Developing Customer Edge Switching Test Framework

Erkki Laite

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 18.04.2017

Thesis supervisors:

Prof. Raimo Kantola

Thesis advisor:

M.Sc. Hammad Kabir

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Erkki Laite

Title: Developing Customer Edge Switching Test Framework

Date: 18.04.2017 Language: English Number of pages: 10+75

Department of Communications and Networking

Professorship: Communications Engineering

Supervisor: Prof. Raimo Kantola

Advisor: M.Sc. Hammad Kabir

Customer Edge Switching (CES) and Realm Gateway (RGW) are technologies designed to solve core challenges of the modern Internet. Challenges include the ever increasing amount of devices connected to the Internet and risks created by malicious parties. CES and RGW leverage existing technologies like Domain Name System (DNS).
Software testing is critical for ensuring correctness of software. It aims to ensure that products and protocols operate correctly. Testing also aims to find any critical vulnerabilities in the products. Fuzz testing is a field of software testing allowing automatic iteration of unexpected inputs.
In this thesis work we evaluate two CES versions in performance, in susceptibility of Denial of Service (DoS) and in weaknesses related to use of DNS. Performance is an important metric for switches. Denial of Service is a very common attack vector and use of DNS in new ways requires critical evaluation.
The performance of the old version was sufficient. Some clear issues were found. The version was vulnerable against DoS. Oversights in DNS operation were found. The new version shows improvement over the old one.
We also evaluated suitability of expanding Robot Framework for fuzz testing Customer Edge Traversal Protocol (CETP). We conclude that the use of the Framework was not the best approach.
We also developed a new testing framework using Robot Framework for the new version of CES.

Keywords: CES, CETP, PRGW, Fuzzification, testing, Robot Framework, security

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Erkki Laite

Työn nimi: Asiakasreunakytkennän testausalustan kehitys

Päivämäärä: 18.04.2017 Kieli: Englanti Sivumäärä: 10+75

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka

Työn valvoja: Prof. Raimo Kantola

Työn ohjaaja: DI Hammad Kabir

Customer Edge Switching (CES) asiakasreunakytkentä ja Realm Gateway (RGW) alueen yhdyskäytävä tarjoavat ratkaisuja modernin Internetin ydinongelmiin. Ydinongelmiin kuuluvat kytkettyjen laitteiden määrän jatkuva kasvu ja pahantahtoisten tahojen luomat riskit. CES ja RGW hyödyntävät olemassa olevia tekniikoita kuten nimipalvelua (DNS).
Ohjelmistojen oikeellisuuden varmistuksessa testaus on välttämätöntä. Sen tavoitteena on varmistaa tuotteiden ja protokollien oikea toiminnallisuus. Testaus myös yrittää löytää kriittiset haavoittuvuudet ohjelmistoissa. Sumea testaus on ohjelmistotestauksen alue, joka mahdollistaa odottamattomien syötteiden automaattisen läpikäynnin.
Tässä työssä arvioimme kahden CES version suorituskykyä, palvelunestohyökkäyksien sietoa ja nimipalvelun käyttöön liittyviä heikkouksia. Suorituskyky on tärkeä mittari kytkimille. Palvelunesto on erittäin yleinen hyökkäystapa ja nimipalvelun uudenlainen käyttö vaatii kriittistä arviointia.
Vanhan version suorituskyky oli riittävä. Joitain selviä ongelmia löydettiin. Versio oli haavoittuvainen palvelunestohyökkäyksille. Löysimme epätarkkuuksia nimipalveluiden toiminnassa. Uusi versio vaikuttaa paremmalta kuin vanha versio.
Arvioimme työssä myös Robot Framework testausalustan laajentamisen soveltuvuutta Customer Edge Traversal Protocol (CETP) asiakasreunalävistysprotokollan sumeaan testaukseen. Toteamme, ettei alustan käyttö ollut paras lähestymistapa. Esitämme myös työmme Robot Framework alustaa hyödyntävän testausalustan kehityksessä nykyiselle CES versiolle.
Kehitimme myös uuden testausalustan uudelle CES versiolle hyödyntäen Robot Frameworkia.

Avainsanat: CES, CETP, PRGW, sumea testaus, testaus, Robot Framework, tietoturva

Preface

I want to thank Professor Raimo Kantola and my instructor Hammad Kabir for their guidance and help with writing the thesis. I would also like to thank Jesus Llorente for help with understanding the development of the new version and environment of the CES and for comments on technical approach of the new development.

I am also thankful for all the support and encouragement I received from my parents

Otaniemi, 18.04.2017

Erkki Laite

Contents

Abstract ii

Abstract (in Finnish) iii

Preface iv

Contents v

List of Figures viii

List of Tables ix

List of Listings ix

Abbreviations x

1 Introduction 1

2 Internet protocol suite, common protocols and related technologies 4
- 2.1 Internet protocol suite . . . 4
- 2.2 Internet protocol version 4 . . . 5
  - 2.2.1 IPv4 Header . . . 5
- 2.3 Classless Internet Domain Routing and Subnetting . . . 7
  - 2.3.1 IP address allocation . . . 8
- 2.4 Internet Protocol version 6 IPv6 . . . 9
- 2.5 Transport Control Protocol TCP . . . 9
  - 2.5.1 TCP Header . . . 9
  - 2.5.2 Operation . . . 10
- 2.6 User Datagram Protocol . . . 12
- 2.7 Internet Control Message Protocol ICMP . . . 12
- 2.8 Link layer, Ethernet, MAC addresses and ARP . . . 14
  - 2.8.1 Ethernet . . . 15
  - 2.8.2 Address Resolution Protocol . . . 15
- 2.9 Domain Name System . . . 17
  - 2.9.1 Domain Name Querying and Messages . . . 17
- 2.10 Application Layer protocols . . . 19
  - 2.10.1 Hyper Text Transport Protocol . . . 19
  - 2.10.2 File Transfer Protocol . . . 20

3 Network address translation, security, software and fuzz testing 22
- 3.1 Network Address Translation . . . 22
- 3.2 Inherent security weaknesses of IP networks . . . 23
- 3.3 Common attacks . . . 23
  - 3.3.1 Spoofing . . . 23
  - 3.3.2 Denial of Service . . . 23

3.3.3 Brute force and dictionary . . . 24
3.3.4 Buffer overflow and injections . . . 24
3.4 Encryption, encoding and hashing . . . 25
3.5 Software testing . . . 26
3.5.1 Fuzz testing . . . 26

4 Systems under test **28**
4.1 Test enviroment . . . 28
4.1.1 CES version 1 . . . 28
4.1.2 CES version 2 . . . 29
4.2 CES version 1 . . . 30
4.2.1 RGW . . . 30
4.2.2 CETP . . . 30
4.3 CES version 2 . . . 32

5 Tools **34**
5.1 Robot Framework . . . 34
5.1.1 Test case styles . . . 34
5.1.2 Variable types . . . 34
5.1.3 Files, tables and settings . . . 35
5.1.4 Libraries . . . 35
5.1.5 User keywords . . . 37
5.2 CETPLibrary . . . 38
5.2.1 Design decisions . . . 38
5.2.2 `CETPLibrary.py` . . . 38
5.2.3 CETP_lib.robot . . . 39
5.3 Software and tools . . . 40
5.3.1 Wireshark packet dissector and analyzer . . . 40
5.3.2 Commands: `time`, `watch` and `tail` . . . 41
5.3.3 Network bandwidth measurement: `iperf` . . . 42
5.3.4 Packet generator and port scanner: `hping3` . . . 42
5.3.5 DNS enumeration tools: `dnsrecon` and dnsenum . . . 43
5.3.6 packet generator: `sendip` . . . 43
5.3.7 File retrieval from web: `wget` . . . 44
5.3.8 FTP-client: `tnftp` . . . 44

6 General testing of CESv1 **46**
6.1 General . . . 46
6.2 Performance testing . . . 46
6.2.1 Methodology . . . 46
6.2.2 Results . . . 47
6.2.3 Evaluation . . . 48
6.3 Denial of Service . . . 48
6.3.1 Methodology . . . 48
6.3.2 Results . . . 48

6.3.3 Evaluation and suggestions 49
6.4 DNS attacks 50
6.4.1 Methodology 50
6.4.2 Results 50
6.4.3 Evaluation 51

7 Fuzzing CES version 1 and CETP version 2 52
7.1 General 52
7.2 Fuzz testing CETP version 2 52
7.2.1 Example of work: CETP header size field iterator 53
7.3 Found issues 54
7.3.1 Security handling 56
7.4 Evaluation issues 57
7.5 Evaluation of tools and methodologies 57

8 General testing of CESv2 and porting of RF tests 59
8.1 General 59
8.2 Performance testing 59
8.2.1 Methodology 59
8.2.2 Results 60
8.2.3 Evaluation 60
8.3 Denial of Service 61
8.4 DNS attacks 61
8.5 Porting of Robot Framework test cases 62
8.5.1 General structure 62
8.5.2 `Libraries` files 63
8.5.3 `Connectivitytest` test suites 63
8.5.4 `NATTest` test suites 63
8.5.5 `RGWTest` test suites 64
8.6 Future work 64

9 Conclusions 65

References 67

A Appendix: CETPLibrary documentation 70

B Appendix: CETP_lib.robot 73

C Appendix: Test case for RF CETP Security Handshake with header size-field iteration 75

List of Figures

2.1 IP version 4 header 6
2.2 TCP Header 10
2.3 TCP 3 way handshake 11
2.4 TCP transmission and retransmission 11
2.5 UDP Header 12
2.6 ICMP header 13
2.7 ICMP Time exceeded 13
2.8 ICMP Destination Unreachable 13
2.9 ICMP Protocol Redirect 14
2.10 ICMPl Echo Request and Reply 14
2.11 Example of ARP Request 16
2.12 Example of ARP Response 16
2.13 DNS header 18
2.14 FTP Connection establishment 20
4.1 CETP Header 31
4.2 CETP Control TLV 31
4.3 CETP Payload TLV for Ethernet encapsulation 32
4.4 CETP Payload TLV for IPv4/IPv6 encapsulation 32
5.1 Wireshark with CETP dissector plug-in 41
7.1 Use of `sendip` for random packet generation and Wireshark capture of process 53
7.2 Use of `sendip` for packet replay and Wireshark capture of results 55

List of Tables

2.1 Classfull Internet address 7
2.2 Subnet Mask, Network prefix and host part for `198.18.0.10` 8
2.3 Some IPv4 Special-Use Address Registry entries 8
4.1 List of containers run for testing. 28
4.2 List of lxc-containers and the bridges they are connected to and their addresses 29
6.1 Results of performance testing using `iperf` in Mbit/sec 47
6.2 Results of stress testing RGW using `hping3` 49
8.1 Results of performance testing using `iperf` in Mbit/sec 60

List of Listings

2.1 HTTP/1.1 communication 19
5.1 Excerpt from CETPLibrary.py 36
5.2 User created keyword in Robot Framework, part of global_lib.robot . 37
5.3 Use case of: `iperf` server and client 42
5.4 Use case of: `hping3` tool 42
5.5 Use case of: `dnsrecon` tool 43
5.6 Use cases of: `sendip` tool 44
5.7 Use case of: `wget` 44
5.8 Example of FTP test 45
7.1 Dissection of CETP message causing KeyError 54
7.2 Truncated part of CETP log message in case of KeyError 55
7.3 CETP details for packet containing security requirements and incorrect header length 56
7.4 Output from CES when accepting CES security requirements 56

Abbreviations

ALG	Application Layer Gateway
ARP	Address Resolution Protocol
CES	Customer Edge Switching
CETP	Customer Edge Traversal Protocol
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DST	Destination Session Tag
FTP	File Transfer Protocol
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICE	Interactive Connectivity Establishment
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISP	Internet Service Provider
MAC	Media Access Control
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PKI	Public Key Infrastructure
PRGW	Private Realm Gateway
RFC	Request For Comments
RGW	Realm Gateway
RPC	Remote Procedure Call
SCP	Secure Copy
SHA	Secure Hash Algorithm
SSH	Secure Shell
SST	Source Session Tag
RLOC	Resource LOCation
STUN	Session Traversal Utilities for NAT
TCP	Transport Control Protocol
TLV	Type Length Value
ToS	Type of Service
TTL	Time To Live
TURN	Traversal Using Relays around NAT
UDP	Universal Datagram Protocol
VM	Virtual Machine
XML	eXtensible Markup Language

1 Introduction

Internet was designed with the assumption that communication will take place between trusted hosts. As it has grown in use and size beyond original design goals, this basis of trust has come under attack by malicious entities. The limits of original routing scheme have been reached and better solutions for connecting all of the users around the globe are needed. The current solutions are less than ideal and fresh look for alternatives is needed.

