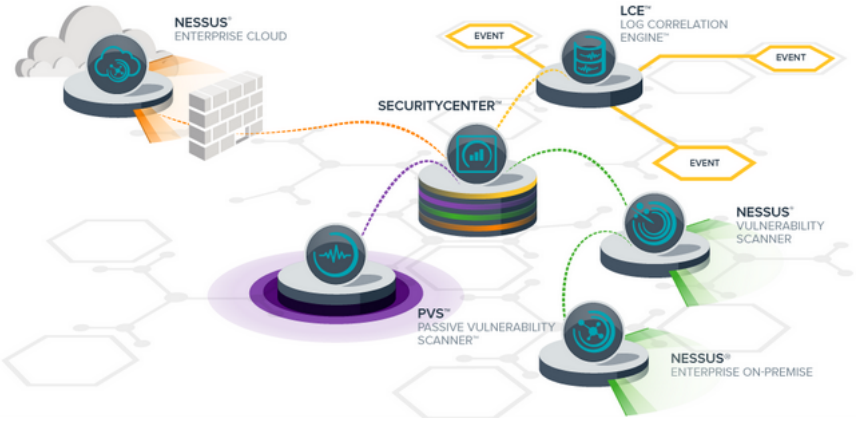


Figure 20: Nessus product platform, (Nessus installation, tenable.com)

Currently, Nessus Enterprise is the licensed software purchased by Ericsson, even if the purpose would be more suitable for the Nessus Enterprise Cloud product. However, each of these component groups provides advanced malware protection, mobile security and compliance, patch auditing, configuration auditing, critical infrastructure protection, and cloud security and compliance. Figure 21 illustrates the different components and software available in Nessus.

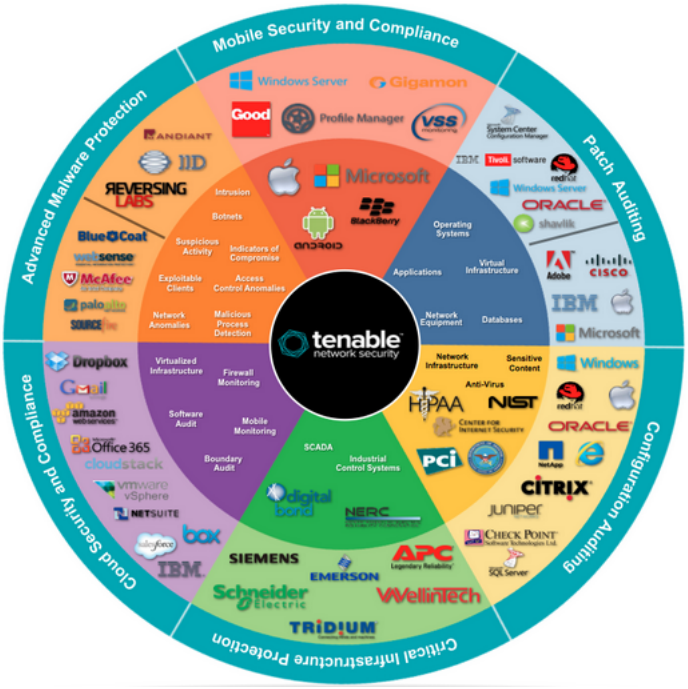


Figure 21: Nessus components in an integrated platform, (Nessus installation, tenable.com)

Nessus uses unique sensors, whereby it can run active scans or passive network and log sensors gathering system data, enabling real-time analytics. The platform incorporates an advanced analytics engine that integrates sensor data with threat intelligence with Security Information and Event Management (SIEM). Figure 22 illustrates the Nessus unique underlying architecture.

Figure 22: Nessus unique underlying architecture, (Nessus Security Center Architecture, tenable.com)

In further detail, Nessus unified security monitors using internal databases for rules, policies, targets, and configurations for different components, and some Graphical User Interface (GUI) are provided to interact with those components. Internally, Nessus communicates by using encrypted data over XML-Remote Procedure Call (XML-RPC). Figure 23 illustrates the Nessus unified security-monitoring architecture.

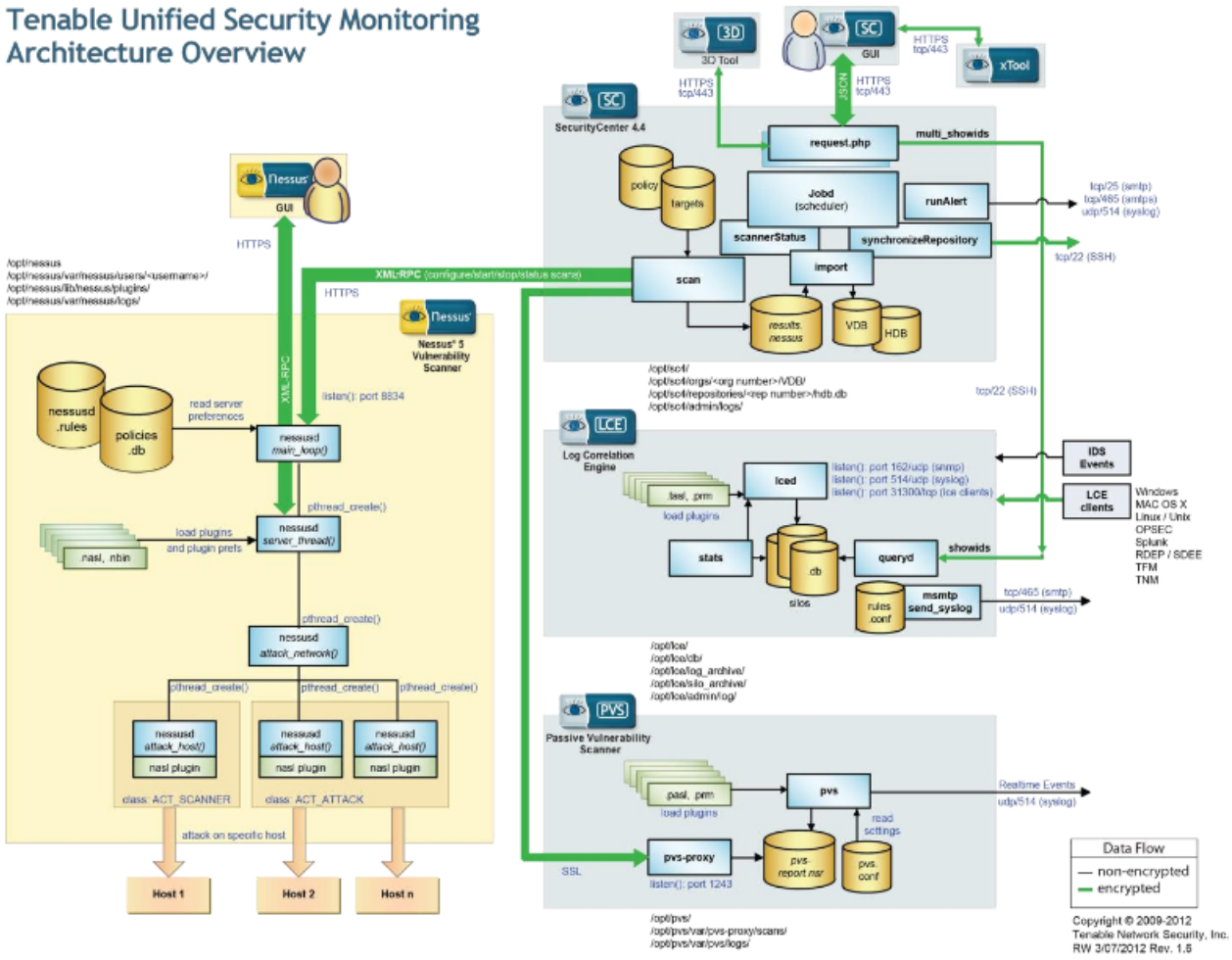


Figure 23: Nessus unified security-monitoring architecture, (Nessus Security Center Architecture, tenable.com)

Nessus basically operates with two types of users after installation: administrator and normal users. Administrators can create users with different capabilities and users just operate the scans.

In Nessus, it is possible to create policies that consist of configuration options related to performing a vulnerability scan. With these policies, it is possible to specify parameters that control technical aspects such as timeouts, number of hosts, type of port scanning, and so on. It could also be possible to specify credentials for local scans such as Windows or Secure Shell (SSH) passwords or keys, and credentials to authenticate databases, HTTP, FTP, POP, IMAP, or Kerberos authentication. Policies also define the granularity and plugin-based scan specifications, policy checks for databases, report verbosity, service detection, UNIX compliance checks, among other things (W03. Several authors. 2015).

In Nessus, there are different kinds of policies:

- **Host discovery**: This identifies live hosts and open ports.
- **Basic network scan**: It is used for scanning internal and external hosts.
- **Credentialed patch audit**: It logs in to the systems and lists missing software updates.

- **Windows malware scan**: It searches for malware on Windows systems.
- **Heartbleed detection**: It remote checks for the biggest Open-source Secure Sockets Layer (OpenSSL) vulnerability.
- **Web application tests**: It performs generic web application scans.
- **Mobile device scan**: It launches scans against Apple Profile Manager, Active Directory Service Interfaces (ADSI), MobileIron, and Mobile Device Management (MDM).
- **Offline config auditing**: It uploads and audits the configuration file of a network device.
- **Prepare for PCI DSS audits**: It prepares a PCI DSS compliance audit.
- **Advanced policy**: This is for users who want total control for policy configurations.

Figure 24 shows different wizards available in Nessus.

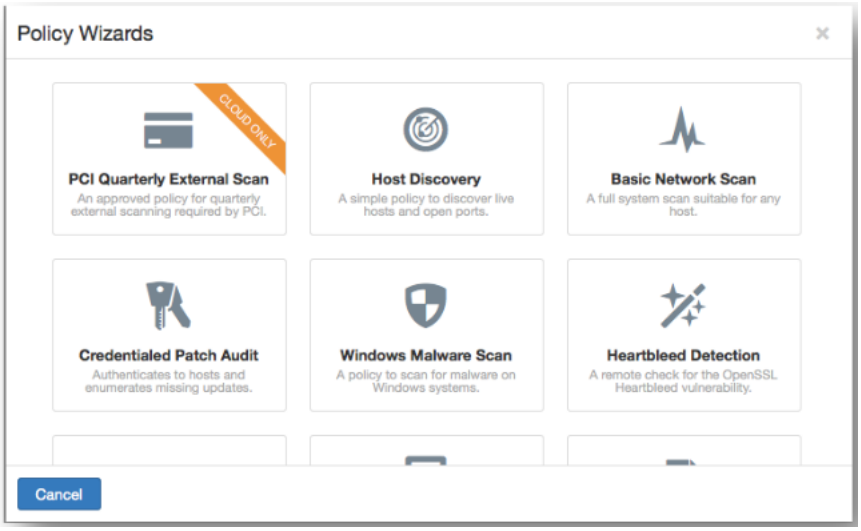


Figure 24: Nessus policy wizards, (Nessus installation and configuration guide, tenable.com)

From the advanced policy, it is possible to specify many details. However, one detail that needs to be highlighted is the plugins option. Nessus allows you to enable or disable a full set of families or specific plugins. Currently, there are more than 60,000 plugins covering local and remote flaws. Figure 25 shows a more advance view for Nessus policies.

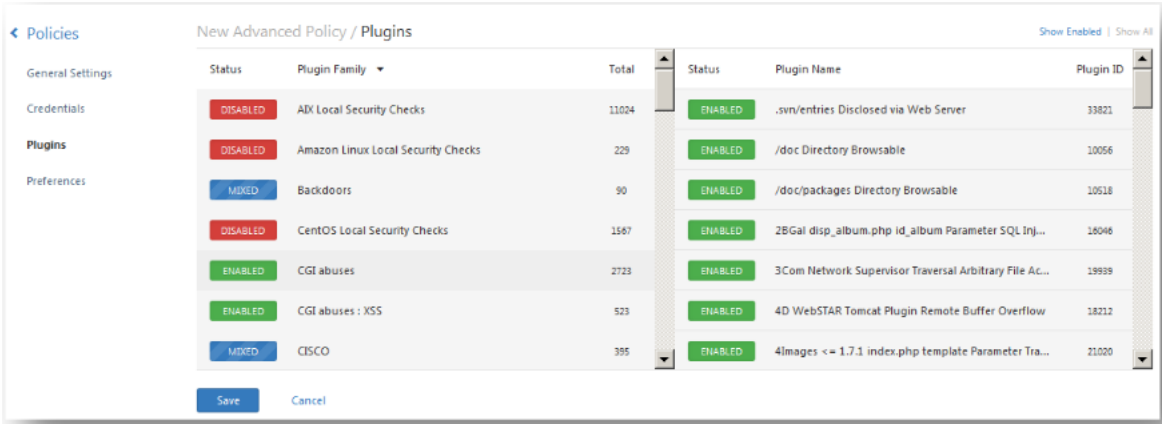


Figure 25: Nessus advanced policies: plugins, (Nessus installation and configuration guide, tenable.com)

With Nessus, it is also possible to automate scans, scheduling them at a certain time. These could be launched immediately, on demand (running in the future as a 'scan template'), once at a specific time, daily, weekly, monthly, or yearly. Figure 26 shows how Nessus scans are scheduled.

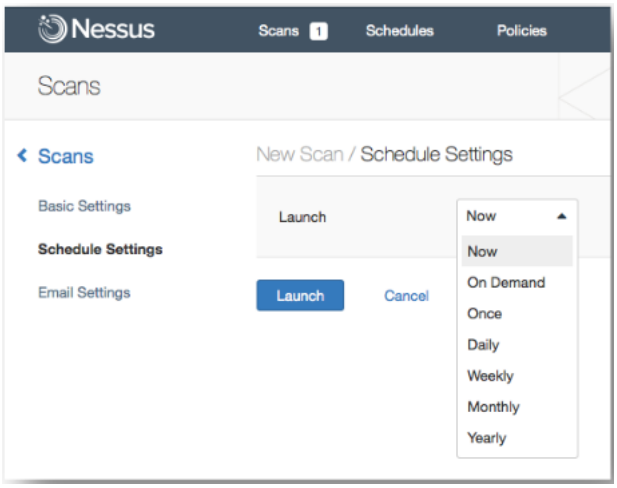


Figure 26: Nessus automating scans, (Nessus installation and configuration guide, tenable.com)

After the scan is performed, when browsing the scan results, it is possible to observe critical, high, medium, or low vulnerabilities, as well as some additional information per scanned host. From each vulnerability, it is possible to expand the report to some detailed information, which will provide not only a description, but an output of an example of the vulnerability as a proof and commands executed, along with the suggestion or recommendation to fix the vulnerability in the system. Figure 26 gives an example of a Nessus scan report.

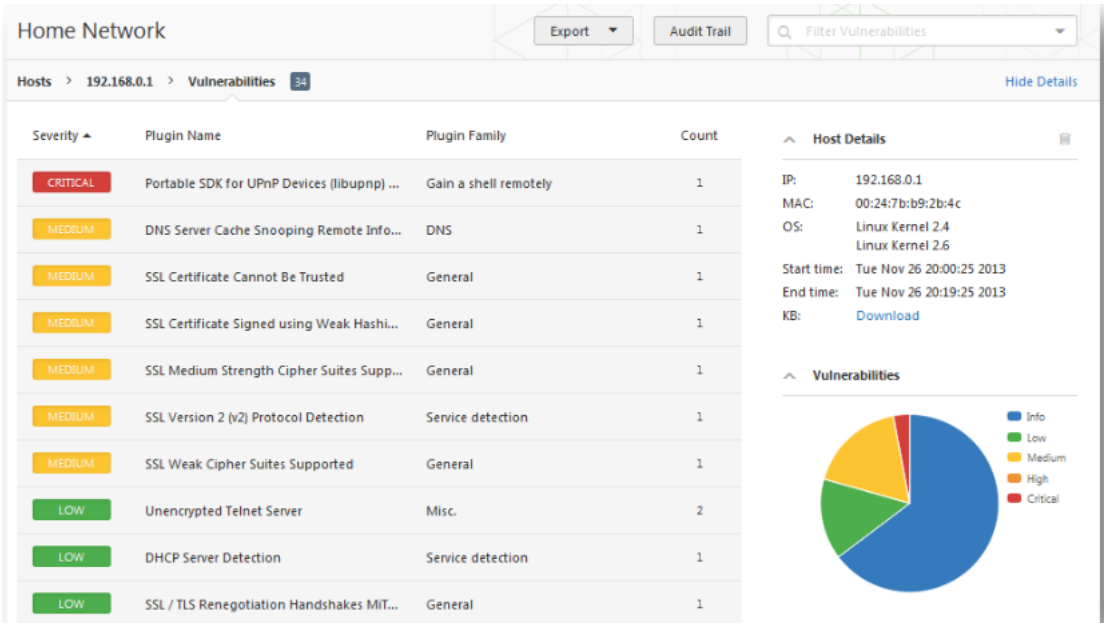


Figure 27: Nessus report example, (Nessus installation and configuration guide, tenable.com)

All the reports from Nessus are saved in an encrypted database, as they would contain confidential information, vulnerabilities about the systems, and credentials used to access hosts (W04. Several authors. 2015).

Load traffic generators have been used for checking the robustness of the solution. IXIA is a tool which accomplish this.

4.3.4 IXIA

IXIA provides products for testing traffic across nodes. IXIA products can be divided into different components:

- IxNetwork: This provides wire-rate traffic generation with service modelling that builds realistic, dynamically controllable data-plane traffic. It supports several protocols for internet, video, voice, storage, wireless, infrastructure, and encapsulation, as well as security. IxNetwork also provides traffic for published vulnerabilities, malware, and high-performance Distributed Denial of Service (DDoS)
- IxLoad: This emulates data, voice, and video subscribers and their associated protocols for ultra-high-performance testing.
- IxVM: This provides a software-based version of a traditional Ixia hardware port that can be deployed in a virtualized environment.
- IxServer: This will control and communicate the Ixia chassis or IxVM through IXOS.
- IxClient: This provides a low- and a high-level API to communicate with IxServer.
- IxExplore: This is a standard way to communicate with IxServer.
- IxAutomate: This provides automation for commanding Ixia with IxServer.
- IXOS: This refers to the OS running on Ixia chassis or IxVM.

Figure 28 clarifies how IxNetwork places IxVM in different networks.

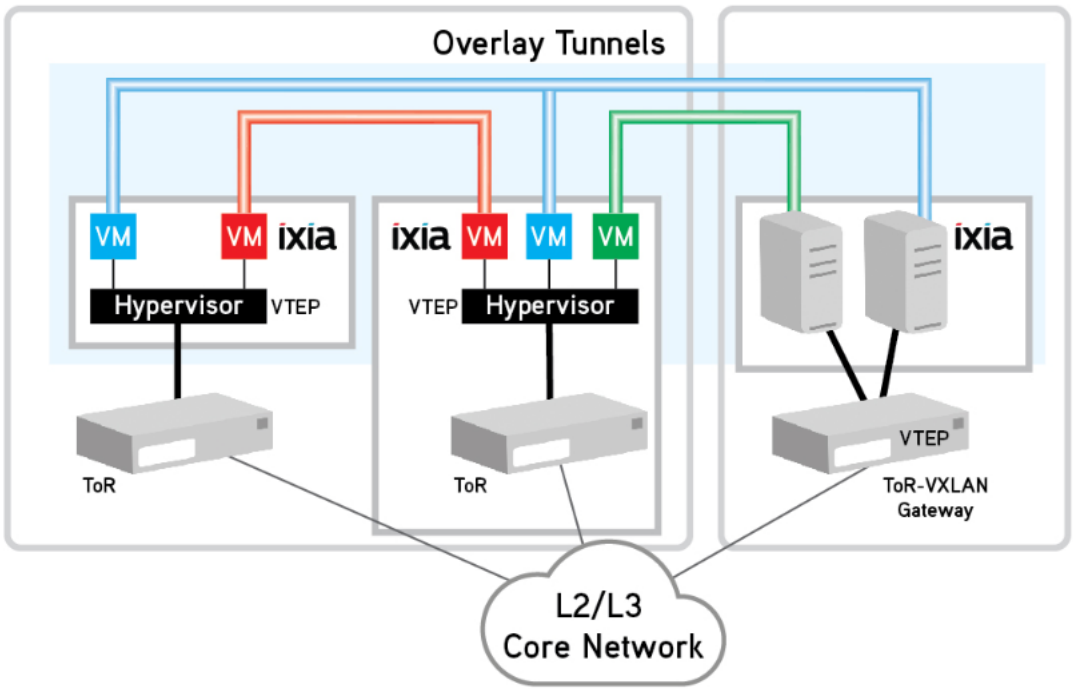


Figure 28: IxNetwork using IxVM placing them in different networks, (Validation virtualized asset and environment, ixia.com)

Additional components from Ixia are useful from the security perspective. The main one is IxLoad-Attack, which addresses the testing of security devices with known vulnerabilities, DDoS attacks, and IPsec encryption for Virtual Private Network (VPN) gateways. Authentication mechanisms, including 802.1x and Network Access Control (NAC) (W05. Several authors. 2014). Figure 29 gives an example of IxNetwork component in IXIA.

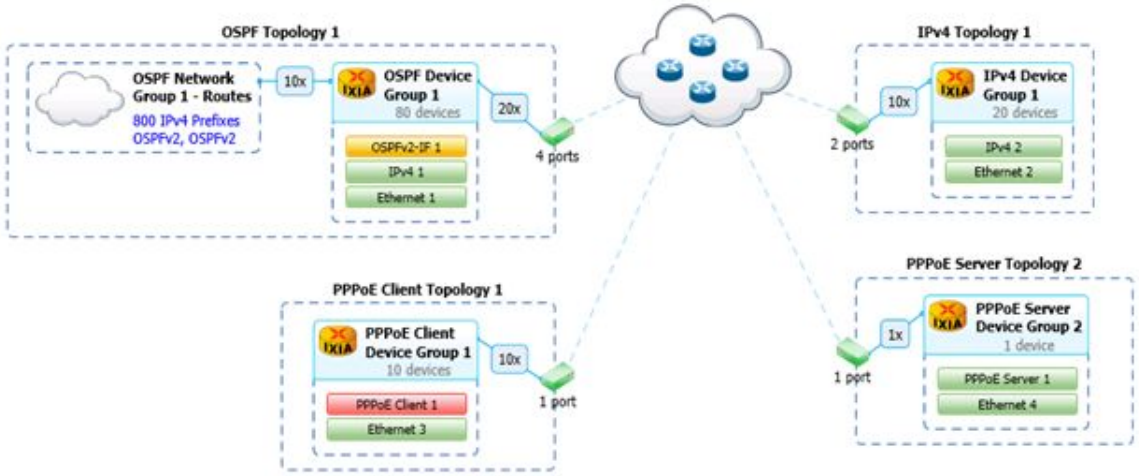


Figure 29: Example of IxNetwork component in IXIA, (IX Network VXLAN emulation, ixia.com)

With Ixia, it is possible to test the performance of the ECS product, but also a bunch of vulnerabilities across the nodes. With IxVM, it is possible to place the testing source in different interfaces, nodes, and components (W06. Several authors. 2014).

Traffic analyzers and advanced packet manipulation has been performed to check some the corner cases. Scapy and Hping tools will cover this area.

4.3.5 Scapy and Hping

Scapy is replacing hping as a command line tool that performs advanced packet manipulation. It can also handle tasks such as scanning, tracerouting, probing, unit tests, attacks, and network discovery. This tool can be used for testing the robustness of the system by manipulating frames, packets, and datagrams for a wide range of protocols. Most of the important protocols supported are: ARP, DHCP, DNS, IEEE 802.1Q, ethernet, ICMP, IP, ISAKMP, LLC, NTP, PPP, NetBIOS, RIP, RADIUS, SNAP, STP, TCP, UDP, or raw sockets (W07. Biondi P. Scapy community. 2014). An IP packet manipulation using scapy is shown in 30.

```
>>> IP()
<IP |>
>>> a=IP(dst="172.16.1.40")
>>> a
<IP dst=172.16.1.40 |>
>>> a.dst
'172.16.1.40'
>>> a.ttl
64
```

Figure 30: IP manipulation packet in scapy, (Scapy documentation, secdev.org)

With scapy, Figure 31 shows the capability to examine and manipulate any fields from the different layers, and then send the command through some specific interface.

```
>>> str(IP())
'E\x00\x00\x14\x00\x01\x00\x00@\x00|\xe7\x7f\x00\x00\x01\x7f\x00\x00\x01'
>>> IP(_)
<IP version=4L ihl=5L tos=0x0 len=20 id=1 flags= frag=0L ttl=64 proto=IP
 chksum=0x7ce7 src=127.0.0.1 dst=127.0.0.1 |>
>>>  a=Ether()/IP(dst="www.slashdot.org")/TCP()/"GET /index.html HTTP/1.0 \n\n"
>>>  hexdump(a)
00 02 15 37 A2 44 00 AE F3 52 AA D1 08 00 45 00  ...7.D...R....E.
00 43 00 01 00 00 40 06 78 3C C0 A8 05 15 42 23  .C....@.x<....B#
FA 97 00 14 00 50 00 00 00 00 00 00 00 00 50 02  .....P........P.
20 00 BB 39 00 00 47 45 54 20 2F 69 6E 64 65 78   ..9..GET /index
2E 68 74 6D 6C 20 48 54 54 50 2F 31 2E 30 20 0A  .html HTTP/1.0 .
0A                                               .
>>> b=str(a)
>>> b
'\x00\x02\x157\xa2D\x00\xae\xf3R\xaa\xd1\x08\x00E\x00\x00C\x00\x01\x00
\x00@\x06x<\xc0
 \xa8\x05\x15B#\xfa\x97\x00\x14\x00P\x00\x00\x00\x00\x00\x00\x00\x00P\x02 \x00
 \xbb9\x00\x00GET /index.html HTTP/1.0 \n\n'
>>> c=Ether(b)
>>> c
<Ether dst=00:02:15:37:a2:44 src=00:ae:f3:52:aa:d1 type=0x800 |<IP version=4L
 ihl=5L tos=0x0 len=67 id=1 flags= frag=0L ttl=64 proto=TCP chksum=0x783c
 src=192.168.5.21 dst=66.35.250.151  |<TCP sport=20 dport=80 seq=0L
 ack=0L dataofs=5L reserved=0L flags=S window=8192 chksum=0xbb39 urgptr=0
 options=[] |<Raw load='GET /index.html HTTP/1.0 \n\n' |>>>>
```

Figure 31: IP packet and ethernet frame manipulation and dump, (Scapy documentation, secdev.org)

Under scapy, it is possible to capture packets in some specific interface and load later on a pcap (packet capture) file as shown in 32.

```
>>> a=rdpcap("/spare/captures/isakmp.cap")
>>> a
<isakmp.cap: UDP:721 TCP:0 ICMP:0 Other:0>
>>> a[423].pdfdump(layer_shift=1)
>>> a[423].psdump("/tmp/isakmp_pkt.eps",layer_shift=1)
```

Figure 32: Load-captured file and graphical dump, (Scapy documentation, secdev.org)

Figure 33 shows scapy capability of graphical dumps, either in Post Script (PS) or Portable Document Format (PDF) extensions.

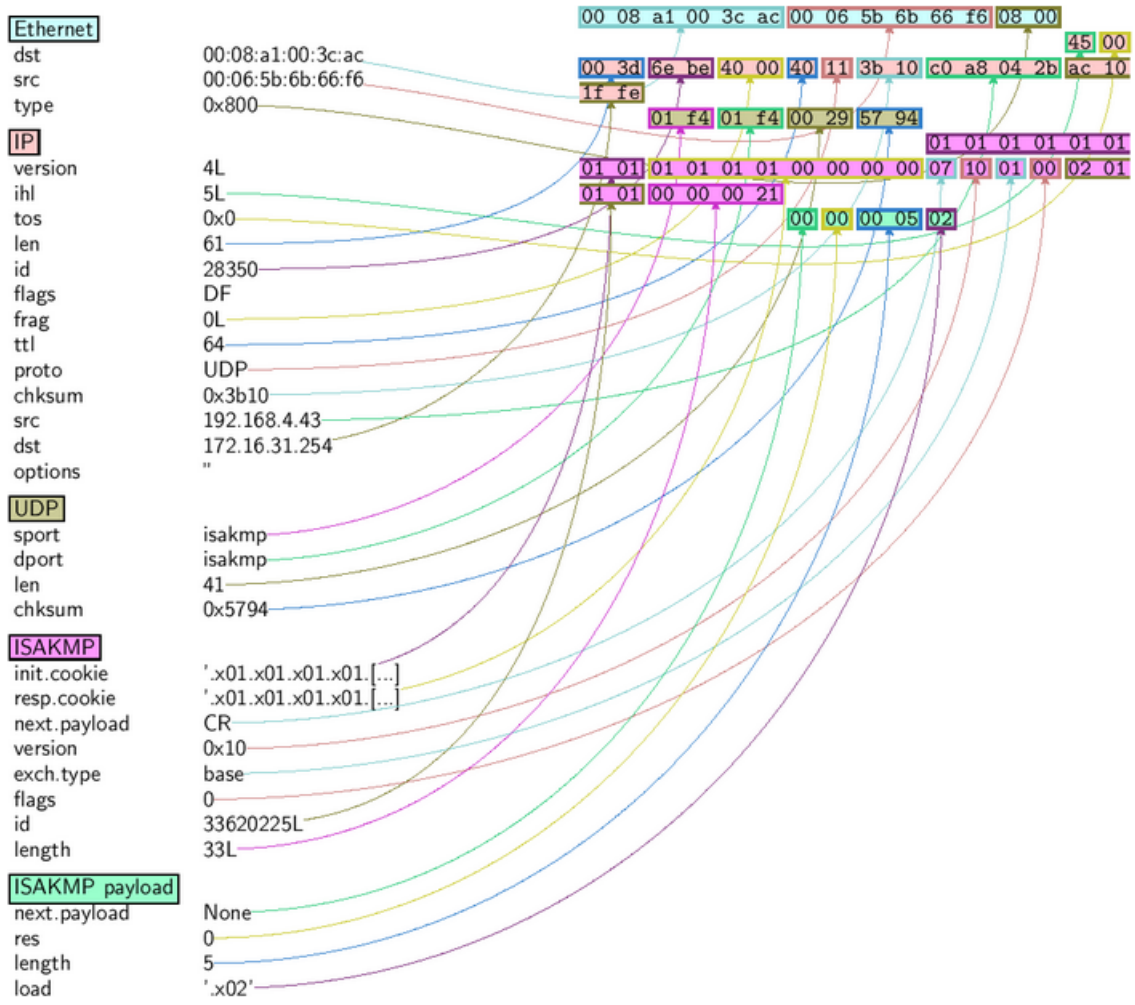


Figure 33: Graphical representation of a packet, (Scapy documentation, secdev.org)

Next section describes some tools used for visualizing the traffic captured from the traffic generated by IXIA, Scapy and Hping.

4.3.6 Tcpdump and Wireshark

Tcpdump and wireshark are other applications useful for capturing packets. Tcpdump is a command-like tool. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached (B07. Chappell L. Combs G. 2012). See Figure 34 for an example of captured traffic used with tcpdump.

```
13:08:05.737768 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1221 > dsl-usw-cust-110.inetarena.com.www: . 342:342(0) ack 1449 win 31856 <nop
,nop,timestamp 1247771 114849487> (DF)
13:08:07.467571 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: . 1449:2897(1448) ack 342 win 31856
<nop,nop,timestamp 114849637 1247771> (DF)
13:08:07.707634 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: . 2897:4345(1448) ack 342 win 31856
<nop,nop,timestamp 114849637 1247771> (DF)
13:08:07.707922 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1221 > dsl-usw-cust-110.inetarena.com.www: . 342:342(0) ack 4345 win 31856 <nop
,nop,timestamp 1247968 114849637> (DF)
13:08:08.057841 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1045 > ns.de.ibm.net.domain: 8928+ PTR? 110.107.102.209.in-addr.arpa. (46)
13:08:08.747598 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: P 4345:5793(1448) ack 342 win 31856
<nop,nop,timestamp 114849813 1247968> (DF)
13:08:08.847870 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: FP 5793:6297(504) ack 342 win 31856
<nop,nop,timestamp 114849813 1247968> (DF)
13:08:08.848063 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1221 > dsl-usw-cust-110.inetarena.com.www: . 342:342(0) ack 6298 win 31856 <nop
,nop,timestamp 1248082 114849813> (DF)
13:08:08.907566 ppp0 < ns.de.ibm.net.domain > slip139-92-26-177.ist.tr.ibm.net.1045: 8928* 3/1/1 PTR dsl-usw-cust-110.inetarena.com., P
TR fingerless.or (199)
13:08:09.151742 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1221 > dsl-usw-cust-110.inetarena.com.www: F 342:342(0) ack 6298 win 31856 <nop
,nop,timestamp 1248112 114849813> (DF)
13:08:10.137603 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: . 6298:6298(0) ack 343 win 31856 <no
p,nop,timestamp 114849967 1248112> (DF)
13:09:01.984210 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: S 920197285:920197285(0) win 32120 <
mss 1460,sackOK,timestamp 1253395 0,nop,wscale 0> (DF)
13:09:03.097569 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: S 1222277738:1222277738(0) ack 92019
7286 win 32120 <mss 1460,sackOK,timestamp 114855252 1253395,nop,wscale 0> (DF)
13:09:03.098197 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: . 1:1(0) ack 1 win 32120 <nop,nop,ti
mestamp 1253507 114855252> (DF)
13:09:03.102171 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: P 1:322(321) ack 1 win 32120 <nop,no
p,timestamp 1253507 114855252> (DF)
13:09:04.147613 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: . 1:1(0) ack 322 win 31856 <nop,nop,
timestamp 114855369 1253507> (DF)
13:09:04.507608 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: . 1:1449(1448) ack 322 win 31856 <no
p,nop,timestamp 114855369 1253507> (DF)
13:09:04.507934 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: . 322:322(0) ack 1449 win 31856 <nop
,nop,timestamp 1253648 114855369> (DF)
13:09:05.627604 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: . 1449:2897(1448) ack 322 win 31856
<nop,nop,timestamp 114855491 1253648> (DF)
13:09:05.857649 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: . 2897:4345(1448) ack 322 win 31856
<nop,nop,timestamp 114855491 1253648> (DF)
13:09:05.857918 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: . 322:322(0) ack 4345 win 31856 <nop
,nop,timestamp 1253783 114855491> (DF)
13:09:06.907557 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: FP 4345:5792(1447) ack 322 win 31856
 <nop,nop,timestamp 114855627 1253783> (DF)
13:09:06.907887 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: . 322:322(0) ack 5793 win 31856 <nop
,nop,timestamp 1253888 114855627> (DF)
13:09:07.401205 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1222 > dsl-usw-cust-110.inetarena.com.www: F 322:322(0) ack 5793 win 31856 <nop
,nop,timestamp 1253937 114855627> (DF)
13:09:08.317623 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1222: . 5793:5793(0) ack 323 win 31856 <no
p,nop,timestamp 114855780 1253937> (DF)
```

Figure 34: Captured traffic with tcpdump, (Tcpdump documentation, tcpdump.org)

Wireshark is similar to tcpdump, but it provides a graphical front-end, filters, and views that will help as a packet analyser tool as shown in Figure 35.