End-to-end principle, where all hosts on Internet are reachable with uniquely identifying IP (Internet Protocol)-address, has been forgone. Due to limitations of IP version 4 (IPv4) the most widely used communication protocol, it is not possible any more to achieve end-to-end for all users.

Internet is a network of interconnected networks. Boundaries between these networks are edges and processing can and do happen at them. It can enable hosts inside different networks to communicate with each other despite not being able to directly address each other. Network Address Translation is a popular solution, but with some undesirable limitations. One of such limitations is that hosts are hard to connect to from a different network.

World has become reliant on the Internet for many services such as finances, communications and information transmission and storage. Large scale development and use of Internet of Things (IoT) is also predicted adding computing and communication to many new devices. In the world, where Internet permeates everything and is used for many aspects of life, risks towards it and its uses are critical for operation of the economy and even public safety. The increased focus should be put to securing the communication and devices, and preventing any potential attacks. Currently popular tools include firewalls capable of filtering traffic and Intrusion Detection Systems (IDS) for more advanced traffic monitoring and intrusion detection.

Preventing attacks is not enough as communicated information itself has value. Securing the communication from leaking to unauthorized parties is also one requirement for any modern system. This is often achieved with cryptography where information is mathematically secured so that only those with relevant knowledge can understand it. Authentication of the party being communicated with is also attained with the use of cryptography. Commonly these goals are reached with Public Key Infrastructure (PKI) were trusted parties validate information provided by one or both parties communicating.

Domain Name System is a core part of the modern Internet, where simple human readable names are used for connecting to services instead of hard to remember numeric or alphanumeric addresses. This provides many benefits such as allowing changing of the IP addresses of the hosts.

Customer Edge Switching (CES) is a system that aims to bring all of the aspects presented above together in a new system. CES system currently consist of CES used to facilitate communication between hosts behind different CES instances and Private Realm Gateway (PRGW) for allowing communication to and from hosts not behind another CES. DNS is leveraged as existing technology to decouple the hosts from traditional addresses and for allowing multiple hosts to share a small pool of

addresses. This approach should allow better access to private hosts from Internet hosts than current NATs do. In communication between CES instances the sides confirm each other's identity using Customer Edge Travelsal Protocol (CETP), which can also be used to secure the communication. CES also offers policies that can be defined as requirements for accepting communication from other CES instances. This makes it possible for instances to negotiate with each other based on policies stored in them to attain trust. RGW protects the hosts behind it by only allowing traffic when a private host is queried for and can provide Application Layer Gateways (ALG) to help communication with protocols with special requirements.

Testing is needed and commonly used to finding defects in software products. Automated testing is beneficial for quick iteration of software development. This can be done on various levels from testing individual pieces of code to testing that whole product performs as expected and behaviour haven't changed from previous versions. Effective regression testing can be achieved with the use of testing frameworks. In this thesis Robot Framework, a high level expandable keyword based testing environment was used. Testing also expands to the field of security. Aim of security is to prevent attacker from gaining access to the system or information in the system, or preventing the attacker from harming the system.

Research Problem

Initially the primary goal was set as introducing fuzz testing using Robot Framework and carry on developing the test framework for CES, by improving coverage and portability. It was also considered valid to get fresh perspective of general security of the system. Fuzz testing was considered to be interesting avenue for testing CES as software product and it was not utilized previously in testing of CES.

Later as a new version of RGW was released, the focus of the work changed to moving existing tests where possible to the new system and checking for correct operation.

Research Objectives

One of the key goals of the work was to evaluate Robot Framework in fuzz testing of CES. For this various existing libraries available were evaluated. As these were found lacking due to nature of the protocol used in CES, custom extension for Robot Framework was seen as the solution.

In addition well known tools were evaluated for testing the general security of CES. Focus in part on DNS as key technology of CES was chosen. In addition to this operation and performance of multiple protocols and systems survivability under load were chosen as additional metrics for research.

Due to new version of software not having the CES features available, work was focused on reintroducing existing tests for testing new version of RGW and to evaluating operation of protocols and their performance on it. Existing test suites were found hard to port to the new version of RGW. Effort therefore was devoted to

building new test suites and building blocks, for testing the new version of RGW, from ground up using best practises.

Structure

The structure of thesis is the following: In Chapter 2 we cover Internet protocol suite and describe protocols in it to build a base of understanding. In the chapter we also expand on other relevant technologies and protocols for the thesis. Chapter 3 discusses Network Address Translation (NAT), gives overview of security and related technologies, and covers software and fuzz testing in relation of this work. Chapter 4 describe our test environment and the systems under test. Chapter 5 gives more technical details of the tools used and developed. Chapters six to eight cover the work done with Chapter 6 focusing on testing CES version 1, Chapter 7 describing application and work done using fuzz testing on CES version 1 and Chapter 8 focusing on work done on CES version 2 and new test suites build. Finally in Chapter 9 we provide the conclusions. Appended are CETPLibrary and its resource file and example of its use.

2 Internet protocol suite, common protocols and related technologies

First in this chapter we cover the Internet Protocol suite standard, which defines the model and standard protocols of the Internet. We describe the 4-layer model of the suite. Next we present the description of protocols defined in the suite: Internet Protocol version 4 (IPv4) with Classless Internet Domain Routing and subnetting and IP address allocation, Transport Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP). After these protocols we provide some relevant information of link-layer namely Ethernet and Medium Access Control (MAC) addressing. We also present some aspects of DNS and conclude the chapter with Hyper Text Transfer Protocol (HTTP), File Transfer Protocol (FTP).

2.1 Internet protocol suite

Communication between and inside computer networks is defined in the framework of Internet protocol suite, later in the chapter referred to as the suite, defined in Request for Comments(RFC) 1122.[1] This model consist of 4 layers namely link, internet, transport and application. There are other popular models such as Open Systems Interconnection model (OSI model) that consists of 7 layers physical, datalink, network, transport, session, presentation and application layers.[2] We however will focus on the Internet protocol suite which some other models expand with 5th physical layer.

Link layer

Link layer in the suite is for hosts to communicate over a network they are directly connected to and it must implement protocols used for this interfacing. The suite also describes encapsulation between link and Internet layer and Address Resolution protocol (ARP), both presented in the Section 2.8.[1] The link layer transfers frames over a network. A frame is a finite sequence of bits or octets. In a packet network a frame has delimiters for the beginning and the ending of the frame. The header of the frame carries the link layer address of the sender and the receiver.

Network layer

Network layer is also known as Internet Layer in the suite. In the suite Internet Protocol (IP) is used to carry data between hosts across a number of link layer networks. All end to end transport layer protocols use it. IP itself is connectionless and provides no guarantees of delivery, making layers above responsible for reliable delivery. ICMP presented in Section 2.7 is considered to be Internet layer protocol in the suite even if it does use IP for carrying data.[1]

We will describe IP version 4 (IPv4) in detail in Section 2.2. There are many benefits with the new version 6 and critical shortcomings of the older version 4, which will be discussed in Section 2.4.

Transport layer

Transport layer provides a channel of end-to-end communication for the application layer. Two of the original protocols in the suite are TCP and UDP. Transport Control Protocol (TCP) is a reliable connection oriented protocol, meaning that the protocol guarantees that data will be delivered to the other party and there being a permanent channel formed over which communications happens somewhat analogous to a phone call. User Datagram Protocol is connectionless ("datagram") transport service. This means that each message is an individual block of data and delivery is not guaranteed. A common feature of both TCP and UDP is that they allow identifying the application on both sending and the receiving host. This is done using 16 bit source and destination port numbers in the protocol headers. These protocols are also discussed in detail later in the chapter.[1]

Application layer

Application layer is the top layer of the suite. The application layer provides services to processes for seamless communication. Examples of application layer protocols are Hyper Text Transfer Protocol (HTTP) designed for browsing websites, File Transfer Protocol (FTP) for transferring files between hosts and Secure Shell (SSH) for encrypted remote connections. In modern web-technology HTTP protocol is used to provide a multitude of services such as video streaming and even operating programs on the Internet. All user facing protocols communicating over Internet utilize one or another application layer protocol.

2.2 Internet protocol version 4

Network layer protocol of the suite is the Internet Protocol. Two versions are in use version 4 (IPv4) supported by all devices connected to Internet directly and new version 6 (IPv6) which is designed to replace IPv4. IPv4 is first described in RFC 791 dating back to 1981 which replaced earlier definition of RFC 760 from 1980.[3][4] Current standard for IPv6 can be found in RFC 2460 from 2006. We will also present some aspects in Section 2.4.[5]

Every IPv4 packet has a header presented in Figure 2.1, containing the various fields including source and destination addresses. These fields make up the minimum length of IPv4 packet of 20 octets. IPv4 addresses were designed to be globally routable meaning that messages can travel from any publicly routed address to any other. IPv4 uses 32 bit addresses providing total address space of over 4 billion unique addresses.[3] IPv4 addresses are commonly written as `192.168.254.101` with each of dot separated number corresponding to 1 byte of the address.

2.2.1 IPv4 Header

The header is presented in Figure 2.1. First 4 bit `version` field defines the version of protocol, 4 for IPv4. In same octet `Internet Header Length` (`IHL`) field contains

number of 4 octet groups in the header in a 4 bit value. Maximum size of the header is 15 such groups or 60 octets, with smallest possible size being 20 octets.

The following octet contains 6 bit long Differentiated Services Code Point (`DSCP`) and 2 bit long Explicit Congestion Notification (`ECN`) fields. `DSCP` was originally reserved for Type of Service, but was later redefined for use with Differentiated services such as Voice over IP, which would allow providing different classes of transport services based on the type of traffic.[6] `ECN` allows end-to-end notifications of congestion.[7]

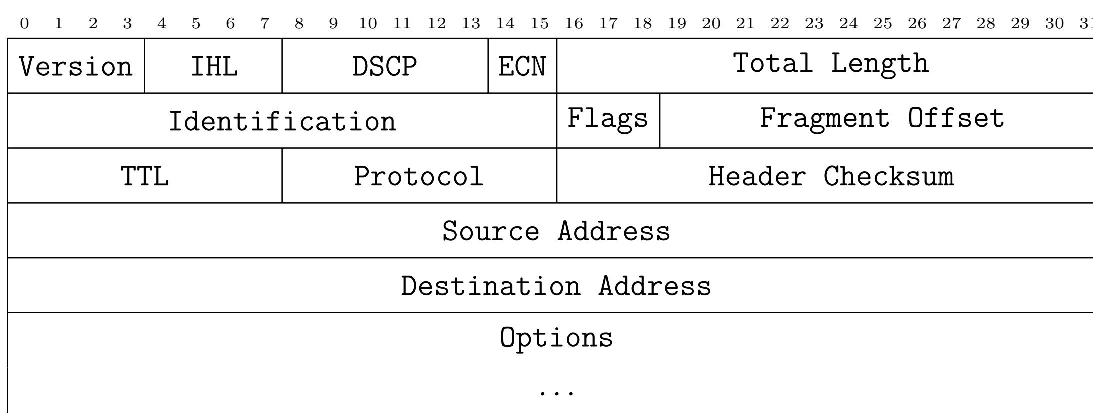

Figure 2.1: IP version 4 header

The 16 bit `Total Length` field stores the length of the datagram in octets limiting the maximum length to 65535 octets. The protocol requires support for minimum of 576 octet long datagrams, but in practice the datagrams are larger than the minimum. The upper limit, however, is a theoretical limit that is rarely used as lower layers limit it.[8]. Too large datagrams can require fragmentation which is a practise where a datagram is broken into multiple parts and reassembled at either another router or receiving host. The 16 bit `Identification` field is primarily used for tracking fragments of datagrams mentioned above. In the past it has been used for spoofing detection, but this practise has been prohibited[9].

Of the following two octets 3 bits are reserved for flags with the first bit being reserved, second bit Don't Fragment (`DF`) and third one More Fragments (`MF`). When `DF` is set, the datagram will be dropped if transmitting it would require fragmentation. This feature can be used for Maximum Transmission Unit (MTU) discovery where datagrams increasing in size are sent to discover the maximum size of the datagram that can be transmitted over the path without fragmentation. `MF` flag is used with fragmented datagrams, where it is set for all the datagrams apart from the last one, which also has non-zero fragment offset field. Fragment offset field is stored in the following 13 bits. Offset value is the number of 8 octet groups from the beginning of the datagram, such that the first fragment has value of 0 and the value of 185 would correspond to 1481th octet of the datagram.