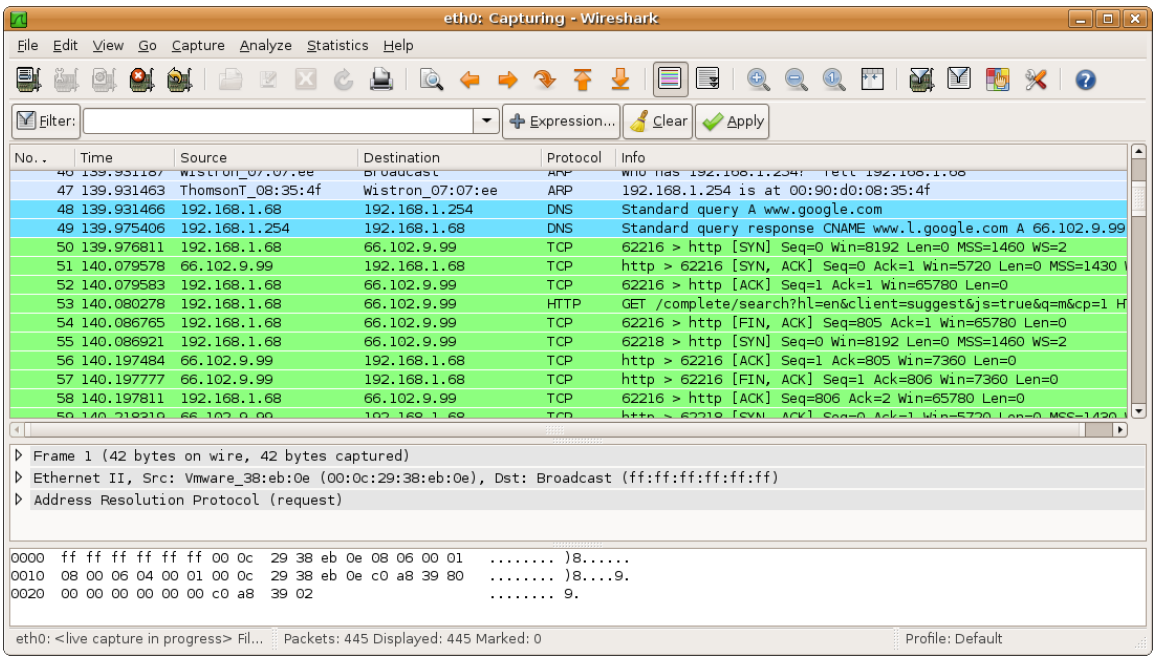


Figure 35: Wireshark representation data of captured traffic, (Wireshark documentation, wireshark.org)

As seen in the above picture, Wireshark represents every packet by an identifier. Wireshark includes relevant information such as timestamp, source and destination

addresses, protocol, information and visualizes the packet encapsulation and decodes the frame.

Other tools have been used to complement additional needs executed in the verified nodes. Next section describes these tools.

4.3.7 Binutils, Coreutils, GNU, and Other Utils

In GNU/Linux systems, several tools that come with the system are extremely powerful. The UNIX (and clones) philosophy says to make each program do one thing well, and that the powerfulness comes from within their relationship.

There are different sets of utils provided in big packages: binutils, coreutils, and GNU utils mainly. The binutils are a set of programming tools for creating and managing binary programs, object files, libraries, profile data, and assembly source code. The coreutils are a package of GNU software containing many of the basic tools, such as cat, ls, and rm, needed for Unix-like operating systems. It is a combination of a number of earlier packages, including textutils, shellutils, and fileutils, along with some other miscellaneous utilities. The GNU utils is a toolchain that contains several components such as GNU make, GNU compiler collection, GNU binutils, GNU bison, GNU m4, GNU debugger, and GNU build system (W08. Several authors. 2014; W09. Several authors. 2014).

Apart from these sets of utils, there are some other utils that would be useful to verify the status of the system, as well as the security:

- **netstat**: This shows the network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.
- **socklist** or **sockstat**: These display the list of open sockets.
- **lsof**: This lists open files.
- **nc**: This reads and writes data across the network connections using TCP or UDP protocol.
- **curl**: This transfers data from or to a server.
- **iptraf**: This is the IP network monitoring software.
- **iperf**: This is the TCP/UDP bandwidth measurement tool.
- **iptables**: This is an administration tool for IPv4 packet filtering and Network Address Translation (NAT).
- **strace**: This traces system calls and signals.

Figure 36 shows the active connections:

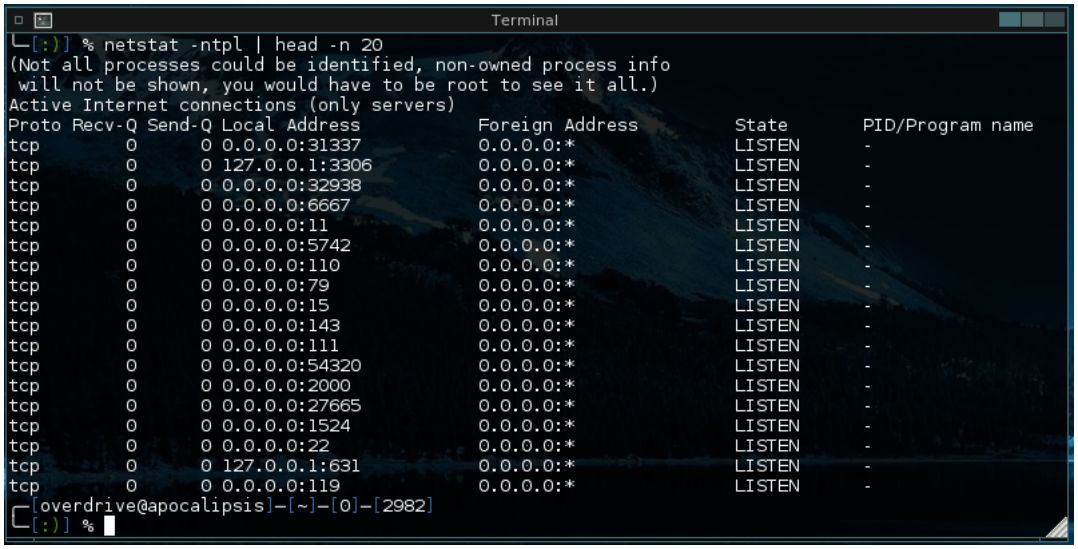


Figure 36: Listening to open connections with netstat

Figure 37 illustrates the list current open files:

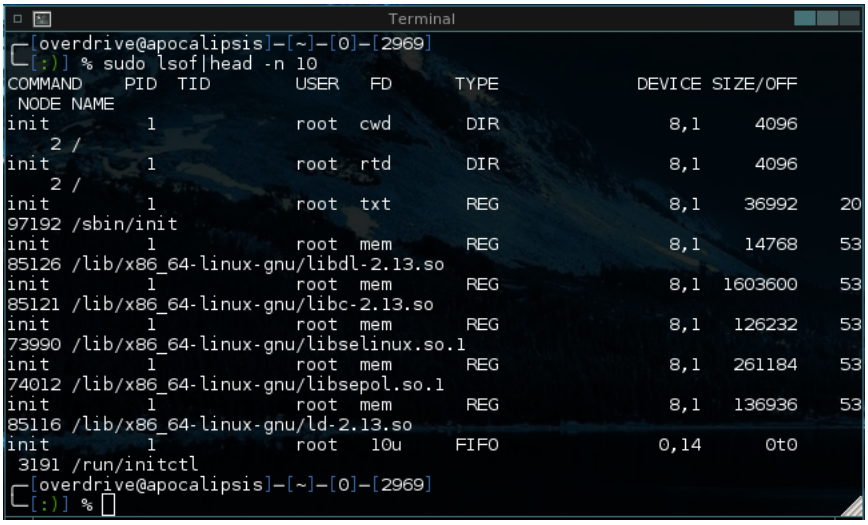


Figure 37: List open files with lsof

Firewall rules are shown in Figure 38:

```
[root@localhost sysconfig]# iptables-restore < /etc/sysconfig/iptables.orig
[root@localhost sysconfig]# service iptables status
Table: filter
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    RH-Firewall-1-INPUT  all  --  0.0.0.0/0           0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination
1    RH-Firewall-1-INPUT  all  --  0.0.0.0/0           0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain RH-Firewall-1-INPUT (2 references)
num  target     prot opt source               destination
1    ACCEPT     all  --  0.0.0.0/0           0.0.0.0/0
2    ACCEPT     icmp --  0.0.0.0/0           0.0.0.0/0           icmp type 255
3    ACCEPT     esp  --  0.0.0.0/0           0.0.0.0/0
4    ACCEPT     ah   --  0.0.0.0/0           0.0.0.0/0
5    ACCEPT     udp  --  0.0.0.0/0           224.0.0.251         udp dpt:5353
6    ACCEPT     udp  --  0.0.0.0/0           0.0.0.0/0           udp dpt:631
7    ACCEPT     tcp  --  0.0.0.0/0           0.0.0.0/0           tcp dpt:631
8    ACCEPT     all  --  0.0.0.0/0           0.0.0.0/0           state RELATED,ESTABLISHED
9    ACCEPT     tcp  --  0.0.0.0/0           0.0.0.0/0           state NEW tcp dpt:22
10   REJECT     all  --  0.0.0.0/0           0.0.0.0/0           reject-with icmp-host-prohibited
```

Figure 38: Iptables firewall rules

System calls can be traced with strace as illustrates Figure 39:

```
raghu@raghu-Linoxide ~ $ strace ls
execve("/bin/ls", ["ls"], [/* 39 vars */]) = 0
brk(0)                                  = 0x989c000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb77c8000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=86538, ...}) = 0
mmap2(NULL, 86538, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb77b2000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\340B\0\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=120668, ...}) = 0
mmap2(NULL, 125884, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7793000
mmap2(0xb77b0000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c) = 0xb77b0000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\31\0\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=30696, ...}) = 0
mmap2(NULL, 33352, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb778a000
```

Figure 39: System call trace with strace

All the above binaries and tools used for testing have been included in the tests cases. Test cases have been implemented within a specific test framework.

4.4 Test Frameworks

Several test frameworks can be chosen to use the Testing Tools (section 4.3). After long discussions within the organization, the cloud security team decided to use the Robot framework.

There were several reasons behind the decision; the main decision is that Robot framework is Domain-Specific Language (DSL), with the benefit of auto documentation when writing test cases. Another reason for choosing Robot is that testers and developers do not need to learn another programming language or deal with debugging in specific programming languages such as Java if tests are failing; instead, the DSL focuses completely on the task and domain of the intended test, and not in the paradigm of the programming languages for general purposes. The last benefit from this framework is that as it is free software (Apache License), it has an extended built-in and full set of external libraries that will cover almost all the requirements as a testing tool, but it will also be possible, if needed, to extend these existing libraries or add new ones by using Python programming language.

The Robot framework will be used not only for End-to-End (E2E) testing, but also for component test, acceptance testing, and Acceptance Test-Driven Development (ATDD). This will allow developers, testers, PO, managers, and CI members to understand the test cases (W10. Several authors. 2014).

4.4.1 Robot Framework Installation and Configuration

The Robot framework is the test framework chosen for security testing. There are several advantages and not many drawbacks.

For installing the Robot framework, a pip command line tool is used:

```
# pip install robotframework
```

Before each test is run, the Robot framework will be upgraded, also using the pip command line tool:

```
# pip install --upgrade robotframework
```

To verify the installation, the following command is required:

```
# pybot --version
Robot Framework 2.8.5 (Python 2.7.3 on apocalipsis)
```

The standard library provides functions to test the most common scenarios:

- **Builtin**: This provides a set of frequently needed generic keywords. It is always automatically available without imports.
- **Dialogs**: The dialogs provide means for pausing the test execution and getting input from users.
- **Collections**: This provides a set of keywords for handling Python lists and dictionaries.
- **OperatingSystem**: This enables various operating system-related tasks to be performed in the system where the Robot framework is running.
- **Remote**: This is a special library acting as a proxy between the Robot framework and test libraries elsewhere.
- **Screenshot**: This provides keywords to capture screenshots of the desktop.

- **String**: This is used for generating, modifying, and verifying strings.
- **Telnet**: This makes it possible to connect to Telnet servers and execute commands on the opened connections.
- **XML**: This is used for generating, modifying, and verifying XML files.
- **Process**: This is used for running processes in the system.
- **DateTime**: This is used for date and time conversions.

Most of the required functions are included in the built-in libraries. The use of extensions which provide additional libraries and functions is required in some cases.

4.4.2 Extensions

There are different extensions that can be added to the standard functionality to test more specific, but still important, scenarios.

- **Archive**: Archive library Library for handling zip- and tar-archives.
- **Database**: for database testing.
- **Diff**: diff to files together.
- **robotframework-faker**: for Faker, a fake test data generator.
- **FTP**: for testing and using the FTP server with the Robot framework.
- **HTTP livetest**: for HTTP-level testing using the livetest tool internally.
- **HTTP requests**: for HTTP-level testing using Request internally.
- **MongoDB**: for interacting with MongoDB from the Robot framework using pymongo.
- **Rammbock Generic network protocol test**: an easy way to specify network packets and inspect the results of sent and received packets.
- **SeleniumLibrary**: web testing library that uses popular Selenium tool internally.
- **Selenium2Library**: web testing library that uses Selenium 2. For most parts, drop-in-replacement for old SeleniumLibrary.
- **SSHLibrary**: enables executing commands on remote machines over an SSH connection. Also supports transferring files using SFTP.

There are additional extensions for the Robot framework; however, those are not relevant for the goal of this project.

For installing any extension, it is also possible use a pip command line tool. In this example, the SSHLibrary is installed:

```
# pip install robotframework-sshlibrary
```

Set up the test framework, some guidelines have been created to write test cases in an standard manner.

4.4.3 Guidelines to Write Test Cases

Before writing test cases for ECS, some rules and good practices have been described:

1. In order to write test cases, different sections need to be created. These include 'Settings' and 'Test Cases'.
2. Hardcoded values must be avoided inside the test cases and should be moved to configuration.robot, where variables are allocated.
3. Avoid TABs (they are evil) and use spaces to separate keywords and parameters. Four spaces are fine, but they can also be aligned with previous lines.
4. Semantics is important. Test cases must be readable in a high-level domain.
5. Specific parameters for commands should not be included inside the test cases. Please use keywords instead.

Figure 40 gives an example of a test case using the Nmap extension:

```
*** Settings ***
Library     NmapLibrary
Resource    configuration.robot

*** Test Cases ***
Verify Hypervisors and CICs are UP and running
    Ping Scan          ${IP_RANGE}
    Host Is Up         ${HYPERVISOR1}
    Host Is Up         ${HYPERVISOR2}
    Host Is Up         ${HYPERVISOR3}
    Host Is Up         ${CIC1}
    Host Is Up         ${CIC2}
    Host Is Up         ${CIC3}
```

Figure 40: Example of basic test case using Nmap extension

Next section describes how to run test cases with Robot.

4.4.4 Running Test Cases with Robot

Normally, a test would be run with the pybot command line tool:

```
pybot file.robot
```

This will give an output with the current execution and a report in different formats (see Figure 41):

```
==============================================================================
Tests                                                                 | FAIL |
185 critical tests, 176 passed, 9 failed
185 tests total, 176 passed, 9 failed
==============================================================================
Output:  <path>/ecs/layers/meta-sec/output.xml
Log:     <path>/ecs/layers/meta-sec/log.html
Report:  <path>/ecs/layers/meta-sec/report.html
```

Figure 41: Example of executing test cases with pybot

Test cases for the IdAM solution have been described in ?? (??).

4.4.5 Writing Extensions for Robot Framework

In Robot, extensions can be written in case current features are not sufficient. However, most of the cases could be covered by calling third-party binaries by invoking them using **Run Process** and **Execute Command** from the Process library and the SSH library, respectively.

Internally at Ericsson, a security team has been developed in addition to the 'Nmap' and 'Hydra' libraries to cover needs from the ECS project:

- The Nmap library performs testing actions with regard to network port scanning and services.
- The Hydra library performs testing actions with regard to dictionary attacks for different protocol families.

Before delivering a product to the customer, it is not enough that a set of passed tests for a given functionality. Hardening actions (which are explained in the next chapter) are critical in that sense.

5. Hardening

Every software project which requires a base operating system and third party products requires hardening. Hardening is the process of reduce the surface of security vulnerabilities by remove unnecessary default functions and users, changing default passwords and remove or disable software services.

This is the case for ECS, which uses OpenStack and GNU/Linux. Default users must be disabled and default credentials must be randomized. Services should run with specific separate own users to restrict capabilities.

5.1 Entropy for Random Generation

The first problem appears in the randomization of the passwords in a virtualized environment. Entropy is insufficient and prediction of the pseudo randomized values increases significantly.

A significant part of the global state of the microprocessor is not architecturally visible through the instruction set (caches, branch predictors, and buffers). HArdware Volatile Entropy Gathering and Expansion (HAVEGE) leverages the uncertainty introduced in the internal states of the processor-external events.

First, the hardware clock cycle counter of the processor can be used to gather part of the uncertainty introduced by OS interruptions in the internal state of the processor.

As the HAVEG algorithm relies on the hardware clock cycle counter and combines with simple pseudo random number generation with a very high throughput (more than 100 Mbits/s of unpredictable random numbers), relying the security on its internal hardware state, not being accessible even for the user running the generator (Seznec A. and Sendrier N., 2002, HArdware Volatile Entropy Gathering and Expansion: Generating unpredictable random number at user level; A05. Seznec N. Sendrier N. 2012).

HAVEGE will add entropy directly to /dev/random, where the processes fetch entropy from the Linux kernel.

In order to install HAVEGE daemon (haveged) in a virtualized environment, the vim install script should be modified by adding: apt-get -y install haveged apg

The Apg tool was used to retrieve the random strings from /dev/random as a source.

A generate password script is also provided in case a new set of credentials needs to

be generated for the given services from the YAML format. See Haveged integration implementation in the Appendix Implementation; Haveged implementation.

5.2 Configuration Manager

A second problem appeared in the deployment across nodes of these generated values in case they need to share credentials for accessing remote services. A configuration manager was needed.

Configuration managers focused on this particular problem. They are designed to manage the installation, configuration, deployment, and maintenance of the service, as well as provide a full set of dependencies and triggers that keep the services up to date and act depending on the events that occurred.

5.3 Puppet Deployment

For configuration management, Puppet was selected for the ECS project.

In Puppet, the user describes system resources and their state, using Puppet declarative language or a Ruby Domain-Specific Language (DSL). This information is stored in files called puppet manifests. Puppet discovers the system information and compiles the puppet manifests into a system-specific catalogue containing resources and resource dependencies, which are applied against the target systems (W13. Several authors. 2015).

Custom puppet manifests describes the system configuration. These manifests apply to the system that needs to be configured by using agents and specifying in a high level of abstraction what configuration applies to each node as shown in Figure 42.

Each puppet module uses the generate-passwords, password-for-service scripts in order to access the passwords.yaml file. The next section describes a list of puppet modules implemented.

5.4 Puppet Modules

MySQL, OVFT, heat, and RabbitMQ puppet modules have been created to provide configurations for different services. In particular for the database which will store the orchestration of a managed cloud.

OVFT

OVFT is a template format that follows the OVF standards for packaging and distributing software to be run in virtual machines. This standard describes a secure, open, portable, efficient, and extensible format without being tied to any particular hypervisor or processor architecture.

See OVFT implementation in the Appendix Implementation; OVFT implementa-

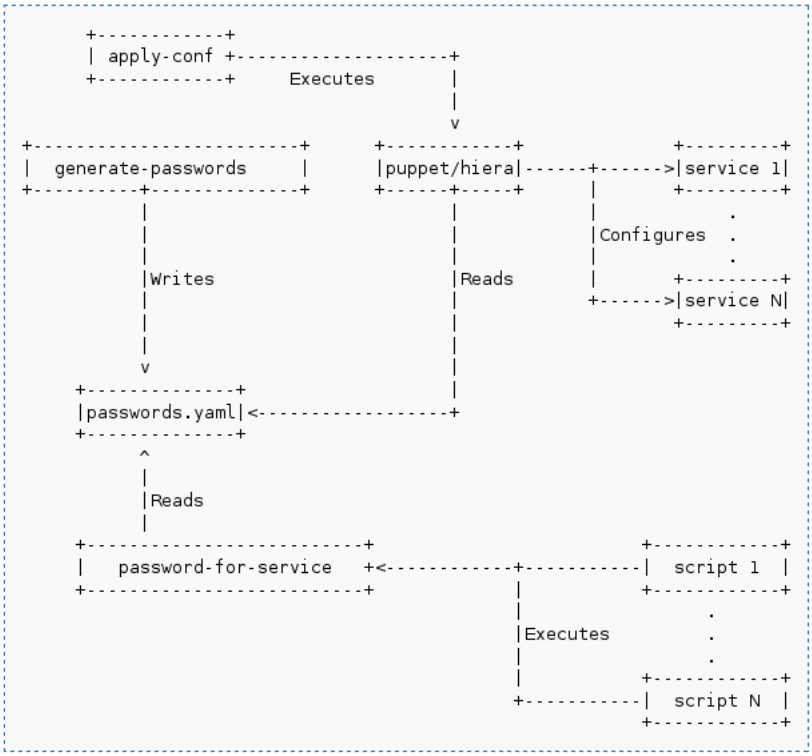


Figure 42: Design of configuration manager applied to the ECS project

tion.

Heat

Heat is a software suite that defines functions to simplify and automate processes in order to improve service quality.

See Heat implementation in the Appendix Implementation; Heat implementation.

RabbitMQ

RabbitMQ is an advance message queuing protocol for clustering and failover scenarios.

See RabbitMQ implementation in the Appendix Implementation; RabbitMQ implementation.

MySQL

MySQL is an open-source relational database management system that provides high-profile and large-scale database services.

See MySQL integration implementation in the Appendix Implementation; MySQL implementation.

Apply-conf

Apply configuration script is provided to run the installation and configuration deployment.

Password for services script is also provided. This will provide a simple but clean mechanism of password retrieval.

See apply-conf implementation in the Appendix Implementation; Apply-conf implementation.

Next chapter describes how centralized IdAM has been configured and integrated to the ECS solution.

6. Centralized IdAM Solution

Every single node in ECS needs to be accessed for different reasons. Several administrators requires access to the cloud. In order to provide a secure way of authentication, and distinguish which administrator accessed which node, Lightweight Directory Access Protocol (LDAP) has been integrated in the solution.

The LDAP is an internet protocol for accessing distributed directory services that act in accordance with X.500 data and service models.

The general model adopted by LDAP is one of the clients performing protocol operations against servers. A client transmits a protocol request describing the operation to be performed to a server, while the server is then responsible for performing the necessary operations in the directory. The server returns a response containing appropriate data to the requesting client.

Protocol operations are generally independent of one another. Each operation is processed as an atomic action, leaving the directory in a consistent state.

Requests and responses for multiple operations generally may be exchanged between a client and server in any order. But if required, synchronous behaviour may be controlled by client applications.

The core protocol operations are defined to a subset of the X.500: Directory Abstract Service (X.511).

Infrastructure administrators are stored in LDAP, while OpenStack users are stored in MariaDB and managed by a Keystone service.

6.1 Multi-Master Architecture

Each controller node has an LDAP instance installed. OpenLDAP has been used for implementation. The initial LDAP content is provisioned within the primary controller and replicated over to other instances. Next Figure gives a general idea about the IdAM architecture.

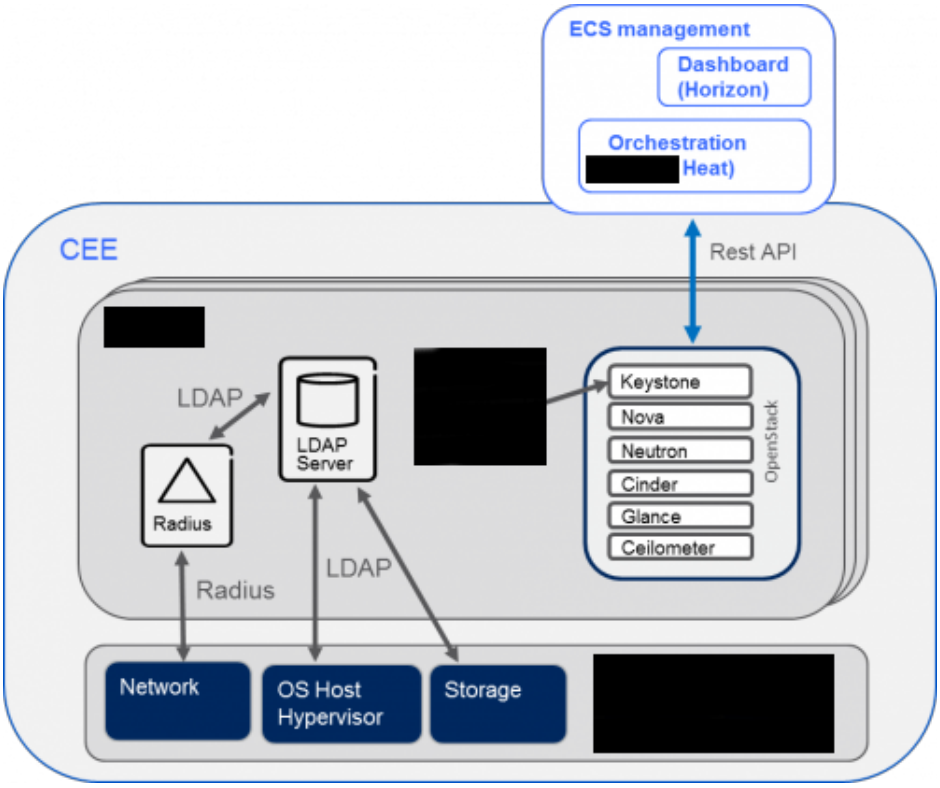


Figure 43: IdAM architectural overview

As seen in figure 43 and LDAP server is running as part of the CEE solution. It is used mainly to provide access to the hypervisor, storage and network elements. LDAP service is the main component of the IdAM solution. An LDAP service is using Abstract Syntax Notation One (ASN.1) as a description language, which it is a standard way to describe attributes through LDAP search.

6.2 IdAM Implementation

High Availability (HA) for LDAP has been implemented with the use of multi-master replication between multiple LDAP instances and HA proxy configuration.

HA proxy has been configured with the primary controller as the primary node and other controllers as backups. Stickiness has been implemented based on the destination to avoid flip-flopping in case of the primary instance being cyclic.

OpenLDAP multi-master replication is used to keep all LDAP instances in sync. Each LDAP instance acts as a provider to other instances.

The Remote Authentication Dial In User Service (RADIUS) server is implemented by deploying and configuring a FreeRADIUS server with a puppet on each controller.

The RADIUS server does not currently have any configurable parameters, except the client network definitions to control where the access to radius server is allowed.

The LDAP client has been implemented by deploying and configuring the GNU/Linux Pluggable Authentication Modules (PAM) client and nscd daemon with LDAP configuration. Extreme switches may authenticate using the RADIUS towards the accounts

created in LDAP with RADIUS authentication enabled. Additional details about implementation are not part of this thesis, as these have been covered by other developers.

See tests implementation for Identity and Access Management (IdAM) in the Appendix Implementation; IdAM tests implementation.

6.3 Testing Hardcoded Credentials

Apart from tests for IdAM, some other tests are needed to verify the default credentials for already existing accounts. This will verify that default passwords and accounts are disabled or randomized at deployment time.

See tests implementation for hardening in the Appendix Implementation; Hardening tests implementation.

7. Results

The previous design and implementation was verified by looking to the list and status of the Puppet Modules. This is the verification after migrating the services to the puppet modules in VIM. Please note that passwords shown are not used in any deployments.

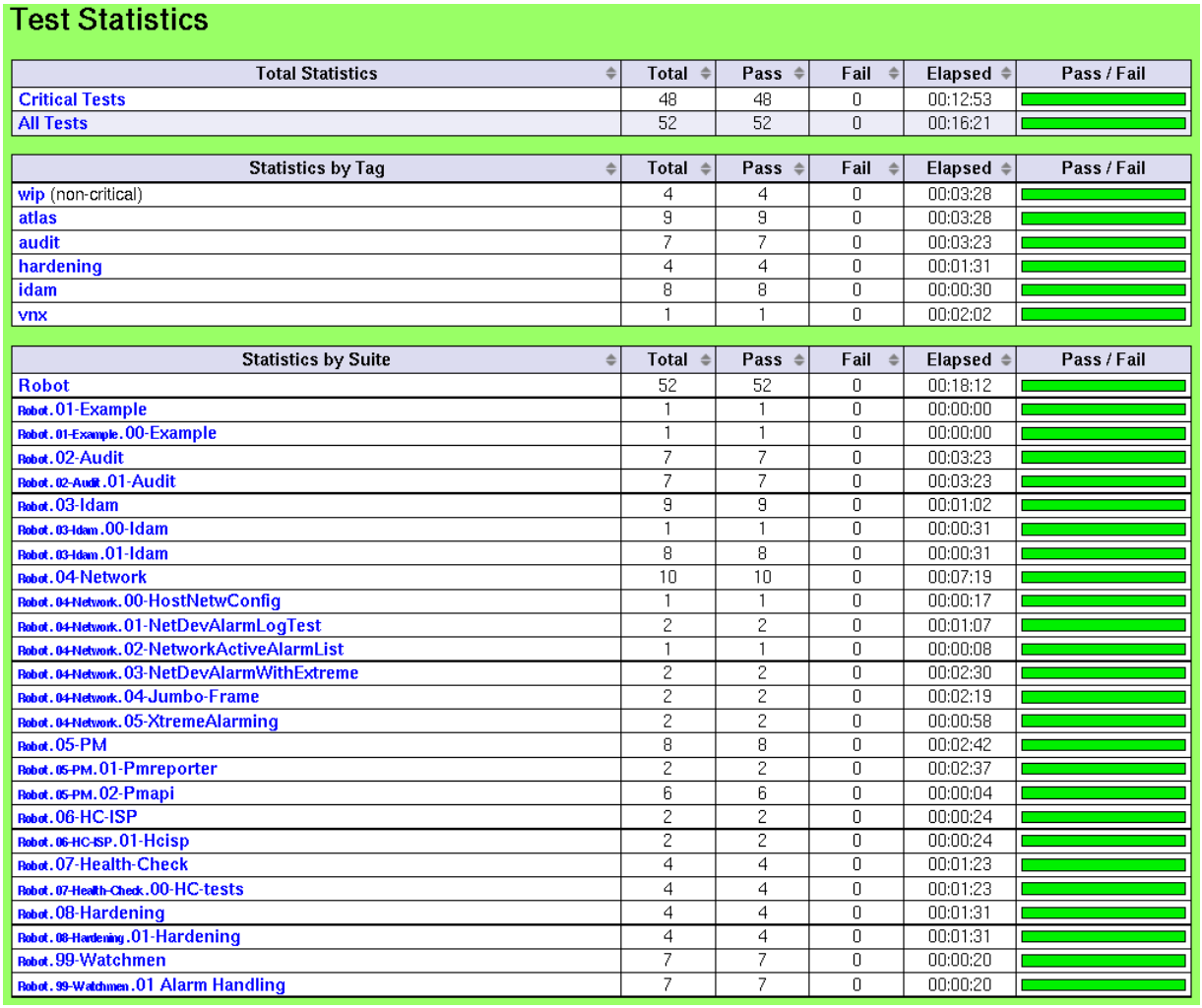
The idea of the Puppet modules here is to verify that all the functionality can be automatically managed. Each module represents the handler for a specific service. Administratively actions can be applied for a given service.

Below is shown a list of puppet modules executed in VIM. For listing puppet modules installed in a system **puppet module list** retrieves that information:

```
root@vim:~# puppet module list
/etc/puppet/modules
+-- vim-heat (v0.1.0)
+-- vim-ovft (v0.1.0)
+-- vim-rabbitmq (v0.1.0)
+-- vim-role (v0.1.0)
+-- nanliu-staging (v1.0.3)
+-- puppetlabs-mysql (v3.2.0)
+-- puppetlabs-stdlib (v4.5.1)
```

Each module verification details are shown in detail in the Appendix Implementation; Verification of the Puppet Modules

More results are shown by running tests which are executed as part of the CI process. Each test runs a set of instructions which verifies the functionality and security of the different services managed by Puppet. All tests have been successfully run, which means that VIC and compute nodes are working properly and securely. Figure 44 shows the results of the Robot test cases executed in CI.



Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	48	48	0	00:12:53	
All Tests	52	52	0	00:16:21	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
wip (non-critical)	4	4	0	00:03:28	
atlas	9	9	0	00:03:28	
audit	7	7	0	00:03:23	
hardening	4	4	0	00:01:31	
idam	8	8	0	00:00:30	
vnx	1	1	0	00:02:02	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Robot	52	52	0	00:18:12	
Robot.01-Example	1	1	0	00:00:00	
Robot.01-Example.00-Example	1	1	0	00:00:00	
Robot.02-Audit	7	7	0	00:03:23	
Robot.02-Audit.01-Audit	7	7	0	00:03:23	
Robot.03-Idam	9	9	0	00:01:02	
Robot.03-Idam.00-Idam	1	1	0	00:00:31	
Robot.03-Idam.01-Idam	8	8	0	00:00:31	
Robot.04-Network	10	10	0	00:07:19	
Robot.04-Network.00-HostNetwConfig	1	1	0	00:00:17	
Robot.04-Network.01-NetDevAlarmLogTest	2	2	0	00:01:07	
Robot.04-Network.02-NetworkActiveAlarmList	1	1	0	00:00:08	
Robot.04-Network.03-NetDevAlarmWithExtreme	2	2	0	00:02:30	
Robot.04-Network.04-Jumbo-Frame	2	2	0	00:02:19	
Robot.04-Network.05-XtremeAlarming	2	2	0	00:00:58	
Robot.05-PM	8	8	0	00:02:42	
Robot.05-PM.01-Pmreporter	2	2	0	00:02:37	
Robot.05-PM.02-Pmapi	6	6	0	00:00:04	
Robot.06-HC-ISP	2	2	0	00:00:24	
Robot.06-HC-ISP.01-Hcisp	2	2	0	00:00:24	
Robot.07-Health-Check	4	4	0	00:01:23	
Robot.07-Health-Check.00-HC-tests	4	4	0	00:01:23	
Robot.08-Hardening	4	4	0	00:01:31	
Robot.08-Hardening.01-Hardening	4	4	0	00:01:31	
Robot.99-Watchmen	7	7	0	00:00:20	
Robot.99-Watchmen.01 Alarm Handling	7	7	0	00:00:20	

Figure 44: Robot results for ECS running in CI

The final robot report from the above screenshot displayed in green, represents a successful run. Each line of the report is a set of test cases. In total 52 test cases in less than 16 minutes have been run. Automated tests were covering the areas of: auditability, IdAM, networking, performance management, health check, hardening, alarm handling and monitoring of the system. More complex tests from the implementation point of view have been performed manually.