`Time To Live` (`TTL`) is an 8 bit field designed to prevent datagrams from trav-

elling infinitely on Internet. The value is specified in seconds but in practice it is the number of hops on the route, where each router decrements it by one. When it reaches zero, the datagram is dropped and `ICMP Time Exceeded` message is sent to source address. The process is described in more detail in Section 2.7.

The 8 bit `Protocol` field stores the protocol of the carried datagram or packet, examples of protocols and their numbers are 6 for TCP, 7 for UDP and 254 for "Use in experimentation and testing".[10] Next 16 bits contain the `Header checksum` used for error checking of the header. Each time a router receives a datagram, it calculates the checksum and compares it to the one stored in the datagram to discover any errors. If checksums do not match, the datagram is dropped. The 32-bit `Source` and `Destination addresses` are next. Addresses are followed by potential optional headers padded to 32 bits.

2.3 Classless Internet Domain Routing and Subnetting

Subnetworking is the practise of dividing an IP address to network prefix and host number, and treating all addresses sharing a network prefix as a single subnetwork. Assigning addresses from a subnetwork to all hosts in the subnetwork is preferred as it allows aggregation in routing and allows the holder of the network to sub-divide its address blocks more freely. Additionally there is the penalty of first and last addresses of the network have special meanings of identifying network and being broadcast addresses for the network respectively. Therefore they cannot be used for identifying hosts.

Class	Prefix length bits	Number of networks	Addresses in network	Address Range
A	8	128	16777216	`0.0.0.0 - 127.255.255.255`
B	16	16384	65536	`128.0.0.0 - 191.255.255.255`
C	24	2097152	256	`192.0.0.0 - 223.255.255.255`

Table 2.1: Classfull Internet address

In early days IP address ranges were divided in Classes A, B, C, D and E for purposes of Classfull routing. The sizes of A, B and C can be seen in Table 2.1. There is a very limited amount of Class A networks with massive 16,7 million addresses, at the time too large to be fully utilized even by the biggest corporations. However, the smallest Class C's 256 addresses are too few for most administrated networks.

Subnetworking is represented with sub-network mask, with one way being an IP address of all ones. An example can be found in Table 2.2 where the address `192.168.254.112` and its subnet mask `255.255.255.0` is shown. All hosts sharing the same subnet mask and network prefix are in the same subnetwork. There can be multiple separate subnetworks with same addresses in different private networks. Being in the same network allows hosts to use the last address of the network as broadcast address to send packets to all hosts in the network. `192.168.254.255`

	Binary	Decimal
Address	11000000.10101000.11111110 .01110000	192.168.254.112
Mask	11111111.11111111.11111111 .00000000	255.255.255.0
Network prefix	11000000.10101000.11111110 .00000000	192.168.254.0
Broadcast	11000000.10101000.11111110 .11111111	192.168.254.255
Host part	00000000.00000000.00000000 .00001010	0.0.0.112

Table 2.2: Subnet Mask, Network prefix and host part for 198.18.0.10

would be such address for a network with a network mask of 255.255.255.0.

Classless Internet Domain Routing (CIDR) was adopted to allow more fine grained division of networks. It is defined in RFC 4632 Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan[11]. In CIDR network prefixes are divided by single bits instead of octets of classfull routing. For example in the 10.0.0.0/8, a Class A private network, the address is the first address of the subnetwork and number after slash is the length of the subnet mask. Our example in Table 2.2 would have 192.168.254.0/24 for network or alternatively denoting that subnetwork in 192.168.254.112/24.

2.3.1 IP address allocation

IP addresses are allocated in ranges by Internet Assigned Numbers Authority (IANA) to regional Internet registries (RIR) who further allocate them to organisations such as corporations, public and private organisations and service providers. Certain ranges are reserved for special use with relevant for this work listed in Table 2.3. Notable in the list are the Private-Use ranges, which are networks that are not globally routed and are free for use in private networks. In addition 198.18.0.0/15 is reserved for "Network Interconnect Device Benchmark Testing" which is also an non-globally routed network and as such used as additional network in test environment for CES version 1.[12] Range 100.64.0.0/10 allocated for Carrier-grade NAT or Large Scale NAT is used in the environment for CES version 2.[13]

0.0.0.0/8	"This host on this network"
10.0.0.0/8	Private-Use
100.64.0.0/10	Carrier-grade NAT
127.0.0.0/8	Loopback
172.16.0.0/12	Private-Use
192.168.0.0/16	Private-Use
198.18.0.0/15	Benchmarking
240.0.0.0/4	Reserved
255.255.255.255/32	Limited Broadcast

Table 2.3: Some IPv4 Special-Use Address Registry entries

2.4 Internet Protocol version 6 IPv6

IP version 6 defined in RFC 2460 is designed to replace IPv4.[5] The main difference between protocols is the address length which has been increased to 128 bits in IPv6. IPv6 allows $3.4{*}10^{38}$ addresses compared to IPv4's $4{*}10^{9}$. The change means that the protocol is unlikely to run out of usable addresses. IPv6 also simplifies header processing by not having the checksum recalculated on each hop. Also IPv6 does not allow fragmentation on the route. Fragmentation is either avoided entirely, performed by sender or end points find MTU using MTU discovery[14][15].

IPv6 also allows taking advantage of the large address space by using sparse allocation. By leaving space between allocations it is possible at later time to expand the sub-networks while keeping the same prefix. This is not possible with IPv4 as new allocations are small and from different blocks. Due to small allocations IPv4 addresses space is fragmented and sub-networks owned by same entities do not form continues blocks.

Routing on Internet often combines multiple smaller sub-networks together in single routes for purposes of forwarding packets to next hop if possible. Fragmentation of the address space results in sub-networks being smaller and more numerous compared to optimal solution leading to higher memory and processing requirements in routers.

2.5 Transport Control Protocol TCP

Transport Control Protocol (TCP) is widely used connection-oriented data streaming protocol that offers reliable ordered and error checked service to protocols using it. Many protocols such as HTTP and FTP use TCP as transport layer protocol. The protocol is defined in RFC 793.[16]

2.5.1 TCP Header

TCP header consist of 20 octets with potential for further information known as options. First 4 octets contain two 2 16-bit fields: `source` and `destination ports`, which allows the maximum of 65536 port numbers. These port numbers together with `source` and `destination IP address` in the IP-header identify the connection of the packet.

Port numbers are followed by 32-bit Sequence- and Acknowledgement numbers. These are used to achieve ordered and reliable service. Next is 16-bits consisting of 4-bit offset for header length in 32-bit words, 3 reserved bits, and 9 flag bits. `NS`, `CWR`, `ECE`, `UGR`, `ACK`, `PSH`, `RST`, `SYN` and `FIN`.

`NS`, `CWR` and `ECE` flags are used for various aspects of Explicit Congestion Notification technique, which allows a way of communicating congestion between the end points thus preventing need of dropping packets. These were added to protocol in RFC 3168.[7] `URG` indicates urgent pointer field being significant. `ACK` for significance of acknowledgement field. `PSH` is for pushing buffered data to application. `RST` for resetting connection. `SYN` for synchronisation of sequence numbers with first packets. Some other flags also depend on `SYN`. `FIN` is for ending connection.

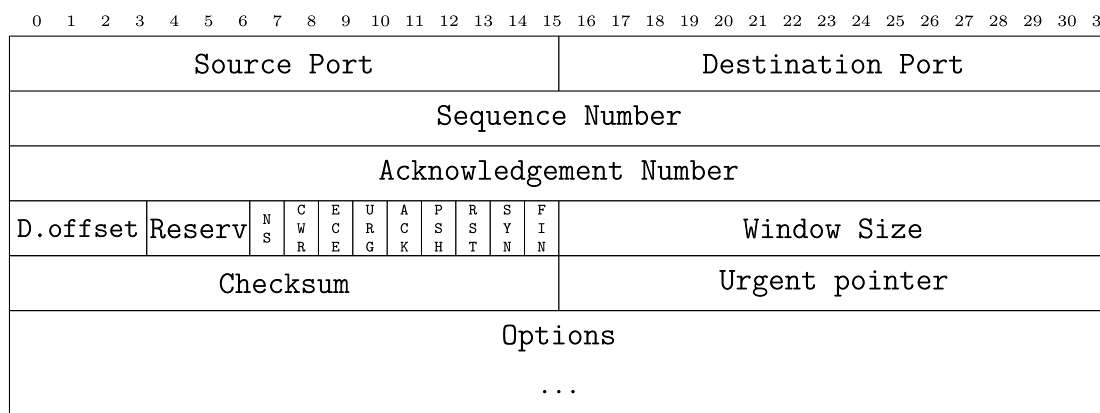

Figure 2.2: TCP Header

16-bit `window size` field indicates the number of units the sender is willing to receive from current acknowledgement value. Followed by 16-bit error checking `checksum` of header and data. Protocol ends with 0 to 40 octets of `options` coded as type-length-value of 1 octet type, 1 octet length and variable length data. Header is padded to 32-bit boundary.

Ports 0-1023 are System Ports, 1023-49151 User Ports and 49152-65535 Dynamic and/or Private Ports. Both System Ports and User Ports are assigned by IANA. Dynamic ports are never assigned.[17] Few ports such as 0, 1021, 1022, 1023 and 1024 are reserved, of these 0 is commonly used as a wish from a program to the operating system for assigning a port, 1021 and 1022 are reserved for experimentation and 1023 and 1024 for potential future expansion of ranges. As such the System ports are commonly used for well known protocols and services and often require higher privileges to use. User Ports are commonly used by specific services, but do not require escalated privileges. Dynamic ports are mainly used for temporary connections and as source ports when connecting to other hosts.

2.5.2 Operation

TCP connection is usually established by a 3-way handshake presented in Figure 2.3. The handshake consist of three packets, first the initiator sends a packet with `SYN` flag, to which the receiver responds with a packet with `SYN+ACK` flags, if it wishes to establish the connection, to which the initiator responds with an `ACK` flagged packet establishing the connection.

After the connection has been established ,data can be transferred over the connection. A `sequence number` is chosen while establishing the connection. For streaming, the data is broken up to pieces. The number of bytes from start of the data to the first byte of a piece is added to `sequence number`, allowing the reordering of the received data if needed. Packets arriving out-of-order can happen for example when they are routed over different paths.

When some amount of packets have been received a packet with `ACK` flag and a sequence number is sent confirming that all data before that sequence number has

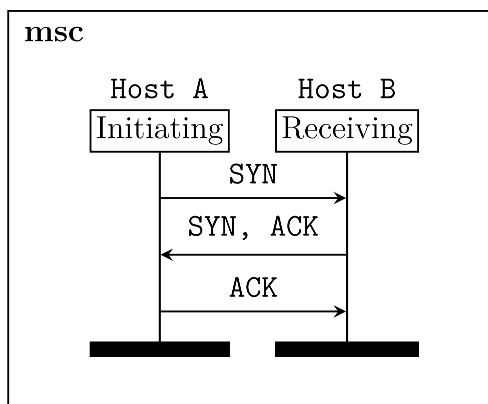

Figure 2.3: TCP 3 way handshake

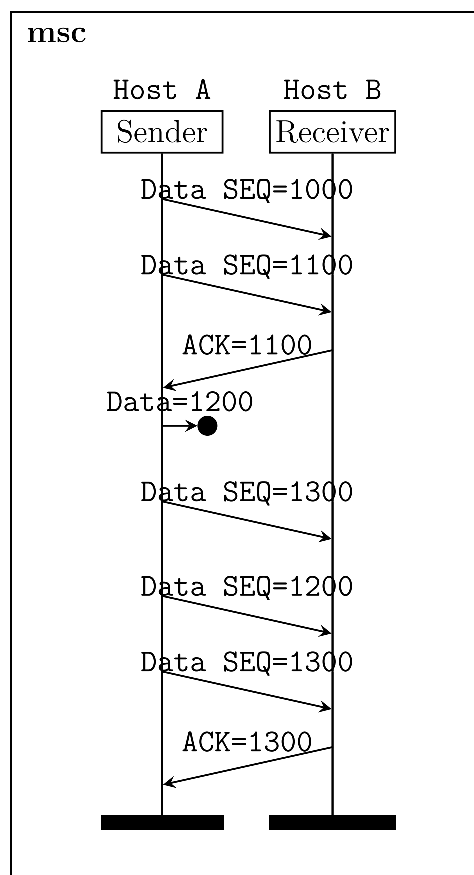

Figure 2.4: TCP transmission and retransmission

been received, as seen in Figure 2.4. Where first the data packets with sequence numbers `1000` and `1100` are sent from the `Host A` and received by the `Host B` which then sends a packet with the `ACK` flag set and the sequence number of `1100`. The packet with the sequence number of `1200` from `Host A` to `Host B` is lost in transit. When this happens the `Host B` does not send an `ACK` packet but waits for the `Host A` to retransmit the packets, which it then acknowledges with a packet with

corresponding `SEQ number`. With the last packet of a certain amount of the data such as single message, `PSH` flag can be used to tell that the receiver should forward data to application holding the socket.