Automated test cases have been performed iteratively for each change merged into the ECS software. Complex test cases were executed before releases. Each release was representing the maturity of this project. Results presented in this Master's thesis were representing the maturity of an ECS product release.

Conclusions

A lot of research was carried out in order to provide security for ECS. However, cloud environments are complex and involve a huge infrastructure of different devices of virtualized and physical nodes. Different networks are interconnected between them, with additional deployment of network devices such as firewalls, routers, and switches; they can be also virtual or physical. Additional devices and components are added to provide security, such as TPM and HSM. Some of the nodes are not always present and could appear at deployment time but some others are in operational mode. A full set of different users such as cloud administrators, cloud tenants, service users, and OpenStack services have different roles and would have different privilege levels as well as partial visibility of the whole CEE. Different systems for authenticating local and remote users were implemented. Authentication and authorization levels were deployed for different purposes such as RADIUS, LDAP, and PAM interconnected with an IdAM solution. Remote storage as well as several DBMS were installed for different purposes, and all of them need to be available for HA and replication. HA also applies for other services and users. Remote logs identifying a user or service from a particular host or node, or a device from a different region are also important in order to provide auditability. Security must also be provided in order to avoid attacks such as VLAN hopping and bypassing the hypervisor from a virtualized node. Automatic upgrades from software, configuration deployments, and third-party software installed by tenants are also adding new threats from cloud developers. DoS and DDoS need to be considered as well.

For all of this, the cloud is an extremely complex world and there are too many potential vector attacks. This thesis covered only a small portion of security implemented in order to provide security. Still it included after a huge effort and discussions regarding architecting and design. Implementation has been adapted for cloud needs and tested sufficiently. But this only covers a small part, and even one mistake could lead hackers to break into Ericsson's cloud, as hackers are always one step ahead. It will be Ericsson's responsibility to provide a good level of security. The company cannot blame security experts for finding bugs and holes that Ericsson was unable to find at the time of development.

Bibliography

[1] **B01** Kavis, M. (2014). Architecting the cloud: design decisions for cloud computing service models. Wiley.

[2] **B02** Bento A. and Aggarwal A. (2012). Cloud Computing Service and Deployment Models: Layers and Management. IGI Global.

[3] **B03** Gordon A. (2015). Official (ISC)2 Guide to the CISSP CBK. (ISC)2 Press.

[4] **B04** Henderson, C. (2006). Building Scalable Web Sites. Building, scaling, and optimizing the Next Generation of Web Applications. O'Reilly.

[5] **B05** Kaner C. Falk J. Nguyen H. (2006). Testing Computer Software. Wiley.

[6] **B06** Fyodor G. (2009). Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Nmap project.

[7] **B07** Chappell L. Combs G. (2012). Wireshark Network Analysis (Second Edition): The Official Wireshark Certified Network Analyst Study Guide. Laura Chappell University.

[8] **A01** Mell, P. Grance T. (2011). The NIST definition of Cloud Computing, Special Publication 800-145. National Institute of Standards and Technology.

[9] **A02** Archer J. Cullinane D. Puhlmann N. Boehme A. Kurtz P. Reavis J. (2011). Security Guidance for critical areas of focus in Cloud Computing V3.0, Special Publication 800-145. Cloud Security Alliance. CSA.

[10] **A03** Several authors. (2013). Payment Card Industry (PCI) Data Security Standard: Requirements security assessment procedures v3.0. PCI Security Standards Council.

[11] **A04** Barker E. Barker W. Burr W. Polk W. Smid M. (2012). Computer Security: Recommendation for Key Management - Part 1: General, Special Publication 800-57. National Institute of Standards and Technology.

[12] **A05** Seznec A. Sendrier N. (2012) HArdware Volatile Entropy Gathering and Expansion: generating unpredictable random numbers at user level. Institut de Recherche en Informatique et Systemes Aleatoire. IRISA.

[13] **W01** Several authors. Website for Cloud Controls Matrix (CCM): https://cloudsecurityalliance.org/research/ccm/ Accessed in 2015. Cloud Security Alliance. CSA.

[14] **W02** Several authors. Targeted attacks and opportunistic hacks: State of Cloud Security Report: https://www.alertlogic.com/downloads/AlertLogic_CloudSecurityReport_Spring13.pdf Accessed in 2015. Alertlogic. Security Compliance Cloud.

[15] **W03** Several authors. Tenable Network Security: security center 4.4 architecture http://static.tenable.com/prod_docs/SecurityCenter_4.4_Architecture.pdf Accessed in 2015. Tenable.

[16] **W04** Several authors. Tenable Network Security: Nessus 5.2 HTML5 User Guide: http://static.tenable.com/documentation/nessus_5.2_HTML5_user_guide.pdf Accessed in 2015. Tenable.

[17] **W05** Several authors. IxVM: Validating Virtualized Assets and Environments: https://www.ixiacom.com/sites/default/files/resources/datasheet/ixvm_1.pdf Accessed in 2014. IXIA Com.

[18] **W06** Several authors. IXIA: IxNetwork VXLAN Emulation: http://www.ixiacom.com/sites/default/files/resources/datasheet/ixnetwork_vxlan_emulation. Accessed in 2014. IXIA Com.

[19] **W07** Biondi P. Scapy community. Official Scapy website: Scapy v2.1.1-dev documentation. http://www.secdev.org/projects/scapy/doc/usage.html Accessed in 2014. Secdev.org.

[20] **W08** Several authors. Official GNU website: GNU Core utils. https://www.gnu.org/software/coreutils/manual/coreutils.html Accessed in 2014. Free Software Foundation (FSF).

[21] **W09** Several authors. Official GNU website: GNU Bin utils. https://sourceware.org/binutils/docs-2.25/binutils/index.html Accessed in 2014. Free Software Foundation (FSF).

[22] **W10** Several authors. Official Robot framework website: User guide. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html Accessed in 2014. Nokia Solutions and Networks.

[23] **W11** Several authors. Official Mirantis website: Fuel: User guide. http://docs.mirantis.com/openstack/fuel/fuel-6.0/pdf/Mirantis-OpenStack-6.0-UserGuide.pdf Accessed in 2015. Mirantis Inc.

[24] **W12** Several authors. Official OpenStack website: documentation. http://docs.openstack.org/ Accessed in 2014. OpenStack powered by Rackspace Cloud Computing.

[25] **W13** Several authors. Official Puppetlabs website: Puppet documentation: https://docs.puppetlabs.com/ Accessed in 2015. Puppet Labs.

[26] **W14** Ericsson copyright. Ericsson Cloud system Portfolio: http://www.ericsson.com/ourportfolio/products/cloud-system

[27] **W15** Sverdlik Y. (2015) Announce and news about Ericsson Cloud System technologies. http://www.datacenterknowledge.com/archives/2015/03/03/ericsson-cloud-system-to-use-abb-data-center-management/ Accessed in 2015. Datacenterknowledge journal

Appendices

Conventions

This thesis has been written entirely using free tools:

- LaTeXand TeXLive document markup language for professional quality document processing.
- AUCTeX v11.87 for writing and formating TeX files in GNU Emacs.
- PdfTex 3.14 with the following dependencies has been used to generate this pdf: libpng 1.6.13, zlib 1.2.8, poppler 0.26.5 and kpathsea 6.2.1.
- GNU Emacs v24.3.50.1 environment as an editor and an Integrated Development Environment (IDE).
- GNU Free Fonts has been used for the fonts.
- GNU Free Documentation License (GDFL) v1.3 (GNU Free Documentation License) has been used to licensing this report.
- GNU Gimp 2.8.14 has been used for image manipulation program.
- Xwd 1.0.6 has been used for dumping images of an X window, making screenshots.

The systems involved in the thesis that are included in the document are using the following technologies:

- GNU/Linux v3.x kernel are the systems used in the thesis.
- GNU bash version 4.2.37.

The following conventions has been used in this thesis:

- Regular content and explanations uses classic (OT1) font encoded computer modern roman, medium weight, normal shape, 12pt font.
- Literal quoted text uses classic (OT1) font encoded computer modern roman, medium weight, italic shape, 12pt font.
- Margings for the document are: text width used is 390 pt. Odd side margin is 39 pt. Even side margin is 40 pt. Top margin is 17 pt.
- Command line examples uses teletype family, courier, fixed font, 12pt font.

Harvard derived citation reference system has been used for referencing.

- Quoted text: *"Quoted text in italic"*. Full name of the author.
- Referenced text from a book: (Reference ID, Author(s), Year).

- Referenced text from an article: (Reference ID, Author(s), Year).
- Referenced text from a website: (Reference ID, Author(s), Year).
- Referenced image from a book: (Image title, Book title, p. page number).
- Referenced image from an article or website: (Image title, Article or website owner).
- Referenced internal Ericsson image or own image: (Image title).
- Articles will use reference ID as A*XX* format.
- Books will use reference ID as B*XX* format.
- Website will use reference ID as W*XX* format.
- Referenced texts will be detailed in Bibliography.
- Referenced Images will be detailed in List of Figures.
- Referenced tables will be detailed in List of Tables.

Implementation

Haveged

install.sh:

```
# Install haveged and apg tool
apt-get -y install haveged
apt-get -y install apg
# Configure haveged in boot time
chkconfig --add haveged
```

generate-passwords:

```
#!/usr/bin/env python

import argparse
import logging

import rand_passwds

class CliParser:

   def parse(self):
      description = 'Generate passwords for services in '\
                '{}'.format(rand_passwds.PASSWD_FILE)
      parser = argparse.ArgumentParser(description=description)

      parser.add_argument('-d', '--debug', action='store_true',
                     default=False, dest='debug',
                     help='print extra information')
      parser.add_argument('passwd_file', action='store',
         nargs='?',
                     default=rand_passwds.PASSWD_FILE,
                     help='input YAML password file')
      parser.add_argument('-s', '--stdout', action='store_true',
                     default=False, dest='stdout',
                     help='do not edit file, print to stdout')
      parser.add_argument('-q', '--quiet', action='store_true',
                     default=False, dest='quiet',
                     help='print only errors')

      args = parser.parse_args()
```

```
        if args.debug:
            logging.getLogger().setLevel(logging.DEBUG)
            logging.debug('Setting log level to verbose')
        if args.quiet:
            logging.getLogger().setLevel(logging.ERROR)
        return args

def main():

    rand_passwds.configure_logging()

    p = CliParser()
    args = p.parse()

    passwords = rand_passwds.HieraYamlPasswordFile()
    passwords.update()

    if args.stdout:
        passwords.write_to_stdout()
    else:
        passwords.write_to_file(args.passwd_file)
        logging.info('Passwords written to
            {}'.format(args.passwd_file))

if __name__ == '__main__':
    main()
```

OVFT

init.pp:

```
# == Class: ovft
#
# Class ovft.
#
# === Copyright
#
# Copyright 2014 Custom modified.
#
class ovft (
  $db_host = $ovft::params::db_host,
  $db_name = $ovft::params::db_name,
  $db_user = $ovft::params::db_user,
  $db_pass = $ovft::params::db_pass,

  $auth_key = $ovft::params::auth_key,

  $rabbit_host = $ovft::params::rabbit_host,
  $rabbit_pass = $ovft::params::rabbit_pass,

  $keystone_admin_user = $ovft::params::keystone_admin_user,
```

```
  $keystone_admin_pass = $ovft::params::keystone_admin_pass,
  $keystone_admin_tenant = $ovft::params::keystone_admin_tenant,

  $config_directory = $ovft::params::config_directory,

  $engine_config_file = $ovft::params::engine_config_file,
  $engine_config_template = $ovft::params::engine_config_template,

  $api_config_file = $ovft::params::api_config_file,
  $api_config_source = $ovft::params::api_config_source,

  $policy_config_file = $ovft::params::policy_config_file,
  $policy_config_source = $ovft::params::policy_config_source,

  $cache_directory = $ovft::params::cache_directory,

  $db_sync_command = $ovft::params::db_sync_command,

  $service_name = $ovft::params::service_name,
  $service_ensure = $ovft::params::service_ensure,
  $service_enable = $ovft::params::service_enable,

  $package_name = $ovft::params::package_name,
  $package_ensure = $ovft::params::package_ensure,
) inherits ovft::params {
  if $db_user == undef {
    fail('The parameter $db_user is undefined')
  }

  if $db_pass == undef {
    fail('The parameter $db_pass is undefined')
  }

  if $auth_key == undef {
    fail('The parameter $auth_key is undefined')
  }

  if $rabbit_pass == undef {
    fail('The parameter $rabbit_pass is undefined')
  }

  if $keystone_admin_pass == undef {
    fail('The parameter $keystone_admin_pass is undefined')
  }

  contain 'ovft::install'
  contain 'ovft::config'
  contain 'ovft::service'

  Class['::ovft::install'] ->
  Class['::ovft::config'] ~>
  Class['::ovft::service']
}
```

config.pp:

```
class ovft::config inherits ovft {
   file { $config_directory:
     ensure  => directory,
     owner   => 'ovft_owner_password',
     group   => 'ovft_group',
     mode    => hidden_permissions_in_octal,
   }

   file { $engine_config_file:
     ensure => file,
     owner => 'ovft_owner',
     group => 'ovft_group',
     mode  => 'hidden_permissions_in_octal',
     content => template($engine_config_template),
   }

   file { $api_config_file:
     ensure => file,
     owner => 'ovft_owner',
     group => 'ovft_group',
     mode  => 'hidden_permissions_in_octal',
     source => $api_config_source,
   }

   file { $policy_config_file:
     ensure => file,
     owner => 'ovft_owner',
     group => 'ovft_group',
     mode  => 'hidden_permissions_in_octal',
     source => $policy_config_source,
   }

   exec { $db_sync_command:
     path     => '/bin:/sbin:/usr/bin:/usr/sbin',
     refreshonly => true,
   }

   File[$config_directory] ->
   File[$engine_config_file, $api_config_file,
      $policy_config_file] ~>
   Exec[$db_sync_command]
}
```

params.pp:

```
class ovft::params {
  $db_host = 'hidden_host'
  $db_name = 'ovft_dbname'
  $db_user = undef
  $db_pass = undef
```

```
  $auth_key = undef

  $rabbit_host = 'hidden_host'
  $rabbit_pass = undef

  $keystone_admin_user = 'keystone_admin'
  $keystone_admin_pass = undef
  $keystone_admin_tenant = 'keystone_admin_tenant'

  $config_directory = '/etc/ovft'

  $engine_config_file = "$config_directory/ovft.conf"
  $engine_config_template = 'ovft/ovft.conf.erb'

  $api_config_file = "$config_directory/api-paste.ini"
  $api_config_source = 'puppet:///modules/ovft/api-paste.ini'

  $policy_config_file = "$config_directory/policy.json"
  $policy_config_source = 'puppet:///modules/ovft/policy.json'

  $cache_directory = '/var/cache/ovft'

  $db_sync_command = 'ovft-manage db_sync'

  $service_ensure = 'running'
  $service_enable = true
  $service_name = [ 'ovft-api', 'ovft-engine' ]

  $package_name = 'python-ovft'
  $package_ensure = 'latest'
}
```

service.pp:

```
class ovft::service inherits ovft {
   if ! ($service_ensure in [ 'running', 'stopped' ]) {
      fail('service_ensure parameter must be running or stopped')
   }

   service { $service_name:
      ensure   => $service_ensure,
      enable   => $service_enable,
      hasrestart => true,
   }
}
```

Modulefile:

```
name  'vim-ovft'
version '0.1.0'
author 'vim'
license 'Custom modified'
summary 'OVFT puppet module'
```

```
## Add dependencies, if any:
# dependency 'username/name', '>= 1.2.0'
```

metadata.json:

```
{
  "source": "UNKNOWN",
  "version": "0.1.0",
  "author": "vim",
  "project_page": "UNKNOWN",
  "dependencies": [

  ],
  "checksums": {
  },
  "description": "OVFT puppet module",
  "summary": "OVFT puppet module",
  "license": "Custom modified",
  "name": "vim-ovft",
  "types": [

  ]
}
```

ovft.conf.erb:

```
#
# This file is managed by Puppet, please do not make any manual
   changes
#

[database]
connection = mysql://<%= @db_user %>:<%= @db_pass %>@<%= @db_host
   %>/<%= @db_name %>?charset=utf8

[DEFAULT]
network_vlan_ranges = default:200:4000
rabbit_hosts = <%= @rabbit_host %>
rabbit_password = <%= @rabbit_pass %>
rpc_backend = ovft.openstack.common.rpc.impl_kombu
auth_encryption_key = <%= @auth_key %>

[keystone_authtoken]
signing_dir = /var/cache/ovft
admin_password = <%= @keystone_admin_pass %>
admin_user = <%= @keystone_admin_user %>
admin_tenant_name = <%= @keystone_admin_tenant %>
auth_uri = http://vic-pub-api:5000/v2.0
auth_protocol = http
auth_port = 5000
auth_host = vic-pub-api
```

```
[ec2authtoken]
auth_uri = http://vic-pub-api:5000/v2.0

[ovft_api]
bind_port = 8888
```

Heat

init.pp:

```
# == Class: heat
#
# Class heat.
#
# === Copyright
#
# Copyright 2014 Custom modified.
#
class heat (
  $db_host = $heat::params::db_host,
  $db_name = $heat::params::db_name,
  $db_user = $heat::params::db_user,
  $db_pass = $heat::params::db_pass,

  $auth_key = $heat::params::auth_key,

  $rabbit_hosts = $heat::params::rabbit_hosts,
  $rabbit_pass = $heat::params::rabbit_pass,

  $keystone_admin_user = $heat::params::keystone_admin_user,
  $keystone_admin_pass = $heat::params::keystone_admin_pass,
  $keystone_admin_tenant = $heat::params::keystone_admin_tenant,

  $config_directory = $heat::params::config_directory,

  $engine_config_file = $heat::params::engine_config_file,
  $engine_config_template = $heat::params::engine_config_template,

  $cache_directory = $heat::params::cache_directory,

  $db_sync_command = $heat::params::db_sync_command,

  $service_name = $heat::params::service_name,
  $service_ensure = $heat::params::service_ensure,
) inherits heat::params {
  if $db_user == undef {
    fail('The parameter $db_user is undefined')
  }

  if $db_pass == undef {
    fail('The parameter $db_pass is undefined')
```

```
  }

  if $auth_key == undef {
    fail('The parameter $auth_key is undefined')
  }

  if $rabbit_pass == undef {
    fail('The parameter $rabbit_pass is undefined')
  }

  if $keystone_admin_pass == undef {
    fail('The parameter $keystone_admin_pass is undefined')
  }

  contain 'heat::install'
  contain 'heat::config'
  contain 'heat::service'

  Class['::heat::install'] ->
  Class['::heat::config'] ~>
  Class['::heat::service']
}
```

config.pp:

```
class heat::config inherits heat {

  file { $engine_config_file:
    ensure => file,
    owner => 'heat_ownwer',
    group => 'heat_group',
    mode  => 'hidden_permissions_in_octal',
    content => template($engine_config_template),
  }

  exec { $db_sync_command:
    path => '/bin:/sbin:/usr/bin:usr/sbin',
    refreshonly => true,
  }

  File[$engine_config_file] ~>
  Exec[$db_sync_command]
}
```

params.pp:

```
class heat::params {
  $db_host = 'hidden_host'
  $db_name = 'heat_dbname'
  $db_user = undef
  $db_pass = undef

  $auth_key = undef
```

```
  $rabbit_hosts = 'hidden_host'
  $rabbit_pass = undef

  $keystone_admin_user = 'keystone_admin_user'
  $keystone_admin_pass = undef
  $keystone_admin_tenant = 'keystone_admin_tenant'

  $config_directory = '/etc/heat'

  $engine_config_file = "$config_directory/heat.conf"
  $engine_config_template = 'heat/heat.conf.erb'

  $cache_directory = '/var/cache/heat'

  $plugin_directory = '/usr/share/pyshared/heat/plugins'
  $plugin_test_directory =
     '/usr/share/pyshared/heat/tests/heat_plugins_tests'

  $db_sync_command = 'heat-manage db_sync'

  $service_ensure = 'running'
  $service_enable = true
  $service_name = ['heat-api', 'heat-api-cfn',
     'heat-api-cloudwatch', 'heat-engine']

  $package_name = ['python-heat', 'heat-api', 'heat-engine',
     'heat-api-cfn', 'heat-api-cloudwatch', 'heat-common']
  $package_ensure = 'latest'
}
```

service.pp:

```
class heat::service inherits heat {
  if ! ($service_ensure in ['running', 'stopped']) {
    fail('service_ensure parameter must be running or stopped')
  }

  service { $service_name:
    ensure   => $service_ensure,
    enable   => $service_enable,
    hasrestart => true,
  }
}
```

Modulefile:

```
name  'vim-heat'
version '0.1.0'
author 'vim'
license 'Custom modified'
summary 'Heat puppet module'
```

```
## Add dependencies, if any:
# dependency 'username/name', '>= 1.2.0'
```

metadata.json:

```
{
  "source": "UNKNOWN",
  "version": "0.1.0",
  "author": "vim",
  "project_page": "UNKNOWN",
  "dependencies": [

  ],
  "checksums": {
  },
  "description": "Heat puppet module",
  "summary": "Heat puppet module",
  "license": "Custom modified",
  "name": "vim-heat",
  "types": [

  ]
}
```

heat.conf.erb:

```
#
# This file is managed by Puppet, please do not make any manual
   changes
#

[DEFAULT]
rabbit_hosts = <%= @rabbit_hosts %>
rabbit_password = <%= @rabbit_pass %>
rpc_backend = heat.openstack.common.rpc.impl_kombu
plugin_dirs = /usr/share/pyshared/heat/plugins
auth_encryption_key = <%= @auth_key %>
heat_watch_server_url = http://hidden_host:8003
heat_waitcondition_server_url =
   http://hidden_host:8000/v1/waitcondition
heat_metadata_server_url = http://hidden_host:8000

[auth_password]

[clients]

[clients_ceilometer]

[clients_cinder]

[clients_heat]

[clients_keystone]
```

```
[clients_neutron]

[clients_nova]

[clients_swift]

[clients_trove]

[database]
connection=mysql://<%= @db_user %>:<%= @db_pass %>@<%= @db_host
   %>/<%= @db_name %>?charset=utf8

[ec2authtoken]
auth_uri = http://vic-pub-api:5000/v2.0

[heat_api]
bind_port = 8004

[heat_api_cfn]
bind_port = 8000

[heat_api_cloudwatch]
bind_port = 8003

[keystone_authtoken]
signing_dir = /var/cache/heat
admin_password = <%= @keystone_admin_pass %>
admin_user = <%= @keystone_admin_user %>
admin_tenant_name = <%= @keystone_admin_tenant %>
auth_uri = http://vic-pub-api:5000/v2.0
auth_protocol = http
auth_port = 5000
auth_host = vic-pub-api

[matchmaker_redis]

[matchmaker_ring]

[paste_deploy]

[revision]

[rpc_notifier2]

[ssl]
```

RabbitMQ

init.pp:

```
# == Class: rabbitmq
#
# Class rabbitmq.
#
# === Copyright
#
# Copyright 2014 Custom modified.
#
class rabbitmq (
  $rabbitmq_host = $rabbitmq::params::rabbitmq_host,
  $epmd_host  = $rabbitmq::params::epmd_host,
  $default_user = $rabbitmq::params::default_user,
  $default_pass = $rabbitmq::params::default_pass,

  $config_directory = $rabbitmq::params::config_directory,

  $engine_config_file = $rabbitmq::params::engine_config_file,
  $engine_config_template =
     $rabbitmq::params::engine_config_template,

  $env_config_file = $rabbitmq::params::env_config_file,
  $env_config_template = $rabbitmq::params::env_config_template,

  $plugins_config_file = $rabbitmq::params::plugins_config_file,
  $plugins_config_source =
     $rabbitmq::params::plugins_config_source,

  $service_name = $rabbitmq::params::service_name,
  $service_ensure = $rabbitmq::params::service_ensure,
  $service_enable = $rabbitmq::params::service_enable,

  $package_name = $rabbitmq::params::package_name,
  $package_ensure = $rabbitmq::params::package_ensure,

  $change_passwd_command = "rabbitmqctl change_password
     $default_user $default_pass"
) inherits rabbitmq::params {
  if $default_pass == undef {
    fail('The parameter $default_pass is undefined')
  }

  contain 'rabbitmq::install'
  contain 'rabbitmq::config'
  contain 'rabbitmq::service'

  Class['::rabbitmq::install'] ->
  Class['::rabbitmq::config'] ~>
  Class['::rabbitmq::service']
}
```

config.pp:

```
class rabbitmq::config inherits rabbitmq {
  file { $engine_config_file:
    ensure => file,
    owner => 'rabbitmq_owner',
    group => 'rabbitmq_group',
    mode  => 'hidden_permissions_in_octal',
    content => template($engine_config_template),
  }

  file { $env_config_file:
    ensure => file,
    owner => 'rabbitmq_owner',
    group => 'rabbitmq_group',
    mode  => 'hidden_permissions_in_octal',
    content => template($env_config_template),
  }

  file { $plugins_config_file:
    ensure => file,
    owner => 'rabbitmq_owner',
    group => 'rabbitmq_group',
    mode  => 'hidden_permissions_in_octal',
    source => $plugins_config_source,
  }
}
```

params.pp:

```
class rabbitmq::params {
  $rabbitmq_host = 'hidden_host'
  $epmd_host  = 'hidden_host'
  $default_user = 'default_rabbitmq_user'
  $default_pass = undef

  $config_directory = '/etc/rabbitmq'

  $engine_config_file = "$config_directory/rabbitmq.config"
  $engine_config_template = "rabbitmq/rabbitmq.config.erb"

  $env_config_file = "$config_directory/rabbitmq-env.conf"
  $env_config_template = "rabbitmq/rabbitmq-env.conf.erb"

  $plugins_config_file = "$config_directory/enabled_plugins"
  $plugins_config_source =
     'puppet:///modules/rabbitmq/enabled_plugins'

  $service_ensure = 'running'
  $service_enable = true
  $service_name = 'rabbitmq-server'

  $package_name = 'rabbitmq-server'
  $package_ensure = 'latest'
}
```

service.pp:

```
class rabbitmq::service inherits rabbitmq {
  if ! ($service_ensure in ['running', 'stopped']) {
    fail('service_ensure parameter must be running or stopped')
  }

  exec { $change_passwd_command:
    path     => '/bin:/sbin:/usr/bin:/usr/sbin',
    refreshonly => true,
  }

  service { $service_name:
    ensure   => $service_ensure,
    enable   => $service_enable,
    hasrestart => true,
  }

  Service[$service_name] ->
  Exec[$change_passwd_command]
}
```

Modulefile:

```
name  'vim-rabbitmq'
version '0.1.0'
author 'vim'
license 'Custom modified'
summary 'RabbitMQ puppet module'

## Add dependencies, if any:
# dependency 'username/name', '>= 1.2.0'
```

metadata.json:

```
{
  "source": "UNKNOWN",
  "version": "0.1.0",
  "author": "vim",
  "project_page": "UNKNOWN",
  "dependencies": [

  ],
  "checksums": {

  },
  "description": "RabbitMQ puppet module",
  "summary": "RabbitMQ puppet module",
  "license": "Custom modified",
  "name": "vim-rabbitmq",
  "types": [

  ]
```

```
}
```

enabled_plugins.erb:

```
[rabbitmq_management].
```

rabbitmq-env.conf.erb:

```
export RABBITMQ_NODE_IP_ADDRESS=<%= @rabbitmq_host %>
export RABBITMQ_NODE_PORT=5672
export ERL_EPMD_ADDRESS=<%= @epmd_host %>
```

rabbitmq.config.erb:

```
[
  {rabbit, [
    {default_user, <<"<%= @default__user %>">>},
    {default_pass, <<"<%= @default_pass %>">>}
  ]},
  {rabbitmq_management,
    [{listener, [
      {ip,   "<%= @rabbitmq_host %>"},
      {port, 15671}]}
  ]},
  {kernel, [
    {inet_dist_use_interface, {<%= @rabbitmq_host.tr('.', ',') %>}}
  ]}
].
% EOF
```

MySQL

vim.pp:

```
class role::vim (
  $mysql_ovft_user = $role::vim::params::mysql_ovft_user,
  $mysql_ovft_pass = $role::vim::params::mysql_ovft_pass,
  $mysql_ovft_db_name = $role::vim::params::mysql_ovft_db_name,
  $ovft_auth_key  = $role::vim::params::ovft_auth_key,

  $mysql_heat_user = $role::vim::params::mysql_heat_user,
  $mysql_heat_pass = $role::vim::params::mysql_heat_pass,
  $mysql_heat_db_name = $role::vim::params::mysql_heat_db_name,
  $heat_auth_key  = $role::vim::params::heat_auth_key,

  $rabbitmq_pass  = $role::vim::params::rabbitmq_pass,

  $keystone_admin_pass = $role::vim::params::keystone_admin_pass,
) inherits role::vim::params {
```

```
if $mysql_ovft_pass == undef {
  fail('The parameter $mysql_ovft_pass is undefined')
}

if $ovft_auth_key == undef {
  fail('The parameter $ovft_auth_key is undefined')
}

if $mysql_heat_pass == undef {
  fail('The parameter $mysql_heat_pass is undefined')
}

if $heat_auth_key == undef {
  fail('The parameter $heat_auth_key is undefined')
}

if $rabbitmq_pass == undef {
  fail('The parameter $rabbitmq_pass is undefined')
}

if $keystone_admin_pass == undef {
  fail('The parameter $keystone_admin_pass is undefined')
}

include '::mysql::server'

mysql::db { $mysql_ovft_db_name:
  user   => $mysql_ovft_user,
  password => $mysql_ovft_pass,
  host   => 'hidden_host',
  grant  => [ 'ALL' ],
}

mysql::db { $mysql_heat_db_name:
  user   => $mysql_heat_user,
  password => $mysql_heat_pass,
  host   => 'hidden_host',
  grant  => [ 'ALL' ],
}

class { '::rabbitmq':
  default_pass => $rabbitmq_pass,
}

class { '::ovft':
  db_user         => $mysql_ovft_user,
  db_pass         => $mysql_ovft_pass,
  auth_key        => $ovft_auth_key,
  rabbit_pass     => $rabbitmq_pass,
  keystone_admin_pass => $keystone_admin_pass,
}

class { '::heat':
```

```
    db_user         => $mysql_heat_user,
    db_pass         => $mysql_heat_pass,
    auth_key        => $heat_auth_key,
    rabbit_pass     => $rabbitmq_pass,
    keystone_admin_pass => $keystone_admin_pass,
  }

  Class['::rabbitmq'] ->
  Class['::ovft', '::heat']

  Class['::mysql::server'] ->
  Mysql::Db[$mysql_ovft_db_name, $mysql_heat_db_name] ->
  Class['::ovft', '::heat']
}
```

init.pp:

```
# == Class: role
#
# Full description of class role here.
#
# === Parameters
#
# Document parameters here.
#
# [*sample_parameter*]
#  Explanation of what this parameter affects and what it
   defaults to.
#  e.g. "Specify one or more upstream ntp servers as an array."
#
# === Variables
#
# Here you should define a list of variables that this module
   would require.
#
# [*sample_variable*]
#  Explanation of how this variable affects the funtion of this
   class and if
#  it has a default. e.g. "The parameter enc_ntp_servers must be
   set by the
#  External Node Classifier as a comma separated list of
   hostnames." (Note,
#  global variables should be avoided in favor of class
   parameters as
#  of Puppet 2.6.)
#
# === Examples
#
# class { role:
#   servers => [ 'pool.ntp.org', 'ntp.local.company.com' ],
# }
#
# === Authors
```

```
#
# Author Name <author@domain.com>
#
# === Copyright
#
# Copyright 2014 Custom modified.
#
class role {

}
```

params.pp:

```
class role::vim::params {
  $mysql_ovft_user = 'mysql_ovft_user'
  $mysql_ovft_pass = undef
  $mysql_ovft_db_name = 'ovft_dbname'
  $ovft_auth_key  = undef

  $mysql_heat_user = 'mysql_heat_user'
  $mysql_heat_pass = undef
  $mysql_heat_db_name = 'heat_dbname'
  $heat_auth_key  = undef

  $rabbitmq_pass  = undef

  $keystone_admin_pass = undef
}
```