Finally, packets with `FIN` flags and responses to these with `ACK` can be used to close a connection. `RST` flag can also be used to tear down the connection when the other party does not have a context for it anymore.

2.6 User Datagram Protocol

User Datagram Protocol is the other popular transport-layer protocol on Internet defined in RFC 768.[18] It is a connectionless datagram protocol, offering no guarantees of delivery or ordering, unlike TCP. However, in some applications, such as Voice over IP and low latency gaming and control, these are desired qualities as loss of some data is preferable to waiting for full retransmission of past data as in TCP. UDP is also less resource intensive to implement compared to full TCP/IP-stack, as it does not necessitate large buffers or complex state handling. Thus it offers a good alternative if the use of standard and common networking technology is desired, but the hardware resources are restricted. Examples of such cases are various sensors and simple devices like electronic locks. DNS is one protocol which utilizes UDP often as most requests and responses are small and connection setup and teardown is not desired, although for large transfers TCP is used.

0–15	16–31
Source Port	Destination Port
Length	Checksum

Figure 2.5: UDP Header

UDP is a very light weight protocol only having `source` and `destination port`, `length` and `checksum` fields in its header. UDP ports used to share the same purpose allocation as the TCP, but nowadays only port in the relevant protocol is assigned. This process is described in RFC 6335.[17] As seen in Figure 2.5 all fields are 16-bit meaning that maximum length of the packet is 65536 octets. The `checksum` of UDP is not mandatory with IPv4 as the functionality is provided by the lower layer.

2.7 Internet Control Message Protocol ICMP

ICMP is the protocol used for communicating error messages and relaying query messages. Protocol is defined in RFC 792 with some parts deprecated in RFC 6918.[19][20] Common use cases for it include `Time exceeded` send when IP datagrams TTL reaches 0, `Echo request` tool known as ping commonly used to find if a host is reachable and answers, `Destination unreachable` generated when datagram can not be routed to destination, `Redirect` when the receiving router knows that

there is a better route via another router. For these we presented the messages in this section.

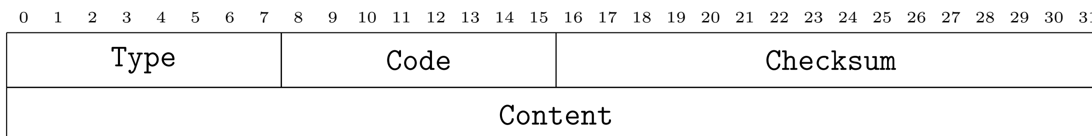

Figure 2.6: ICMP header

ICMP header consists of 8-bit `Type` value, 8-bit `Code` value and 16-bit `Checksum` with a variable number of 4 octet long data. Common types used are 0 and 8 with the `code of 0` for `echo reply` and `request` respectively. `Code 3 destination unreachable` with various codes. `Code 5 redirect` message for redirecting further communication for the network or host. `Code 11 time exceeded` for TTL expiring in the transit.

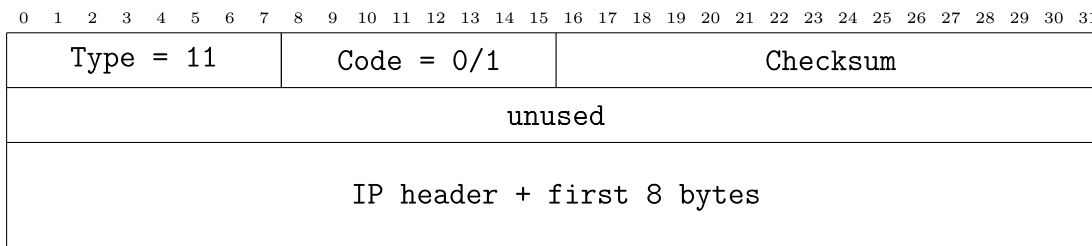

Figure 2.7: ICMP Time exceeded

`ICMP Time exceeded` see Figure 2.7 is used when TTL of a packet reaches 0 as described in Section 2.2, in cases when TTL reaches 0 the code is set to 0. Another use case is when a host fails to reassemble fragmented datagram and `ICMP Time Exceed` is sent with `code of 1`. A common use of this message is traceroute utility used for mapping the route between hosts.

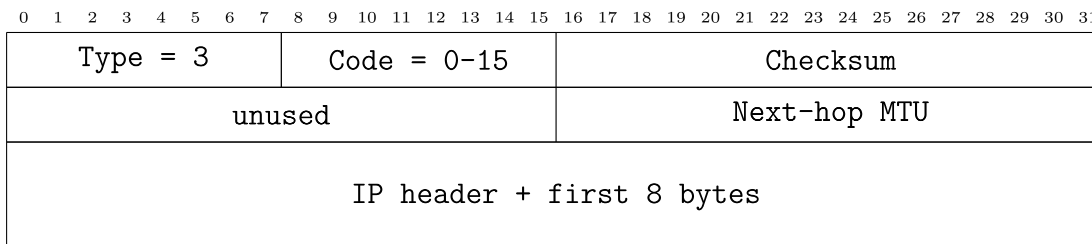

Figure 2.8: ICMP Destination Unreachable

`ICMP Destination Unreachable` Figure 2.8 is send to inform that the host or network cannot be reached. Typical codes include `unreachable network 0`, `host 1`, `protocol 2` and `port 3`. `Code 4` is used when too large packet is received with `don't fragment` flag.

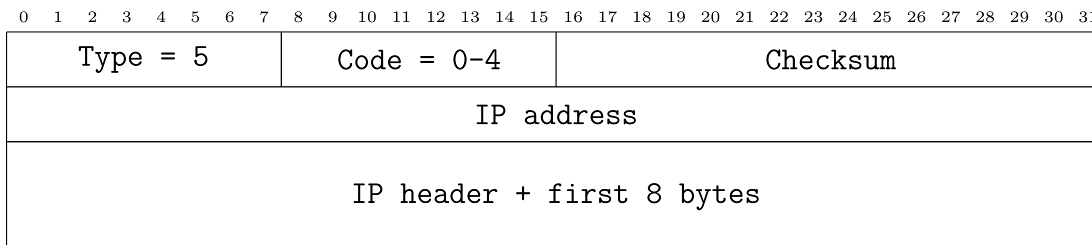

Figure 2.9: ICMP Protocol Redirect

`ICMP Redirect` presented in Figure 2.9 is used when a router needs to convey routing information to a host. It has 4 codes for redirections of datagrams: `0` for `network`, `1` for `host`, `2` for `Type of Service` (ToS) and `network`, and `3` for `ToS and host`. The message includes new destination that the router deems the host having a more direct path and IP Header plus 8 bytes for the host to identify the packet which triggered the message.

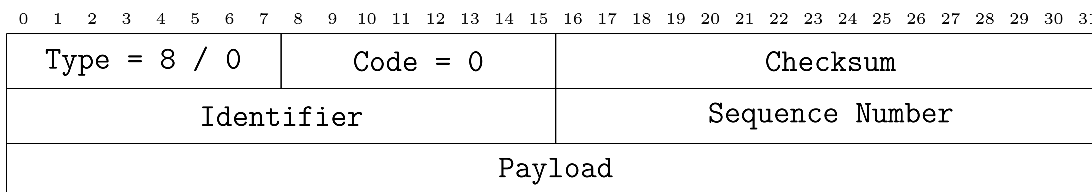

Figure 2.10: ICMPl Echo Request and Reply

Figure 2.10 presents packets for `ICMP Echo Request type 8` and `Echo Reply type 0`. These messages are the most common way to establish that host is reachable in process known as pinging. In the process, one host sends Echo Request packet with an identifier and sequence number set which the other host answers with same identifier and sequence number. If a payload is included, it must also be in the reply packet.

2.8 Link layer, Ethernet, MAC addresses and ARP

The data link layer is responsible for the low level routing of frames and physical transmission of data in the chosen medium. Internet Protocol allows the interconnection of multiple different data link layer technologies, by having routing itself happen on the network layer. Technologies in the layer are commonly defined in standards

written by working groups of Institute of Electrical and Electronics Engineers (IEEE). Thus it allows, for example WiFi devices utilizing IEEE 802.11 standards wireless LAN, cell phones utilizing LTE's E-UTRA, desktops utilizing Fast Ethernet and core routers operating with various fiber-based technologies to communicate with each other. Many of these are standards under IEEE working group 802.[21] Next, Ethernet will be described as the most relevant technology for our test setup.

2.8.1 Ethernet

Ethernet datagram or frame carries 48-bit `destination` and `sources addresses`, optional IEEE 802.1Q header allowing Virtual LANs in the network. Followed by 16-bit `Ethertype` or `length` field, used to identify the protocol encapsulated inside the frame for values over 1500. Next is the payload data for example datagram of network layer protocol, followed by cyclic redundancy check `(CRC) checksum`.

Ethernet addresses are commonly known as Medium Address Control (MAC) addresses. MAC addresses should uniquely identify a network adapter. The 48-bit long addresses are usually presented as groups of 6 hex pairs such as `00:01:02:00:01:11` where first 3 pairs are the `organizationally unique identifier` (OUI) purchased from IEEE Registration Authority and latter 3 pairs are `Network Interface Controller Specific`. These identifiers are used for lower level routing by switches which can learn them and thus route traffic only to ports where such address is known to communicate from. The process allows better utilization of networking capacity as not all traffic need to be sent on all connected lines.

2.8.2 Address Resolution Protocol

Address resolution protocol is an integral part of the routing mentioned above and is used to find MAC addresses for IP addresses. The protocol is defined in RFC 826.[22] Examples of ARP headers are presented in figures 2.11 and 2.12.

The header consist of 16-bit `Hardware type` field specifying the protocol, 16-bit `Protocol type` field storing the protocol for `IPv4 0x0080`, `Hardware address Length` for storing length of Hardware address 6 for Ethernet, `Protocol address length` for storing length of protocol address with IPv4 this is 4 and `Operation` field specifying operation with `1 for request` and `2 for reply`. Next is variable length sender hardware address storing the `Media address` that is MAC address in case of Ethernet and sender `protocol address` such as IPv4 address, in request these fields contain information of the sender of the request. These are followed by similar two fields for the target, which in reply are used to indicate the address of the host sending the original request.

A simple case of ARP operating is when a host needs to connect to an IP address it has not communicated with or heard before, as the modern systems listen to the ARP traffic and store it for future use. For example if the host with IP `198.18.0.112` tries to connect to host at `198.18.0.10`, i.e case both hosts are in the same subnetwork, it will send an ARP seen in Figure 2.11 where it stores its `hardware address`, `protocol address` and the `protocol address` which `hardware address` it wants to discover, the `target hardware address` is filled with zeroes.

0–15
Hardware type: Ethernet (1)
Protocol type: IP (0x0800)
Operation: request (1)
Sender hardware address: 00:01:02:00:01:12
Sender protocol address: 198.18.0.112
Target hardware address: 00:00:00:00:00:00
Target protocol address: 198.18.0.10

Figure 2.11: Example of ARP Request

0–15
Hardware type: Ethernet (1)
Protocol type: IP (0x0800)
Operation: reply (2)
Sender hardware address: 00:00:00:00:02:a1
Sender protocol address: 198.18.0.10
Target hardware address: 00:01:02:00:01:12
Target protocol address: 198.18.0.112

Figure 2.12: Example of ARP Response

In the response seen in Figure 2.12, when such host is around, it sends a reply containing its `hardware` and `protocol addresses` in sender fields and has filled requesters `hardware` and `protocol addresses` in target fields.

2.9 Domain Name System

Domain Name System is a hierarchical decentralized system for naming services of hosts and resources on the Internet. It is not only limited to retrieval of the numerical IP addresses matching to the names, but also can store other information. Thus it allows interconnecting these two address spaces. Domain name system is defined in RFC 1034 and 1035 later clarified by RFC 2181.[23][24][25]

Domain Name System allows the use of human readable addresses such as `www.aalto.fi` by mapping these to network addresses. `Fully qualified domain (FQDN)` names such as `www.aalto.fi` are addresses that are complete containing both `hostname` and `domain name`. In this case, `.fi` is the top-level domain for Finland and its holder Finnish Communications Regulatory Authority (FICORA) has authority over it forming a zone for all `.fi` domains. Aalto University has registered the `aalto.fi` domain from them and has authority over domains and hosts under it forming another zone. One such host is the `www`, from world wide web, which is commonly used for the address of a server or servers serving the websites. Similarly, there can be other domain names such as `mail.aalto.fi` which is the internal webmail service of Aalto University. Inside the same network, it is often possible to also use just the hostnames. As such if you want to connect to `kosh.aalto.fi` one of University's general use Linux servers you can just write `kosh` in a tool.

Domain name system forms a hierarchy. Highest in the hierarchy are the root name servers responsible for the root zone that is the zone containing all of the top-level domains such as `.fi`, `.com` and `.net`. For these top-level-domains, there are responsible servers controlling these zones. And then under them, there are registered domain names each being their own domain and potentially so on. In the system when an address is queried, first the root name server is queried for matching top-level-domain and server responsible, then the server responsible for top-level-domain is queried for the domain itself returning the matching name server, which can have the relevant address available or further zones are checked.

2.9.1 Domain Name Querying and Messages

Two ways of making queries are recursive queries where the DNS server to which the user sends the request performs the whole query process and iterative queries where the DNS server return a hint for next server to query for an address.

Domain name `query/response` header presented in Figure 2.13 consist of 16 bit `ID` field identifying the query, followed by 1 bit `QR` Flag indicating if the header is query or response. 4-bit `Opcode` for Operation Code used for the type of query. The supported types are standard query, status request, notify used for notifying the slave servers about changes in the zone data and update for allowing selective changes in resource records. This is followed by 1-bit Flags: `AA` for an authoritative answer,

in which case the server is authoritative for the record's zone; TC for indicating truncation of the message; RD for recursion desired in a query used to indicate that recursive query is desired and RA recursion available to indicate if the server supports or allows recursive queries; Z for zero as that is a reserved bit; AD flag for authenticated data indicating that server has authenticated data according to servers policies; CD for checking disabled. Filling the 16 bits is 4 bits for RCode: a return code used for possible errors. Next 4 16-bit fields in the end are Question Count for the number of questions in a message, although no more than 1 question is supported. ANcount for Answer record count i.e number of answers message. NSCount is for the number of records in Authority section and Additional Record Count for records in the Additional section.

The DNS system supports multiple types of records which have both human readable and identification value. Human readable ones are usually used in writing configuration files and interpreting DNS responses. Common types include A(1) for IPv4 addresses, AAAA(28) for IPv6 addresses, MX(15) for mail servers, PTR(12) usually used for reverse records meaning for names from IP address and NAPTR(35) Naming authority pointers for special use cases. The system also has CLASSes which allow multiple namespaces, most common is the IN for the Internet.

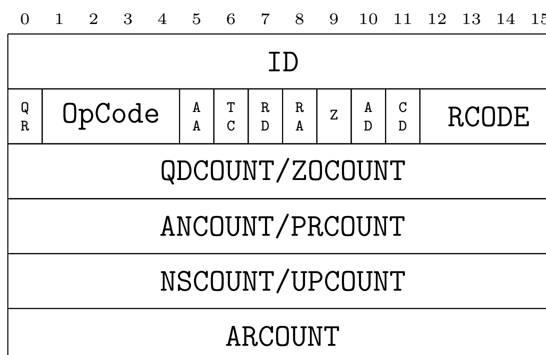

Figure 2.13: DNS header

A normal DNS query consists of DNS message with ID field set to transaction ID. QR bits set to 0 to indicate a query. RD flag can be used if recursion is desired. The QDCOUNT is set to one. The header is followed by the query which starts with the domain name with length of label and label pairs and ending at zero length label. Such as for www.aalto.fi value being hex 03 77 77 77(www), 05 61 61 6c 74 6f(aalto), 02 66 69(fi) and 00. This is followed by 2 octet type and 2 octet class.

The answer to the above query would have a reply with the same ID and QR bit set to 1 to indicate a response. RA bit can also be set to indicate that the server does offer recursion. The question count would be set to 1 and ANCOUNT to the number of answers included. Following the header would be the original question and the answers, hat include name, type, class, time to live, data length and value. Time to live indicates for how long the answer is still valid in seconds. The practice

allows caching of answers either at hosts or recursive servers which lessen the load on authoritative servers.

2.10 Application Layer protocols

Application layer protocols are the protocols that the applications use. Next we present HTTP and FTP as examples of them.

2.10.1 Hyper Text Transport Protocol

Hyper Text Transport Protocol HTTP is a protocol designed to transfer hypertext, interlinked text of traditional websites, pictures, stylistic guides for web page presentation and computer code run inside browsers. But it is also used for a multitude of other purposes such as video and music, both live and served to requests, as its popularity has meant that it has a wide support on a variety of platforms and it is allowed to pass via most firewalls.

Various versions of HTTP exist with the commonly used version being HTTP/1.1 defined in RFC 7230.[26] HTTP 1.1 is a plain text Request-Response protocol. Each request consist of request a line with a keyword from: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTION`, `TRACE` and `PATCH`; followed by possible header fields including further information such as the version of the application, an empty line and potential body for payload. Responses consist of `status code`, header fields, an empty line and message body.

```
GET / HTTP/1.1
Host: test.gwa.demo
User-Agent: curl/7.47.0
Accept: */*


HTTP/1.1 200 OK
Server: nginx/1.10.0 (Ubuntu)
Date: Mon, 27 Feb 2017 13:21:09 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

... HTML site ...
```

Listing 2.1: HTTP/1.1 communication

In Listing 2.1 we present a simple communication. Capture of website retrieval from `test.gwa.demo` using `HTTP/1.1`. `GET` method is used for retrieving the site as a request. In header fields: `User-Agent` indicates `curl`, a command line browser, as the application used. `Accept: */*` indicates that any file is accepted. In the response we get protocol used and the status code of `200 OK` for successful retrieval. In header fields we see the `server`: nginx, `date` and time of the request, details about content, encoding used for transfer and wish for connection to be kept alive. The above is followed by the html code of the website.

2.10.2 File Transfer Protocol

File Transfer Protocol (FTP) is an old protocol designed to transfer files between hosts operating on TCP. It is defined in RFC 959.[27] In many cases, it has been replaced by other protocols such as HTTP, SCP and even Peer-to-peer protocols like BitTorrent. It does have many use cases as many platforms have tools supporting it default configurations and it being relatively simple and light protocol due to its age pre-dating TCP/IP itself.

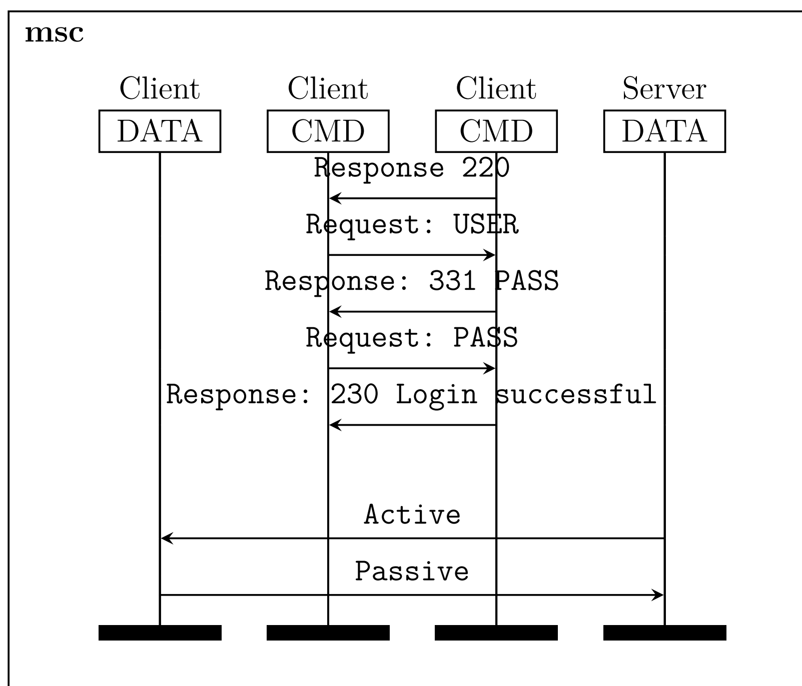

Figure 2.14: FTP Connection establishment

FTP is text-based protocol, were simple requests are sent and 3 number response codes are received from the server. FTP has two operating modes, namely `active` and `passive`. In both cases, client initiates the negotiation by connecting to the server port 21. In the active, mode client provides server an `address port pair` it wished to be connected to. In the passive mode the server provides a second `address port pair` for the client to connect to for the data path. The passive mode is needed for cases where the client is behind a NAT or a firewall that blocks incoming connections. In Figure 2.14 a simple scenario of a user connecting to a server receiving information, sending username, receiving prompt for password and being successfully logged in is presented. From the figure, the process of agreeing on transfer mode and transmission of information related to directory and such is omitted.

Data transfer itself happens over the data channel formed as described above. Data can be transferred in 4 different modes, ASCII of 8-bit ascii bytes of host representation, binary mode of pure bytes, EBCDIC mode for transferring EBCDIC

character set and local mode for proprietary formats matching on both end points. Three modes are also present, stream mode where TCP is responsible for processing, block mode where data is broken into chunks of header, length and data, and the compressed mode where compression algorithm such as run-length encoding, in cases of repeated bytes their number and character in question is stored, is employed.

3 Network address translation, security, software and fuzz testing

In this chapter we first present relevant information about Network Address Translation. Followed by inherent security weaknesses of IP networks and common attacks. After them we discuss some aspects of cryptography. Finally, we conclude the chapter with overview of software and fuzz testing.

3.1 Network Address Translation

Network Address Translation (NAT) is a method of mapping one address to another by changing IP addresses and the checksum of a datagram. The motivation for this is connecting one address space to another, such as a private network to the general public Internet. This is a popular use case as there are more devices connected to the Internet than there are available addresses, thus NAT is used to allow multiple devices to connect to the Internet using a shared address or addresses.[28]

The simplest form of the NAT is the basic NAT or one-to-one NAT. Where only the IP addresses and the needed checksums are changed. These type of NATs are used to interconnect networks with incompatible addressing schemes. One of such scenarios is two private networks sharing the same network prefix. The second form is one-to-many NAT, where a single IP address can be mapped to multiple addresses. This is the type of NAT that is used in most scenarios. One such scenario is a home-network where Internet Service Provider (ISP) provides single IP-address and this is NATed to allow all customer's devices to connect to the Internet.

One-to-many NATs generally fall under a few types. Translation is mapped when an outgoing connection is established. With a full-cone NAT when a connection from an internal address-port pair is made all traffic to the external address-port pair is sent to same internal pair and the outgoing traffic goes via external pair. Address-restricted full-cone differs in only allowing traffic from an address if there has been outgoing traffic. The port-restricted cone NAT further limits the incoming traffic to the exact pairs that have had communication. A symmetric NAT maps a new external address and port for each connection and the incoming traffic must use these.

When two hosts wanting to communicate with each other are behind separate NATs process known as NAT Traversal is required. The most common methods for this are Session Traversal Utilities for NAT (STUN), Traversal Using Relays around NAT (TURN) and Interactive Connectivity Establishment (ICE).[29][30][31]

The NAT cannot support all types of applications and might require Application Layer Gateways (ALG) to operate. One such case is the Active mode of FTP described in Section 2.10.2. In this mode, the client connecting to the server expects the server to connect back but to a different port. This, however, is not possible as the NAT only has a mapping for the outgoing connection. An ALG is needed to monitor the traffic and to do operations on the packets for the protocol to operate. With such help, the protocols requiring incoming connections can operate.

Often devices providing NAT have support for Dynamic Host Configuration Protocol (DHCP) which allows automatic allocation of IP address.[32] These IP addresses are provided dynamically from a pool by a router and have set expiration time requiring occasional renewing. The process allows connecting hosts to new networks without configuring a static IP address to them. It is another way for ISPs to conserve their IP resources as often the customers do not use all of the addresses they are allowed.