Modulefile:

```
name  'vim-role'
version '0.1.0'
source 'UNKNOWN'
author 'vim'
license 'Apache License, Version 2.0'
summary 'UNKNOWN'
description 'UNKNOWN'
project_page 'UNKNOWN'

## Add dependencies, if any:
# dependency 'username/name', '>= 1.2.0'
```

metadata.json:

```
{
  "project_page": "UNKNOWN",
  "types": [

  ],
  "version": "0.1.0",
  "license": "Apache License, Version 2.0",
```

```
  "source": "UNKNOWN",
  "author": "vim",
  "summary": "UNKNOWN",
  "dependencies": [

  ],
  "checksums": {
    "spec/spec_helper.rb": "a55d1e6483344f8ec6963fcb2c220372",
    "Modulefile": "6a76335a9841605cd7106223c86341b7",
    "tests/init.pp": "b9e27e86b26e44ffad77bd2db170e079",
    "README": "9e0323e43ee03187d619d09c869aeef7",
    "manifests/init.pp": "12643004cb1f59c1b82ed9c704e615cc"
  },
  "description": "UNKNOWN",
  "name": "vim-role"
}
```

hiera.yaml:

```
---
:backends: yaml
:yaml:
  :datadir: /etc/puppet/hieradata
:hierarchy:
  - passwords
  - common
:logger: console
```

common.yaml:

```
---
classes:
role::vim::keystone_admin_pass: keystone_admin_password
```

site.pp:

```
include role::vim
```

Apply-conf

apply-conf:

```
#!/bin/bash

set -o nounset
set -o errexit

export PATH="/bin:/sbin:/usr/bin:/usr/sbin"

readonly PUPPET_MAIN='/etc/puppet/manifests/site.pp'
```

```
readonly SYSLOG_TAG='apply-conf/puppet'
readonly SYSLOG_PRIORITY='local4.info'

rm_terminal_colors() {
   sed -r 's/\x1B\[([0-9]{1,2}(;[0-9]{1,2})?)?[m|K]//g'
}

to_syslog() {
   rm_terminal_colors | logger --priority "${SYSLOG_PRIORITY}" \
      --tag "${SYSLOG_TAG}"
}

main() {
   echo '>>>' | to_syslog
   puppet apply $@ ${PUPPET_MAIN} 2>&1 | tee >(to_syslog)
   echo '<<<' | to_syslog
}

main $@
```

passwords-for-service:

```
#!/usr/bin/env python

import argparse

import rand_passwds

VALID_SERVICES = rand_passwds.SERVICES.keys()


class CliParser:

   def __init__(self):
      description = 'Retrieve a password for a given service '\
                'from {}'.format(rand_passwds.PASSWD_FILE)
      self.parser =
         argparse.ArgumentParser(description=description)

   def parse(self):
      self.parser.add_argument('-f', '--file', action='store',
                        default=rand_passwds.PASSWD_FILE,
                        dest='passwd_file',
                        help='input YAML password file')
      self.parser.add_argument('-l', '--list',
         action='store_true',
                        default=False, dest='list',
                        help='lists possible services')
      self.parser.add_argument('service', action='store',
         nargs='?',
                        choices=VALID_SERVICES,
                        help='service token', default=None)
      return self.parser.parse_args()
```

```
    def print_help(self):
        self.parser.print_help()


def main():

    p = CliParser()
    args = p.parse()

    if args.list:
        print '\n'.join(VALID_SERVICES)
    elif args.service:
        passwords =
            rand_passwds.HieraYamlPasswordFile(args.passwd_file)
        passwords.read()
        print passwords.find(args.service)
    else:
        p.print_help()

if __name__ == '__main__':
    main()
```

generate-passwords:

```
#!/usr/bin/env python

import argparse

import rand_passwds

VALID_SERVICES = rand_passwds.SERVICES.keys()


class CliParser:

    def __init__(self):
        description = 'Retrieve a password for a given service '\
                      'from {}'.format(rand_passwds.PASSWD_FILE)
        self.parser =
            argparse.ArgumentParser(description=description)

    def parse(self):
        self.parser.add_argument('-f', '--file', action='store',
                                 default=rand_passwds.PASSWD_FILE,
                                 dest='passwd_file',
                                 help='input YAML password file')
        self.parser.add_argument('-l', '--list',
            action='store_true',
                                 default=False, dest='list',
                                 help='lists possible services')
        self.parser.add_argument('service', action='store',
            nargs='?',
```

```
                        choices=VALID_SERVICES,
                        help='service token', default=None)
      return self.parser.parse_args()

   def print_help(self):
      self.parser.print_help()


def main():

   p = CliParser()
   args = p.parse()

   if args.list:
      print '\n'.join(VALID_SERVICES)
   elif args.service:
      passwords =
         rand_passwds.HieraYamlPasswordFile(args.passwd_file)
      passwords.read()
      print passwords.find(args.service)
   else:
      p.print_help()

if __name__ == '__main__':
   main()
```

Tests Implementation

Idam test cases verifies that AAA policies are applied properly.

01-idam.robot:

```
*** Settings ***
Library     String
Library     Collections
Resource    idam-cee.robot
Resource    common.robot
Force Tags  idam
Suite Setup Run Keywords Select random target vic
            ...        LDAP and nscd are up on vics
Test Setup  Run Keywords Initialize random username
            ...        The user is created
Test Teardown Run Keywords Remove user
            ...        Reset all ssh connections

*** Test cases ***
User cannot set weak passwords
   When a weak password is suggested
   Then password update fails

User can set strong password and login to all nodes
```

```
   When a strong password is set for the user
   Then the user can authenticate successfully to ${vics}
   And the user can authenticate successfully to ${hosts}

User cannot reuse the existing password
   Given a strong password is set for the user
   When the same password is suggested
   Then password update fails

User is forced to change expired password after login
   Given a strong password is set for the user
   When password is forced to expire
   Then the user is asked to change the password after login

User cannot set a historical password
   When a historical password is suggested
   Then password update fails

User in the sudo group can run sudo commands
   When the user is added to sudo group
   Then the user can execute sudo command in ${hosts}

User not in sudo group cannot run sudo commands
   Given a strong password is set for the user
   When the user is not added to sudo group
   Then the user cannot execute sudo command in ${vics}
   And the user cannot execute sudo command in ${hosts}

There are no groups left over after deleting the user
   When the user is added to sudo group, deleted and created again
   Then the user cannot execute sudo command in ${vics}
   And the user cannot execute sudo command in ${hosts}
```

idam-cee.robot:

```
*** Variables ***
${ldap_service} slapd
${nscd_service} nscd

*** Keywords ***
Initialize random username
   ${user} Generate Random String 7 [LOWER]
   Set Suite Variable ${user}

LDAP and nscd are up on vics
   :FOR  ${node}  IN   @{vics}
   \   Switch Connection ${node.name}
   \   ${stderr} ${ldap_service_rc} Execute Command service
      ${ldap_service} status return_stdout=False return_rc=True
      return_stderr=True
   \   ${stderr} ${nscd_service_rc} Execute Command service
      ${nscd_service} status return_stdout=False return_rc=True
      return_stderr=True
```

```
    \   Should Be Equal As Integers ${ldap_service_rc} 0
    \   Should Be Equal As Integers ${nscd_service_rc} 0

Fail if the user does not exist
    ${stderr} ${rc}  Execute command id ${user} return_stdout=False
       return_rc=True return_stderr=True
    Should Be Equal As Integers ${rc} 0

The user is created
    Switch Connection ${target_vic.name}
    ${stderr} ${rc}  Execute Command cee-idam user-create ${user}
       return_stdout=False return_rc=True return_stderr=True
    Should Be Equal As Integers ${rc} 0
    Wait Until Keyword Succeeds 30 s 1 s Fail if the user does not
       exist

Remove user
    Switch Connection ${target_vic.name}
    ${stderr} ${rc}  Execute Command cee-idam user-delete -l
       ${user} return_stdout=False return_rc=True
       return_stderr=True
    Should Be Equal As Integers ${rc} 0

Set password ${password} for the user
    Switch Connection ${target_vic.name}
    Write  passwd ${user}
    Read Until New password:
    Write  ${password}
    Read Until Re-enter new password:
    Write  ${password}
    ${stdout} Read Until \#
    ${status} Run Keyword and Return Status Should Contain
       ${stdout} password updated
    ${rc}  Convert To Boolean ${status}
    [Return] ${status}

A weak password is suggested
    ${password} Generate Random String 7 [LETTERS][NUMBERS]
    Set Test Variable ${password}

Password update fails
    ${rc}  Run Keyword and continue on failure Set password
       ${password} for the user
    Should Be True ${rc} == False

A strong password is set for the user
    ${password} Generate Random String 16 [LETTERS][NUMBERS]
    Set Test Variable ${password}
    ${rc}  Set password ${password} for the user
    Should Be True ${rc} == True

The user can authenticate successfully to ${nodes}
    :FOR  ${node} IN  @{nodes}
```

```
    \   Open Connection ${node.address} port=${node.port}
    \   ${can_login} Run Keyword And Return Status Login ${user}
        ${password}
    \   Should Be True ${can_login}

The same password is suggested
  No Operation

Password is forced to expire
    Switch Connection ${target_vic.name}
    Write           cee-idam user-modify -l ${user} -e
    Read Until      User modification completed

The user is asked to change the password after login
    Open Connection ${hosts[0].address} port=${hosts[0].port}
    ${login_stdout} Login           ${user} ${password}
    Should Contain ${login_stdout}  You must change your password
        now and login again!

Another password is set for the user
    Set Test Variable ${historical_password} ${password}
    A strong password is set for the user

A historical password is suggested
    A strong password is set for the user
    Another password is set for the user
    ${set_password} Run Keyword and continue on failure Set
        password ${historical_password} for the user
    Set Test Variable ${set_password}

The user is added to sudo group
    A strong password is set for the user
    ${stderr} ${rc}  Execute Command cee-idam user-modify -l
        ${user} -G sudo return_stdout=False return_rc=True
        return_stderr=True
    Should Be Equal As Integers ${rc} 0

The user can execute sudo command in ${nodes}
    :FOR  ${node} IN  @{nodes}
    \   Open Connection ${node.address} port=${node.port}
    \   ${can_login} Run Keyword And Return Status Login ${user}
        ${password}
    \   Should Be True ${can_login}
    \   Write  sudo id
    \   Read Until [sudo] password for ${user}:
    \   Write  ${password}
    \   ${stdout} Read Until \$
    \   Should Contain ${stdout} uid=0(root) gid=0(root)
        groups=0(root)

The user is not added to sudo group
    No operation
```

```
The user cannot execute sudo command in ${nodes}
   :FOR  ${node} IN  @{nodes}
   \   Open Connection ${node.address} port=${node.port}
   \   ${can_login} Run Keyword And Return Status Login ${user}
      ${password}
   \   Should Be True ${can_login}
   \   Write  sudo id
   \   Read Until [sudo] password for ${user}:
   \   Write  ${password}
   \   ${stdout} Read Until \$
   \   Should Contain ${stdout} ${user} is not in the sudoers
      file.

Select random target vic
   ${total_nodes} Get Length ${vics}
   ${index}  Evaluate  random.randint(0, ${total_nodes}-1) random
   Set Suite Variable ${target_vic} ${vics[${index}]}

The user is added to sudo group, deleted and created again
   A strong password is set for the user
   The user is added to sudo group
   Remove user
   The user is created
   A strong password is set for the user
   The user is not added to sudo group
```

Hardening tests verifies the security of the system by reducing its surface of vulnerability; typically by randomizing default passwords, disable default accounts, close unnecessary software and services.

01-hardening.robot:

```
*** Settings ***
Library     String
Library     Collections
Resource    hardening-cee.robot
Resource    common.robot
Force Tags  wip hardening
Suite Setup Select random target vic
Test Setup  Initialize random username
Test Teardown Run Keywords Remove user
            ...        Reset all ssh connections

*** Test cases ***
No service accounts have a password set that matches the username
   in vics
   Given the user is created
   And get all users existing in a random ${vics}
   When login with same password as username, then it fails

No service accounts have a password set that matches the username
   in computes
   Given the user is created
```

```
   And get all users existing in a random ${hosts}
   When login with same password as username, then it fails

No service accounts have an empty password set in vics
   Given the user is created
   And get all users existing in a random ${vics}
   When login with empty password for each service, then it fails

No service accounts have an empty password set in computes
   Given the user is created
   And get all users existing in a random ${hosts}
   When login with empty password for each service, then it fails
```

hardening-cee.robot:

```
*** Keywords ***
Initialize random username
   ${user} Generate Random String 7 [LOWER]
   Set Suite Variable ${user}

Fail if the user does not exist
   ${rc}  Execute command id ${user} return_stdout=False
      return_rc=True
   Should Be Equal As Integers ${rc} 0

The user is created
   Switch Connection ${target_vic.name}
   ${stderr} ${rc}  Execute Command cee-idam user-create ${user}
      return_stdout=False return_rc=True return_stderr=True
   Should Be Equal As Integers ${rc} 0
   Wait Until Keyword Succeeds 30 s 1 s Fail if the user does not
      exist

Remove user
   Switch Connection ${target_vic.name}
   ${stderr} ${rc}  Execute Command cee-idam user-delete -l
      ${user} return_stdout=False return_rc=True
      return_stderr=True

   Should Be Equal As Integers ${rc} 0

Set password ${password} for the user
   Switch Connection ${target_vic.name}
   Write  passwd ${user}
   Read Until New password:
   Write  ${password}
   Read Until Re-enter new password:
   Write  ${password}
   ${stdout} Read Until \#
   ${status} Run Keyword and Return Status Should Contain
      ${stdout} password updated
   ${rc}  Convert To Boolean ${status}
   [Return] ${status}
```

```
A strong password is set for the user
   ${password} Generate Random String 16 [LETTERS][NUMBERS]
   Set Test Variable ${password}
   ${rc}  Set password ${password} for the user
   Should Be True ${rc} == True

Select random target vic
   ${total_nodes} Get Length ${vics}
   ${index}  Evaluate  random.randint(0, ${total_nodes}-1) random
   Set Suite Variable ${target_vic} ${vics[${index}]}

Select random target from ${nodes}
   ${total_nodes} Get Length ${nodes}
   ${index}  Evaluate  random.randint(0, ${total_nodes}-1) random
   Set Test Variable ${target_node} ${nodes[${index}]}

Get all users existing in a random ${node}
   A strong password is set for the user
   Select random target from ${vics}
   Open Connection ${target_node.address} port=${target_node.port}
   ${can_login} Run Keyword And Return Status Login ${user}
      ${password}
   Should Be True ${can_login}
   ${output} ${stderr} ${rc} Execute Command getent passwd|cut
      -d':' -f1 return_stdout=True return_rc=True
      return_stderr=True
   Should Be Equal As Integers ${rc} 0
   @{users} = Split To Lines ${output}
   Set Test Variable ${users}
   Close Connection

Login with same password as username, then it fails
   :FOR  ${userx} IN  @{users}
   \   Open Connection ${target_node.address}
      port=${target_node.port}
   \   ${output} Run Keyword And Expect Error * Login ${userx}
      ${userx}
   \   Should Contain ${output} Authentication failed for user
      '${userx}'.
   \   Close Connection

Login with empty password for each service, then it fails
   :FOR  ${userx} IN  @{users}
   \   Open Connection ${target_node.address}
      port=${target_node.port}
   \   ${output} Run Keyword And Expect Error * Login ${userx}
      ${EMPTY}
   \   Should Contain ${output} Authentication failed for user
      '${userx}'.
   \   Close Connection
```

Verifying password generation and dump in yaml format by a generate-passwords script. Simply run **generate-passwords** script:

```
root@vim:~# generate-passwords
root@vim:~# cat /etc/puppet/hieradata/passwords.yaml
---
role::vim::heat_auth_key: DodranMapChiUlg0
role::vim::mysql_heat_pass: ArpAijNunlyntAm7
role::vim::mysql_ovft_pass: SosaunIkfecysAk8
role::vim::ovft_auth_key: RieckHydyenPord7
role::vim::rabbitmq_pass: Mak2FuvReecWutgu
```

For checking different services under a GNU/Linux system, the **service <name of the service> status** command is used.

Check service status for heat service in VIM:

```
root@vim:/# service heat-api status
heat-api start/running, process 12697
root@vim:/# service heat-api-cloudwatch status
heat-api-cloudwatch start/running, process 12706
root@vim:/# service heat-api-cfn status
heat-api-cfn start/running, process 12685
root@vim:/# service heat-engine status
heat-engine start/running, process 12690
```

password-for-service <name of service> script is used to retrieve for a password of a given service. Verification of password for heat service script and service is accessed with the randomized credential and having restricted visibility. To check the mysql database for the heat service **mysql -u <user> -D <database name> -p<password>** command is used. Once connected to the database, **show databases;** will show the accessible databases:

```
root@vim:/# password-for-service mysql-heat-pass
ArpAijNunlyntAm7
root@vim:/etc/puppet/hieradata# mysql -u heat -D heat
   -pArpAijNunlyntAm7
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.5.37-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All
   rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current
   input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| heat               |
+--------------------+
2 rows in set (0.00 sec)

mysql> Bye
```

Check service status for OVFT service in VIM:

```
root@vim:/# service ovft-api status
 * Checking for service OVFT API...                * ovft-api is
    running
root@vim:/# service ovft-engine status
 * Checking for service OVFT ENGINE...             * ovft-engine is
    running
```

Same approach is done for the OVFT service. Verification of password for OVFT service script and service is accessed with the randomized credential and having restricted visibility:

```
root@vim:/# password-for-service mysql-ovft-pass
SosaunIkfecysAk8
root@vim:/etc/puppet/hieradata# mysql -u ovft -D ovft
   -pSosaunIkfecysAk8
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 40
Server version: 5.5.37-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All
   rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current
   input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
```

```
| information_schema |
| ovft               |
+--------------------+
2 rows in set (0.01 sec)

mysql> Bye
```