3.2 Inherent security weaknesses of IP networks

IP networks have inherent security weaknesses. These often stem from its early design as a network between trusted hosts. IP protocol itself is weak against spoofing, a process of masquerading as some other host which is possible just by changing the source IP address of the packet. As there are no other identifiers the receiving host cannot discern the origin of the packet.

TCP protocol relies heavily on sequence numbers. This opens it to two types of attacks. If the initial sequence number generation is easily guessed it is possible to spoof the ACK packet to complete the handshake and have the receiving host to believe that traffic originates from a different host. Another way is to use FIN or RST packets to close existing connections from a host.

ICMP is also a potential avenue of attack. Spoofed redirect messages can be used to direct traffic to malicious gateways and thus allow their capture or alteration. Denial of Service is also possible with these packets.

3.3 Common attacks

Common attacks can be divided to a few types relevant of which we present here. Some attacks can cover multiple types like the presented distributed denial of service.

3.3.1 Spoofing

Spoofing is the process where attacker masquerades as another host. Common attack type is generating IP packets with wrong source IP. While this does not allow the sender of the datagram to communicate, with connectionless protocols it can allow other types of attacks as many protocols do not authenticate the sender in any way. On the other hand, with rightly timed attacks processes can be disturbed, such as FIN packet with sufficiently close values to real ones closing a TCP connection. They are also used in DoS to prevent auditing of the original sender.

3.3.2 Denial of Service

Denial of Service (DoS) is a type of an attack where the attacker tries to overwhelm the target system with traffic or requests. Simply sending enough traffic to a host on the Internet can use all of their available bandwidth capacity and thus prevent legitimate communication. On the other hand, DOS is also a useful test for discovering limits of the system under test.

One common type of DoS is Distributed DoS, or DDoS where traffic originates from multiple sources. Compromised devices with malware capable of controlling them are known as bots and with controlling nodes in a network form botnets. Use of poorly secured Internet of Things (IoT) devices has also happened.

The amount of traffic that a single device can generate can be increased with amplification attacks, where protocols and servers that generate larger response than the sent message are used with spoofing. The DNS system is an example of such a system where a small request can generate a much larger response. Spoofing happens by using victim's address in the query to the powerful servers with high bandwidth, which then flood victim with large amount of responses.

In 2014 and 2015 Network Time Protocol has been used in this type of attack, where `MONLIST` command returning last 600 clients IPs was used[33].

`TCP SYN` Flood is a type of attack that utilizes lower traffic volume and instead aims to starve resources of the target. In the opening of a TCP connection the side receiving connection has an half-open connection state for that connection. Having too many of these states can limit the system from receiving new connections thus denying service from other legitimate parties.

3.3.3 Brute force and dictionary

Accessing a system normally requires right credentials. One way to attain such is just to guess usernames and matching passwords. In process of brute forcing attacker simply tries password after password such as aaa, aab, aac and so on. This, however, becomes exponentially more difficult as password length and character set increase, 4 digit pin code has ten thousand different possible combinations whereas 4 lowercase letters have almost four hundred thousand combinations and 5 letters nearly ten million combinations. As such more refined techniques are better suited. One such is dictionary attacks where popular passwords and words are stored and used. Dictionaries being much smaller, than are combinations of characters and are faster to use and require less time.

3.3.4 Buffer overflow and injections

If the attacker has further goals than just preventing the system from communicating an access to the system is needed. The process of brute forcing at login often leaves marks and is easily discovered and stopped. Attacking weaknesses in a system and programs is often an effective way to gain access or information.

A common weakness on the web is poorly coded and protected fields. An attacker can inject in these fields database query code which gets executed and then can leak contents of the database to the attacker, such as usernames, passwords, emails and other details of the users. Even if hashing is utilized with passwords, attacker has an opportunity to try brute-forcing them in leisure.

Buffer overflow is an attack where a certain amount of space is reserved by a program, but more is erroneously written. This allows attackers to modify memory outside intended allocation. In certain cases, it is possible to write arbitrary machine code to memory which then can get executed. The code in question can allow the

attacker execute programs or code allowing him to gain a better access to system and thus work further from that point.

3.4 Encryption, encoding and hashing

Encryption is the process of coding information or data, with the goal of preventing unauthorized parties from reading it. Hashing is the process of generating fixed-length data from arbitrary length data in a non-reversible, but repeatable process. Encoding is the process of converting data from one format to another.

Hashing

Hashing is used to generate probabilistically unique fixed sized mappings of arbitrary sized data. The process allows generating so called fingerprints from inputs. Hashing of the passwords is a common use case. Good password hash functions are not reversible, which means that input can not easily be generated from output. As such even if the hash of the password is leaked the attacker cannot discern the password from the hash while verifying the provided password is easy. One other use case is hashing messages or entire files to verify their integrity. Hash-functions and hashes are used this way to offer an extra layer of security.

Public key cryptography and infrastructure

Public-key cryptography is a way to achieve confidentiality in communication. It consists of a system where two keys, public and private, are generated. A message can be encrypted using the public key and then decrypted only using the private key. This allows confidentiality and integrity of data between communication partners. Digital signing is the process of using the private key to generate an output, which is verifiable with the corresponding public key, from a message. Generally, to save space the message is hashed and the hash is signed instead.

Public key infrastructure (PKI) is built on the public-key cryptography. The main component of PKI is the certificates. They are documents issued by trusted third parties containing the identity and the public key of an entity with information signed by the trusted third party. This makes possible to transfer trust. The trusted party guarantees the authenticity of the entity who they provide a certificate to and the signature process proves the integrity of the certificate. The most prevalent use case is the web, where secure connections are formed by using public keys in certificates.

Encoding

Base64 is an example of encoding scheme standardized in RFC 4648.[34] In base64 binary data of 8-bit bytes is encoded limited character set of a-Z, A-Z, 0-9, / and + while using = for padding. This process allows transferring arbitrary data as a text which most systems should be able to handle.

3.5 Software testing

"Testing is the process of executing a program with the intent of finding errors"[35]. Black-box testing or data-driven testing is an method where internal structure and the behaviour of a program is not the concern, but focus is finding circumstances where program does not follow the specifications. In exhaustive input testing, all possible inputs in some range are tested. However, testing all the valid and invalid inputs results in near infinite number of test cases.[35]

White-box or logic-driven testing allows examination of internal structure of the program. As such the strategy could be to build tests that execute all statements in the code at least once, which is also known as exhaustive path testing. However, even in this case, the possible code paths in complex program is astronomical.[35]

Regression testing is software testing done with the aim of guaranteeing that changes made in software do not change the working functionality of the program. This type of testing can be automated and run every time when changes have been made to a system.

3.5.1 Fuzz testing

Fuzz testing is a technique used to find errors in software products. It can be used to test both programs and protocol implementations. Fuzz testing is easily automated and can provide test cases that are not usually considered by the testers. Fuzz testing is an additional tool but does not replace traditional testing or guarantee that all the issues are found.

Fuzz testing allows automated testing of systems against a wide range of incorrect inputs. Incorrect inputs can often result in critical vulnerabilities and bugs such as buffer overruns, crashes, memory leaks and unhandled exceptions. In this sense, functional testing is a subset of data-driven testing where the approach is to aim at edges of valid input base or take a valid input and modify it and check the results. It can be generally divided into a few categories: random, mutation, template or protocol based.

The simplest form of fuzz testing is the random testing. Where entirely random inputs of varying lengths are generated as inputs. It can find situations where invalid inputs are handled incorrectly causing issues. In mutation based fuzz testing known valid inputs are used and modified slightly. Generation or template based methods use pre-defined templates where format or specification of data is known and values are generated in these constraints.

CVE-2014-0160 or Hearthbleed bug in OpenSSL is an example of bug found with fuzz testing by Codenomicon, a company producing fuzz testing tools. The bug could lead to leaking of information in the memory of OpenSSL such as private keys, passwords and email-addresses.

The weakness was in the handling of inputs and memory allocations. TLS/SSL has a feature where the client sends a message and server responds by sending the same message back. The request includes the length of the message and the server allocates this much memory for the message, however, the message sent by the client

was shorter than the length indicated. This difference resulted in the server sending data beyond the message provided, which likely included previously used data.

4 Systems under test

In this section we present the systems we are testing. First in what type of environment we are running them and then components of these environments. After this we shortly present the CESv1 and CESv2 we tested.

4.1 Test enviroment

All tests were done in Virtual Machines (VM) with Oracle VM VirtualBox version 5.0.28 r111378 running under 64-bit Ubuntu 14.04 LTS operating system, on a machine with four core Intel i5-6500 @3.20 GHz Central Processing Unit (CPU), 8GB of ram and 500GB traditional HDD.

Virtualisation was chosen as nothing in the system necessitated direct access to hardware and it allowed simulating multiple hosts on a single computer. Additionally it provided opportunities for fast iteration low risks from breaking stuff as well as backing up and restoring and later entirely rebuilding the test environment was quick. With new iptables CES it also allowed easily transferable environment as configurations could be updated and transferred allowing building the same system by any developer.

4.1.1 CES version 1

For the earlier CES version Proxmox Virtual Environment version 3.4-11/6502936f was provided with two cores and 2048MB of ram inside the VirtualBox, `Vt-X`, `Nested Paging` and `PAE/NX` were turned on. Three network adapters were also configured, Host-only Adapter, NAT and Internal Network. Containers themselves run Ubuntu 14.04.3 LTS (`GNU/Linux 2.6.32-40-pve x86_64`).

Containter:	Memory:	Swap	Role:
`cesa`	128 MB	128 MB	CESA
`cesa-ext`	128 MB	128 MB	DNS for CESA
`cesb`	128 MB	128 MB	CESB
`cesb-ext`	128 MB	128 MB	DNS for CESB
`public-dns`	256 MB	128 MB	main DNS
`public1`	512 MB	64 MB	public host
`public2`	64 MB	64 MB	second public host
`hosta1`	128 MB	64 MB	host behind CESA
`hosta2`	64 MB	64 MB	second host behind CESA
`hostb1`	512 MB	64 MB	host behind CESB

Table 4.1: List of containers run for testing.

List of containers run can be found in Table 4.1. Two instances of the CES were run one on container `cesa` and one on `cesb`, connected to these were containers `cesa-ext`

and cesb-ext respectively providing DNS relays to CES instances. Connected to the cesa were hosts hosta1 and hosta2 and to cesb hostb1. The test involving hosts behind the CES primarily used hosta1. Hosta1, hosta2 and hostb1 communicated to public network via the CES they were connected to. The container public-dns provided dns services. The public1 was primary container used in testing to run tools required for attacks.

4.1.2 CES version 2

For CES version 2 up to date Lubuntu 16.04 LTS virtual machine was used. The VM was provided with 4096MB of memory and VT-x, Nested Paging and PAE/NX was turned on. VM was also configured with a single NATed network adapter for Internet connectivity.

Containter:	Bridge:	Address:
router	br-wan0	100.64.0.1/24
resolver	br-wan0	100.64.0.2/24
public	br-wan0	100.64.0.100/24
br-wan0	br-wan0	100.64.0.254/24
router	br-wan1	100.64.1.1/24
gwa	br-wan1	100.64.1.130-(142)/24
router	br-wan2	100.64.2.1/24
gwb	br-wan2	100.64.2.130-(142)/24
gwa	br-lan0a	192.168.0.1/24
test_gwa	br-lan0b	192.168.0.100(-250)/24
gwb	br-lan0a	192.168.0.1/24
test_gwb	br-lan0b	192.168.0.100(-250)/24

Table 4.2: List of lxc-containers and the bridges they are connected to and their addresses

The test network was set up using lxc-containers. A custom orchestration environment was used to generate 7 containers: router, resolver, public, gwa, gwb, test_gwa and test_gwb. The system allowed automatic building of the test system. In the building process multiple packages are installed automatically in a base container ctbase with some common configurations, SSH key files and test files are included. This container is then cloned in listed and individual configurations for networking and starting required services are set up. Additionally, various scripts are run at launch such as pulling the CES version 2 from git to gwa and gwb. Also the namespaces for public, test_gwa and test_gwb are created.