Check service status for RabbitMQ service in VIM:

```
root@vim:/# service rabbitmq-server status
Status of node rabbit@vim ...
[{pid,12474},
 {running_applications,
    [{rabbitmq_management,"RabbitMQ Management Console","3.3.5"},
     {rabbitmq_management_agent,"RabbitMQ Management
        Agent","3.3.5"},
     {rabbit,"RabbitMQ","3.3.5"},
     {os_mon,"CPO CXC 138 46","2.2.7"},
     {rabbitmq_web_dispatch,"RabbitMQ Web Dispatcher","3.3.5"},
     {webmachine,"webmachine","1.10.3-rmq3.3.5-gite9359c7"},
     {mochiweb,"MochiMedia Web
        Server","2.7.0-rmq3.3.5-git680dba8"},
     {amqp_client,"RabbitMQ AMQP Client","3.3.5"},
     {xmerl,"XML parser","1.2.10"},
     {inets,"INETS CXC 138 49","5.7.1"},
     {mnesia,"MNESIA CXC 138 12","4.5"},
     {sasl,"SASL CXC 138 11","2.1.10"},
     {stdlib,"ERTS CXC 138 10","1.17.5"},
     {kernel,"ERTS CXC 138 10","2.14.5"}]},
 {os,{unix,linux}},
 {erlang_version,
    "Erlang R14B04 (erts-5.8.5) [source] [64-bit] [smp:2:2]
       [rq:2] [async-threads:30] [kernel-poll:true]\n"},
 {memory,
    [{total,32021176},
     {connection_procs,160672},
     {queue_procs,173600},
     {plugins,368472},
     {other_proc,9160720},
     {mnesia,91232},
     {mgmt_db,138712},
     {msg_index,39904},
     {other_ets,1082856},
     {binary,190776},
     {code,17261817},
     {atom,1560961},
     {other_system,1791454}]},
 {alarms,[]},
 {listeners,[{clustering,25672,"::"},{amqp,5672,"hidden_host"}]},
 {vm_memory_high_watermark,0.4},
 {vm_memory_limit,856801280},
```

```
 {disk_free_limit,50000000},
 {disk_free,7061319680},
 {file_descriptors,
    [{total_limit,924},{total_used,7},{sockets_limit,829},{sockets_used,5}]},
 {processes,[{limit,1048576},{used,226}]},
 {run_queue,0},
 {uptime,356}]
...done.
```

Verification of password for RabbitMQ service script and service is accessed with the randomized credential and having restricted visibility. As this service it has different nature from the previous service, it was tested by requesting access to an API through HTTP protocol by running **curl** command:

```
root@vim:/# password-for-service rabbitmq-pass
Mak2FuvReecWutgu
root@vim:/# curl -i -u guest:Mak2FuvReecWutgu
   http://hidden_host:15671/api/whoami
HTTP/1.1 200 OK
Server: MochiWeb/1.1 WebMachine/1.10.0 (never breaks eye contact)
Date: Wed, 04 Mar 2015 11:44:21 GMT
Content-Type: application/json
Content-Length: 85
Cache-Control: no-cache

{"name":"guest","tags":"administrator","auth_backend":"rabbit_auth_backend_int
```

Check service status for MySQL service in VIM:

```
root@vim:/# service mysql status
mysql start/running, process 1069
```

As OpenStack is mostly developed in Python programming language, most of the services are executed as an interpreted python scripts. The best way to verify these script services are run is by listing python related processes by executing **pgrep <-options> python** command:

```
root@vim:/etc/mysql# pgrep -fl python
1261 /usr/bin/python /usr/bin/watchmen-history-api-server
6014 /usr/bin/python /usr/bin/heat-engine
6028 /usr/bin/python /usr/bin/heat-api-cloudwatch
6042 /usr/bin/python /usr/bin/heat-api-cfn
6056 /usr/bin/python /usr/bin/heat-api
6061 /usr/bin/python /usr/bin/ovft-engine --debug --config-file
   /etc/ovft/ovft.conf --log-file /var/log/ovft-engine.log
6066 /usr/bin/python /usr/bin/ovft-api --debug --config-file
   /etc/ovft/ovft.conf --log-file /var/log/ovft-api.log
```

Technologies Used

Protocols

- **ARP**: is a telecommunication protocol used for resolution of network layer addresses into link layer addresses, a critical function in multiple-access networks. ARP is used to convert an IP address to a physical address such as an Ethernet address (also known as a MAC address).
  URL: `https://tools.ietf.org/html/rfc826`
- **VLAN**: defines a system of tagging for Ethernet frames and the accompanying procedures to be used by bridges and switches in handling such as frames.
  URL: `http://www.ieee802.org/1/pages/802.1Q.html`
- **HTTP**: is an application protocol for distributed, collaborative, hypermedia information systems. Hypertext is structured text that uses logical links between nodes containing text. HTTP is the protocol to transfer hypertext.
  URL: `http://tools.ietf.org/html/rfc2616`
- **IP**: has the task of delivering packets from the source host to the destination host based on the IP address in the packet headers. IP defines packet structures that encapsulate the data to be delivered.
  URL: `https://www.ietf.org/rfc/rfc791.txt`
- **TCP**: provides a communication service by providing host to host connectivity at the transport layer of the internet model. The protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application.
  URL: `https://www.ietf.org/rfc/rfc793.txt`
- **UDP**: uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues and thus exposes any unreability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering or duplicate protection.
  URL: `https://www.ietf.org/rfc/rfc768.txt`
- **LDAP**: protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. A common usage of LDAP is to provide a single sign on where one password for a user is shared between many services.
  URL: `https://tools.ietf.org/html/rfc4511`

- **RADIUS**: networking protocol that provides Authentication, Authorization, and Accounting (AAA) for users who connect and use a network service. It is often used by Internet Service Provider (ISP) and enterprises to manage access to the internet or networks, wireless and integrated services.
  URL: `https://tools.ietf.org/html/rfc2865`

- **SSL**: cryptographic protocols designed to provide communications security over a computer network. They use X.509 certificates and asymmetric cryptography to authenticate the counterparty to negotiate a symmetric key.
  URL: `https://tools.ietf.org/html/rfc6101`

- **TLS**: ancestor of SSL.
  URL: `https://tools.ietf.org/html/rfc5246`

- **HTTPS**: communication protocol for secure communication over a computer network, with especially wide deployment on the internet. It is a result of layering the HTTP on top of SSL or TLS protocols. The main purpose is to provide authentication on a website and protect the privacy and integrity of exchanged data.
  URL: `http://tools.ietf.org/html/rfc2818`

- **ICMP**: used by network devices to send error messages for diagnostic, control purposes or in response to errors in IP operations.
  URL: `https://tools.ietf.org/html/rfc792`

- **DNS**: translates domain names, which can be easily memorized by humans, to the numerical IP addresses needed for the purpose of computer services and devices.
  URL: `https://www.ietf.org/rfc/rfc1035.txt`

- **IPv4**: fourth version in the development of the IP protocol.
  URL: `https://tools.ietf.org/html/rfc791`

- **IPv6**: sixth version in the development of the IP protocol.
  URL: `https://www.ietf.org/rfc/rfc2460.txt`

- **SSH**: cryptographic network protocol for initiating text-based shell sessions on remote machines in a secure way.
  URL: `https://tools.ietf.org/html/rfc4253`

- **IGMP**: communications protocol used by hosts and adjacent routers on IPv4 networks to establish multicast group memberships.
  URL: `https://tools.ietf.org/html/rfc2236`

- **FTP**: network protocol used to transfer computer files from one host to another over a TCP based network.
  URL: `https://www.ietf.org/rfc/rfc959.txt`

- **SOCKS**: internet protocol that routes network packets between a client and server through a proxy server. Additionaly provides authentication, so only authorized users may access a server.
  URL: `https://www.ietf.org/rfc/rfc1928.txt`

- **CRAM MD5 SASL**: challenge response authentication mechanism based on the HMAC-MD5 algorithm, using SASL as a simple authentication and security layer mechanism.
  URL: `https://tools.ietf.org/html/draft-ietf-sasl-crammd5-10`

- **MD5**: algorithm used widely as a cryptographic hash function for verifying data integrity.
  URL: `https://www.ietf.org/rfc/rfc1321.txt`

- **RPC**: inter-process communication that allows a computer program to cause a subrouting or procedure to execute in another address space.
  URL: `https://tools.ietf.org/html/rfc5531`

- **POP**: application layer internet standard protocol used by local email clients to retrieve email from a remote server over a TCP/IP connection.
  URL: `https://www.ietf.org/rfc/rfc1939.txt`

- **IMAP**: protocol for email retrieval and storage that allows multiple clients simultaneously connected to the same mailbox.
  URL: `https://tools.ietf.org/html/rfc3501`

- **VPN**: extends a private network across a public network, enabling a computer or network enabled device to send and receive data across shared or public networks as if it were directly connected to the private network by establishing a virtual point to point connection through the use of dedicated connections, virtual tunneling protocols or traffic encryption.
  URL: `https://tools.ietf.org/html/rfc4026`

- **ISAKMP**: establishes security associations and cryptographic keys in an internet environment. It provides a framework for authentication and key exchange and is designed to be key exchange independent.
  URL: `https://www.ietf.org/rfc/rfc2408.txt`

- **LLC**: upper sublayer of the data link layer. Provides multiplexing mechanisms that make it possible for several network protocols to coexist within a multipoint network and to be transported over the same network medium. It also provides a flow control and automatic repeat request error management mechanisms.
  URL: `http://www.networksorcery.com/enp/rfc/rfc1042.txt`

- **NTP**: networking protocol for clock synchronization between computer systems over packet siwtched, variable latency data networks.
  URL: `https://www.ietf.org/rfc/rfc5905.txt`

- **PPP**: data link protocol used to establish a direct connection between two nodes. It provides connection authentication, transmission encryption and compression.
  URL: `https://www.ietf.org/rfc/rfc1661.txt`

- **NetBIOS**: provides services related to the session layer of the OSI model allowing applications on separate computers to communicate over a local area network.
  URL: `https://tools.ietf.org/html/rfc1002`

- **RIP**: is a distance vector routing protocol which employs the hop count as a routing metric. RIP prevents routing loops by implementing a limit on the number of hops allowed in a path from the source to a destination.
  URL: `https://tools.ietf.org/html/rfc2453`
- **SNAP**: extension of the IEEE 802.2 logical link control to distinguish much more protocols of the higher layer than using of the 8 bit service access point fields present in the IEEE 802.2 header.
  URL: `https://tools.ietf.org/html/rfc1103`
- **STP**: network protocol that ensures a loop free topology for any bridged Ethernet local area network. The basic function of STP is to prevent bridge loops and broadcast radiation that results from them.
  URL: `https://tools.ietf.org/html/rfc4318`
- **Ethernet**: family of computer networking technologies for local area networks and metropolitan area networks.
  URL: `https://tools.ietf.org/html/rfc894`
- **802.1x**: IEEE standard for port based network access control. It provides authentication mechanism to devices wishing to attach to a LAN or WLAN. It also defines the encapsulation of the Extensible Authentication Protocol over IEEE 802.
  URL: `https://tools.ietf.org/html/rfc3580`
- **NAT**: methodology of remapping one IP address space into another by modifying network address information in IP datagram packet headers while they are in transit across a traffic routing device.
  URL: `https://tools.ietf.org/html/rfc2663`
- **SFTP**: command line interface client program to transfer files using the SSH file transfer protocol, which runs inside the encrypted secure shell connection.
  URL: `https://tools.ietf.org/html/rfc913`

Software

- **AppArmor**: is an effective and easy-to-use Linux application security system. AppArmor proactively protects the operating system and applications from external or internal threats, even zero-day attacks, by enforcing good behavior and preventing even unknown application flaws from being exploited. AppArmor security policies completely define what system resources individual applications can access, and with what privileges.
  URL: `http://wiki.apparmor.net/index.php/Main_Page`
- **SELinux**: is an implementation of mandatory access controls (MAC) on Linux. Mandatory access controls allow an administrator of a system to define how applications and users can access different resources such as files, devices, networks and inter-process communication.
  URL: `http://selinuxproject.org/page/Main_Page`

- **AD**: directory service developed by Microsoft for Windows domain networks.
  URL: `https://msdn.microsoft.com/en-us/library/bb742424.aspx`
- **Puppet**: open source configuration management utility which includes its own declarative language to describe system configuration.
  URL: `https://puppetlabs.com/`
- **OVFT**: tool for manage Open Virtualization Format; an open standard for packaging and distributing virtual appliances.
  URL: `http://www.dmtf.org/standards/ovf`
- **Heat**: service to orchestrate multiple composite cloud applications using templates.
  URL: `http://docs.openstack.org/developer/heat/`
- **RabbitMQ**: open source message broker software that implements the Advanced Message Queuing Protocol (AMQP).
  URL: `https://www.rabbitmq.com/`
- **Linux**: Unix-like and mostly POSIX compliant computer operating system assembled under the model of free and open source software development and distribution. The defining component of Linux is the Linux kernel.
  URL: `https://www.linux.com/`
- **BSD**: Unix operating system derivative developed and distributed from University of California, Berkeley.
  URL: `http://www.bsd.org/`
- **UNIX**: family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, developed in the 1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie and others.
  URL: `http://www.unix.org/`
- **OpenSSL**: open source implementation of the SSL and TLS protocols.
  URL: `https://www.openssl.org/`
- **OpenSSH**: OpenBSD Secure Shell, is a suite of security related network level utilities based on the SSH protocol, which help to secure network communications via the encryption of network traffic over multiple authentication methods and providing secure tunneling capabilities.
  URL: `http://www.openssh.com/`
- **Openstack**: free and open source cloud computing software platform. It consists of a series of interrelated projects that control pools of processing, storage and networking resources throughout a data center, which users manage through a web based dashboard, command line tools or RESTful API.
  URL: `https://www.openstack.org/`
- **PAM**: is a mechanism to integrate multiple low level authentication schemes into a high level API, allowing programs that rely on authentication to be written independently of the underlying authentication scheme.
  URL: `http://www.linux-pam.org/`
- **GNU**: Unix like computer operating system composed wholly of free software.
  URL: `https://www.gnu.org/`

Licenses

- **Apache license**: free software license written by the Apache Software Foundation. Apache license requires preservation of the copyright notice and disclaimer. This license allows the user of the software the freedom to use the software for any purpose, to distribute and modify it, without concern for royalties.
  URL: `https://www.apache.org/licenses/LICENSE-2.0.html`
- **GPL**: free software license which guarantees end users the freedoms to use, study, share and modify the software.
  URL: `https://www.gnu.org/copyleft/gpl.html`
- **GFDL**: copyleft license for free documentation. It is similar to the GPL, giving readers the rights to copy, redistribute and modify a work and requires all copies and derivatives to be available under the same license.
  URL: `https://www.gnu.org/copyleft/fdl.html`

Other Terms

- **ACL**: is a list of permissions attached to an object. ACL specifies which users or system processes are granted access to objects as wel as what operations are allowed on given objects.
  URL: `https://www.ietf.org/rfc/rfc2086.txt`
- **TPM**: international standard for a secure crypto processor, which is a dedicated microprocessor designed to secure hardware by integrating cryptographic keys into devices.
  URL: `http://www.trustedcomputinggroup.org/resources/trusted_platform_module_tpm_summary`
- **HSM**: physical device that safeguards and manages digital keys for strong authentication and provides cryptoprocessing.
  URL: `https://www.pcisecuritystandards.org/documents/PCI%20HSM%20Security%20Requirements%20v1.0%20final.pdf`
- **LaTeX**: document preparation system and document markup language widely used for the communication and publication of scientific documents in different fields.
  URL: `http://www.latex-project.org/`
- **SQL**: special purpose programming language designed for managing data held in a relation database management system or for stream processing in a relational data stream management system.
  URL: `https://tools.ietf.org/html/rfc6922`

- **ECS**: The Ericsson Cloud System which refers to the combination of different cloud technologies for deliverable Ericsson product.
  URL: http://www.ericsson.com/ourportfolio/products/cloud-system
- **CEE**: The Cloud Execution Environment, which refers a part of the ECS that manages a region, which includes a data center with controllers, compute nodes, vim and Fuel.
  URL: http://www.ericsson.com/ourportfolio/products/cloud-execution-e nav=productcategory008|fgb_101_0537
- **HA**: characteristic of a system that defines the up time over the total time determining tolerable down time.
  URL: http://en.wikipedia.org/wiki/High_availability
- **OpenID**: open standard and decentralized protocol by the non-profit OpenID Foundation that allows users to be authenticated by certain cooperating sites using a third party service.
  URL: http://openid.net/
- **OAuth**: open standard for authorization. It provides client applications a secure delegated access to server resources on behalf of a resource owner. Also specifies a process for resource owners to authorized third party access to their server resources without sharing their credentials.
  URL: http://oauth.net/
- **OWASP**: online community dedicated to web application security by creating free available articles, methodologies, documentation, tools and technologies.
  URL: https://www.owasp.org/index.php/Main_Page
- **IMPI**: multi-fabric message passing library that implements the Message Passing Interface specification.
  URL: http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html
- **MAC**: unique identifier assigned to network interfaces for communications on the physical network segment. MAC addresses are used as a network address for most IEEE 802 network technologies which includes Ethernet and WiFi.
  URL: https://tools.ietf.org/html/rfc3422
- **XML**: markup language that defines a set of rules for encoding documents in a format which is both human readable and machine readable.
  URL: http://www.w3.org/XML/
- **NAC**: approach to computer network security that attempts to unify endpoint security technology, user or system authentication and network security enforcement.
  URL: http://www.ieee802.org/1/pages/802.1x-rev.html
- **IdAM**: describes the management of individual principals, their authentication, authorization and privileges across system and enterprise boundaries with the goal of increasing security.
  URL: http://ciog6.army.mil/Portals/1/Architecture/2014/20140929-US_Army_Identity_and_Access_Management_Reference_Architecture_V4-0.pdf

- **PCI DSS**: proprietary information security standard for organizations that handle branded credit cards for the major card schemes.

- **VIC**: Virtual Machine which act as a cloud controller.
  URL: Ericsson Internal.
  URL: `https://www.pcisecuritystandards.org/security_standards/`

- **VIM**: Orchestration node from Ericsson as a part of the ECS solution which manages interfaces and the virtual infrastructure.
  URL: Ericsson Internal.