Containers, the bridges they are connected to and their addresses can be seen in Table 4.2. The br-wan0 is not a container, but a bridge offering a gateway to the host system for connectivity to public Internet. Containers router, resolver and public in the subnetwork 100.64.0.0/24 can connect directly via this. Router and gwa

are in the subnetwork `100.64.1.0/24` and `router` and `gwb` are in the subnetwork `100.64.2.0/24`. Both `gwa` and `gwb` have the addresses `.130` and are designed to use also addresses `.131-142`. Router acts as a gateway for both networks. In separate private `192.168.0.0/24` networks are containers `test_gwa` and `test_gwb` with `gwa` and `gwb` acting as gateways and as CES respectively. `Test_gwa` and `test_gwb` have the address of `.100` and network namespaces `test101-test250` set to use address `.101-250`. These namespaces can also request dynamic addresses with DHCP.

4.2 CES version 1

CES is a technology developed as a replacement for NATs and firewalls. It can also include RGW a component that allows hosts behind a node to communicate with public Internet. CES utilizes Customer Edge Traversal Protocol (CETP) to communicate needed information of edge traversal process between CES devices or nodes. Development of parts of the system has been presented in theses by Jesus LLorente Santons, Maryam Pahlevan and Hammad Kabir.[36][37][38] Later publications expand on these works and describe new findings and present a focus on 5G.[39][40][41][42]

4.2.1 RGW

RGW is a component of the CES node which allows a host behind the node in a private network to communicate with the hosts outside. RGW also contains the ALGs which allow the protocols like SIP and FTP to operate. When hosts behind a node try to communicate to outside the RGW acts like NAT.[36]

For incoming traffic the RGW has a novel way of operating. It has a pool of IP addresses, which are allocated in a circular manner. For a connection to be established a DNS query is required against an FQDN of a host behind the node. When a query is received, a reply with the next IP address from the `circular pool` is sent. The node sets a waiting state for this IP address allowing a single connection to it. After the connection is established, the following packets in the flow are translated and the address is returned to the pool. This process allows edge routing to happen and the host in private network to be accessible from outside unlike with traditional NAT.

The circular pool is open to at least two types of attacks. Circular pools can be exhausted as a single query reserves an address for a time. Multiple queries can thus be generated to stop anyone from resolving the FQDNs of hosts behind the node. DNS records are not cached due to TTL being 0. The IP address in replies are also transient and only available for a short time. Another type of attack the system is susceptible towards is hijacking where the attacker tries to send packets to claim an address in a waiting state. To stop this the RGW has a bot detection system.

4.2.2 CETP

CES nodes communicate using the CETP protocol. The protocol is a binary protocol utilizing fixed header and number of Type Length Value (TLV) elements. There

are two types of TLVs Control and Payload, a single CETP message can contain multiple control TLVs. The version of protocol presented below is from Hammad Kabir's thesis with some information updated to latest available version.[38]

The CETP consist of two planes control and data plane. Control plane is used to communicate signalling information between nodes and data plane is used for encapsulated data traffic. The CETP utilizes policies for negotiation of trust and setting up communication channels.

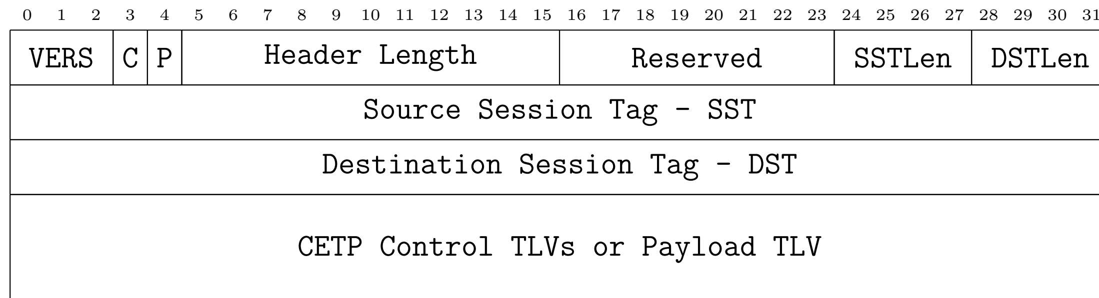

Figure 4.1: CETP Header

CETP protocol messages always has the header, presented in Figure 4.1, and one or more Control TLVs or single Payload TLV with the possibility of some Control TLVs. In the 32-bit fixed header, there is a 3-bit `version` field, `2` for this version, 1-bit `C` and `P` flags used to indicate the presence of Control or Payload TLVs. `Header length` is 11 bits, followed by 8-bits of reserved space and two 4-bit fields for `Source Session Tag Length (SSTLen)` and `Destination Session Tag Length (DSTLen)`. Following these are `SST` and `DST` fields and the payload. The `header length` stores the length of header and control TLVs. `SSTLen` and `DSTLen` store `SST` and `DST` lengths in 4-byte words.

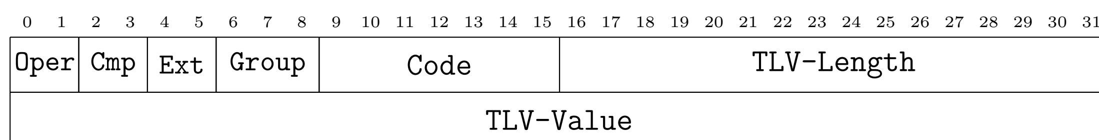

Figure 4.2: CETP Control TLV

CETP control TLV is presented in Figure 4.2. A control TLV has 16 bits for type information, 16 bit for `Length` and variable sized payload padded to 32 bits. The first 2 bits are for `operation`: query, response or information. The next 2 bits for `compatibility` to indicate nodes capabilities: available, not available, not required or required. The 2 `extension` bits are for future extensions. The 3 bit `group` value indicates the general type of TLV: identification, payload, resource location (rloc), control or mobility. The last 7 bits of this two octet group are for the code of the TLV,

which are listed as constants for CETPLibrary in the Appendix B. In Figure 4.3 and Figure 4.4 the TLVs for payloads for use of CETP dataplane are presented.

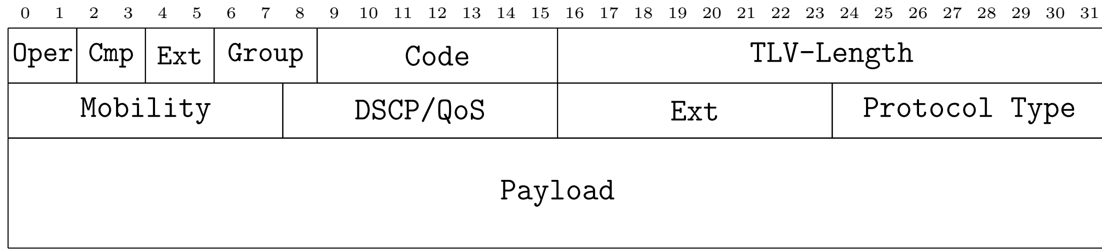

Figure 4.3: CETP Payload TLV for Ethernet encapsulation

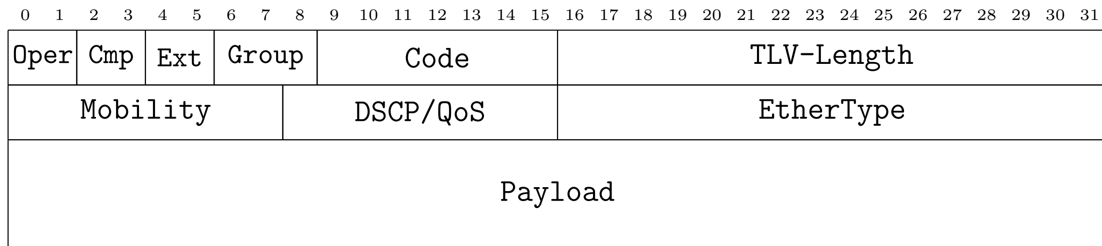

Figure 4.4: CETP Payload TLV for IPv4/IPv6 encapsulation

Most CETP communications consist of establishing a trust between hosts and negotiating a communication channel. Trust establishing happens by exchange of certificates and signatures. In the most basic case, on behalf of connecting host CES sends a message with its certificate and signature and a query for certificate and signature. The signature is a TLV with a hash of the message as a hexstring signed with the private key of the certificate attached.

After the trust has been established the CES nodes try to negotiate a matching policy based on policies stored in them. The policies include details of what they offer, require and what is available. These are usually the identifier, rloc and payload.

4.3 CES version 2

CES version 2 was released during the process of writing this thesis. The CES version 1 was not going to be developed further, rather it is seen as a proof of concept that has already served its purpose. CESv2 includes the ideas of CES but its approach is from a different direction. The main change is leveraging Linux kernel's iptables. The whole approach for the design from our view is use of the standardised technologies and clear separations of responsibilities. Thus designing a system where individual components are easily replaceable and could be in the future provided by different

vendors.

The iptables based approach allows the use of ubiquitous technologies already implemented and well tested for the required task. The iptables is a chain and table based firewall. This structure allows a fine-grained setting of rules for various goals. In addition, iptables allows marking of packets in flows and handling them based on these marks. Additionally, it allows processing chosen packets in user space with user-written programs. This is where the important step of CES packet handling happens, only new incoming connection packets are handled with CES code. After this any further packets can be processed with iptables. Other important design changes are use of python3, asyncio and SYNPROXY. The system uses iptables firewall rules and supports grouping users behind policies and users having individually set policies with the aim of developing a system to generate these policies and store them in a separate service for retrieval to any CES that the user connects to.

5 Tools

In this chapter, we describe the tools used in the work. First, we describe the Robot Framework on which we build our solution. After that, we present our work: `CETPLibrary.py` and CETPLibrary. In Chapter 7 we present results achieved by using it. Finally, we describe the tools we used in other parts of our work.

5.1 Robot Framework

Robot Framework, later in this section just RF or the framework, is an automated testing framework. Originally it was presented in a Master's thesis at Helsinki University of Technology by Pekka Laukkanen and was later adopted by Nokia.[43] The framework is keyword driven. The framework is written in Python. It has excellent report and documentation generation functionalities. In many cases it is heavily Unicode based and as such have very good textual capabilities.[44]

Robot Framework code consist of cells that are separated by either double spaces or space-pipe-space structures. Lines can be in some cases extended with three dots.

5.1.1 Test case styles

Robot Framework supports three different ways of building test cases. In the keyword-driven approach some state is initiated, something is done and finally state is checked. Examples of keyword written in this style can be seen in Listing 5.2. In the example, we open a connection then try to log in and finally check if our login was successful. The test can potentially fail at any point.

Next is data-driven approach where we provide a keyword that accepts parameters as a template. This template can then be easily fed with multiple different parameters to easily iterate over multiple cases.

Finally `Given/When/Then/And/But` prefixes are used in a behaviour-driven style which allows setting initial condition `given`, `when` something is done `and` something else is done resulting in `then` something happening.

5.1.2 Variable types

Robot Framework supports 4 variable types scalar, list, dictionary and environment. Each variable is inside `{}` and denoted with symbol `$` for scalar, `@` list, `&` dictionary and `%` for environment.

Scalar variables alone in a cell are used as is and in a cell they are converted to Unicode presentation and catenated together.

List variables can be used in multiple ways. With `$` they are used as scalar variables. With `@` the values inside are passed as different arguments. Individual items can be also accessed with `[]` notation.

Dictionary variables can also be used in multiple ways. As with list `$` treats them as single scalar value. If named arguments are used with keywords they can also be passed used with `&`. It is also possible to access items inside a dictionary with `[]`

and the key in a similar way to list variables. It should be noted that dictionaries can only be followed by named arguments or other dictionaries.

5.1.3 Files, tables and settings

In the Framework, files can be roughly classified in resource files, variable files, test case files and initialization files. Where resource files generally contain keyword definitions and variable definitions. Variable files allow importing variables from code files. Initialization files are mainly for test suite related settings as they cannot contain test and keywords and variables are not available in lower level test suites. Test case files contain the tests and form a test suite.

There are four different tables used in robot framework noted with `*** ***`. These tables are `Settings`, `Variables`, `Keywords` and `Test Cases`. `Keywords` and `Variables` mainly exist in Resource files, `test cases` only in test case files and settings can be in initialization files. A single table can contain unlimited amount entries. In `test case` and `keyword` tables the settings are noted with square-brackets.

`Settings` table is used to import resources with `Resource`, Libraries with `Library` and variables from variable files with `Variables`. Setting tables can also contain `documentation`, `metadata` and default or forced tags. It allows setting `suite` and `test setup` and `teardown` keywords. Additionally, it is used for test case template and `timeout` functionality.

`Test case` tables contain one or more of the test cases. Each test case can have its own settings for `documentation`, `tags`, `setup`, `teardown`, `template` and `timeout`. These settings override or extend the ones set in `Settings` table.

`Keywords` table contains keywords which can have settings for `documentation`, `tags`, `arguments`, `return`, `teardown` and `timeout`. An example of this can be found in Listing 5.2 where we provide arguments to keywords and the keywords return variables containing connection identifier.

5.1.4 Libraries

The main way to extend more complex capabilities of the Framework is the use of libraries. The framework offers a few standard libraries: `BuiltIn`, `Dialogs`, `DateTime`, `Collections`, `OperatingSystem`, `Process`, `Remote`, `Screenshot`, `String`, `Telnet` and `XML`. We list the libraries used in the work in this chapter. Additionally, there are some external specific purpose libraries such as `SSHLibrary` and many others like libraries for testing and interacting with HTTP, FTP, Android, Database and Django. There is also the library `Rammbock` for generic network protocol testing.

`BuiltIn`

`BuiltIn` is a standard library containing generic keywords which are often needed. It offers very basic logic checks to verify conditions and running keywords in various ways like `Run Keyword If`. Common use cases are checking output strings with `Should Contain` and `Should Not Be Equal`. It also has functionality for setting variables `Set global variable`. Other basic uses are conversions like `Convert To`

`Integer`, logging with `Log`, basic catenation using `Catenate`, and commands like `Sleep`.

String, DateTime, Collection

`String`, `DateTime` and `Collections` are other examples of standard libraries. The `String` library contains keywords for more special string manipulation than the `BuiltIn`, for example, keywords for splitting strings, checking and changing the case and replacing or removing parts of a substring. `DateTime` contains keywords for creating, verifying and doing calculations with date and time values. And `Collections` with keywords for the list and dictionary handling which are the data structures of the Framework.

SSHLibrary

SSHLibrary is an external library useful for communicating with remote servers using SSH. It can be used to manage SSH connections with keywords like `Open Connection`, `Switch Connection` and `Close Connection`. It allows login on servers using `Login` or with key using `Login With Public Key`. The main use for us were the keywords `Execute Command` for running a command in a new terminal and reading output once execution is finished, `Start Command` for running a command and then using `Read Command Output` for reading output. The other way to achieve this is `Write` and `Read` which directly write and read to commandline. For reading, there are also commands like `Read Until Prompt` and `Read Until Regexp` if a more specific functionality is desired. SSHlibrary also supports basic file system access with commands like `Get File`, `List Directory`, `List Files In directory`, `File Should Exist`, `Directory Should Not Exist`.

Remote

`Remote` is a special case of the libraries. It does not have its own keywords but instead is a special library allowing connecting to Libraries running on the remote severs. It acts as a proxy and provides keywords running on the server to be used in test suites. An example of use is `Library Remote http://192.168.254.112:25400`, which links the `Library Remote` to the given address allowing accessing keywords on that server.

RF can be extended with Libraries written in Python. If the framework is run on Jython it is also possible to use Java.[44] Excerpt from `CETPLibrary.py` which we developed can be seen in Listing 5.1. In the excerpt we define a simple function and show how `RobotRemoteServer` is used.

```
class CETPLibrary(object):
  def append_certificate_tlv(self, certFileName, tlvs=None):
    f = open(certFileName)
    cert = f.read()
    f.close()
    tlv = self.new_tlv_dictionary(2, 0, 0, 3, 12, cert)
    return self.append_tlv(tlv, tlvs)
```

```
if __name__ == '__main__':
  import sys
  from robotremoteserver import RobotRemoteServer

  RobotRemoteServer(CETPLibrary(), host=sys.argv[1],
                    port=sys.argv[2], allow_stop=True)
```

Listing 5.1: Excerpt from CETPLibrary.py

The function presented simply takes a file name and potential list of tlvs, reads the file provided, generates a new tlv where it sets relevant values and appends it to the list and returns that list. A use case of the function can be found in Appendix C. In the name of the function, the underscores correspond to spaces in keywords. Parameters and return value can be used directly in the RF code.

If the file is run as the main function, the lower part is entered. Where we `import sys` for getting command line parameters. `RobotRemoteServer` is imported to run the library as remote server. The function is called with library's class name, command line parameters providing IP address and the port and parameter for allowing stopping. The process does everything needed for starting a server, reading and communicating keywords and taking care of communication between the Framework and the server using XML/RPC.

5.1.5 User keywords

User-written keywords are the main way to extend the framework's capabilities and allow easy reuse of the code. In Listing 5.2 we present a few the keywords from our developed testing framework. With original work done by K.C. Amir in his thesis.[45] Later expanded by the author.

```
*** Settings ***
Documentation  Library for general RGW/CES operation and use.
Library  SSHLibrary
Resource  global_variables.robot

*** Keywords ***
Log In
    [Arguments]  ${HOSTNAME}  ${USERNAME}  ${PASSWORD}
    ${conn}  Open Connection  ${HOSTNAME}
    ${output} =  SSHLibrary.Login  ${USERNAME}  ${PASSWORD}
    Should Contain  ${output}  Welcome
    [Return]  ${conn}

Log In with Dict
    [Arguments]  &{LogInfo}
    ${conn}  Open Connection  &{LogInfo}[host]
    ${output} =  SSHLibrary.Login  &{LogInfo}[username]  &{LogInfo}[password]
    Should Contain  ${output}  Welcome
    [Return]  ${conn}

Log In as Root with key
    [Arguments]  ${HOSTNAME}
    ${conn}  Open Connection  ${HOSTNAME}
    ${output}  Login with public key  root  ${KeyFile}
    Should Contain  ${output}  Welcome
    [Return]  ${conn}
```

Listing 5.2: User created keyword in Robot Framework, part of global_lib.robot

In the example we see our usual approach of first providing documentation and importing relevant resources in settings table, in this case, the `SSHLibrary` and variable file. `Log In` keyword is our base case of taking as arguments the server's address and user credentials. After this, we open a new connection to the server and store its identifier in conn variable. Next, we try to login on the server using `SSHLibrary`'s keyword `Login` and checking the output containing word Welcome to indicate successful login. Finally, we return the connection variable for later use.

`Log In with Dict` was our attempt for less verbosity by utilizing a dictionary to have single variable instead of multiple. This approach, however, was not fruitful due to the framework's handling of dictionary variables with other variables. `Log In as Root with key` uses key based authentication where we simply open connection and try to login as root with key provide in variable `KeyFile` containing path and file name to a ssh-key.

5.2 CETPLibrary

In this section we describe our work on `CETPLibrary`. First we cover our general design decisions and the thought process behind these. Then we provide some more specific details about implementation.

5.2.1 Design decisions

As discussed before the existing libraries for Robot Framework even for protocol generation are limited. `Rammbock` is an example of a library we would and very likely would have supported the type of protocol that CETP is and had some existing example relating to DNS which would have allowed more complex systems to be build. However, it did not support protocols directly over IP which we needed for CETP implementation.

The need of raw sockets to send data was found to be a key problem in general. They were needed as the protocol required a setting of IP protocol field to 254. Additionally, these types of sockets require root privileges which with other good practices made general development more complex as a direct iteration of code on non-root users was not possible. Using the root account for general use is considered bad practise.

Robot Framework allows developing libraries in multiple languages, but as most of the development this far had been done in Python it was chosen for implementation of the library too. With Python there are multiple libraries available even for the type of networking needed, but they involved a lot of overhead and we found our problem to be simple. The approach allowed us to use our own data-structures which interlinked into Robot Framework nicely.

5.2.2 `CETPLibrary.py`

For testing CETP-protocol with Robot Framework `CETPLibrary.py` was written with an example provided in Listing 5.1. Documentation of `CETPLibrary`'s keywords

can be found in Appendix A. The library extends the framework and it can be used with `Remote` library.

`CETPLibrary` was developed to allow the generation of relevant packets. It allowed writing test cases in Robot Framework for generation of headers and TLVs with chosen fields. It has functionality for fuzzing any of the fields and potentially allow encapsulating traffic. It was used for generating packets for testing edge cases. Vasic security exchange was implemented utilizing it in Robot Framework.

In the development it was found that XML/RPC does not operate correctly with our binary data. As such base64 encoding was chosen to pass around the data between Robot Framework and library running on a remote host. Core data structures of `CETPLibrary` are the Header and the TLV dictionaries later of which also utilized in the lists containing all of the TLVs of a single message. This approach allows us to set any of the protocols fields to desired values. Additionally, we have keywords for reading the certificate files and private key files and generating the valid certificate and signature TLVs, for which we utilized M2Crypto-library which was also used in CESv1.

On the other hand, we have the various keywords for network communication. These exist usually in both `B64`, for base64, and non-B64 names as mentioned before XML/RPC did not act nicely with our binary data. We developed functions for both sending and receiving IP packets. Additionally, we developed keywords for encoding and decoding our dictionaries to relevant bytes.

5.2.3 CETP_lib.robot

`CETP_lib.robot` found in Appendix B is our generic resource file which contains the known constants of CETP protocol and a few useful keyword. These constants are grouped and prefixed in such a way that we have `OPE` for operations, `CMP` for compatibility, `Ext` for extensions, `GRP` for group and then various code groups: `CRLOC` for `RLOC`, `CPL` for payload, `CCTRL` for control and `CID` for id. In the keywords table, we have a few basic keywords. `Setup` is used to create a new rawsocket at the initialization of communication.

`Send Message` takes in the header dictionary containing the desired payload and source and destination addresses. Then it converts the whole header into bytes, which are encoded to base64 and then sent via the socket.

`Add TLVs` and `Add TLVs and signature` are for adding tlvs to headers. The latter utilizes the first one for this purpose. `Add TLVs` takes in a header dictionary and the list of tlvs converts the list to bytes which are base64 encoded. Is then used to modify headers value field, after which the length field in the header is updated to match. Adding signature works by doing above, calculating the signature using provided private key from the CETP packet and then appending the new tlv list to replace the old one.

`Perform Security Handshake` is used to complete security negotiation using most basic policy. It takes in source and destination addresses, certificate file name and private key file name. Then it creates the header with correct values and two query TLVs for signature and certificate. Next, it appends the certificate it reads

from the provided file to a TLV list and then appends to this list the mentioned query tlvs. Next, it uses `Add TLVs and signature` to append correct signature and sends this message from `198.18.0.112` to `198.18.0.10` our fixed testing addresses. After this, it receives a response message and decodes it to header and tlvs.

5.3 Software and tools

In this section we describe the relevant tools and software we used. All the work was done on Linux. In addition to those listed below we would like to mention `nginx` as the web server and `vsftp` as the FTP server. To provide general DNS services in the environments `bind9` was used in both and `dnsmasq` in the version 2.

Secure Shell SSH and Secure Copy Protocol

Secure Shell (`SSH`) is a popular protocol and a tool offering encrypted communications. It supports both password and key-based authentications. Thus it allows an easy automation with suitable frameworks.

Secure Copy Protocol (`SCP`) allows copying files between remote hosts. It operates over SSH. `SCP` allows both copying from initiating host to remote host and from remote host to initiating host. In addition, it allows copying files from remote to remote host.

5.3.1 Wireshark packet dissector and analyzer

Wireshark is a popular suite of tools used to capture and analyse network traffic. By default, it supports dissection of multiple protocols such as Ethernet, IP, TCP, UDP, DNS, HTTP, FTP and ICMP. It is also extensible with user writeable plug-ins for adding support for new protocols. As there was a need to easily analyse CETP traffic and packets such dissector was needed. Thankfully a version was already written by my advisor Hammad Kabir as a special assignment. In the use of the dissector a bug was found, where a too small variable was used for TLV length, which was fixed. Later the plugin was updated to better match the current state of the protocol as it was originally made for an older version. Plug-in also was specifically compiled for Wireshark 1.8.2 the version already installed in Proxmox environment. Matching the version was important as later versions had changes in architecture making plug-ins compiled in them incompatible with the version installed.

In Figure 5.1 a screenshot of Wireshark with the plugin is presented. In it, we can see how dissectors such as DNS are available by default and provide information about packets. We can also see filtering options at the top where we have chosen only to show DNS and CETP packets. Next, we can see the list of packets with their `ordinal number`, the relative `timestamp` from first packet, `source` and `destination addresses`, `protocol`, `size` and `info` containing dissected information.

Next in the Figure 5.1 we can see more detailed information of a packet. More information of the relevant layer can be shown. In the image layers, are the pseudo frame from virtual bridge `vmbr102`, Ethernet, IP and CETP. We can see the structure of CETP packet here, with first being `flags` and `header length`, followed by the

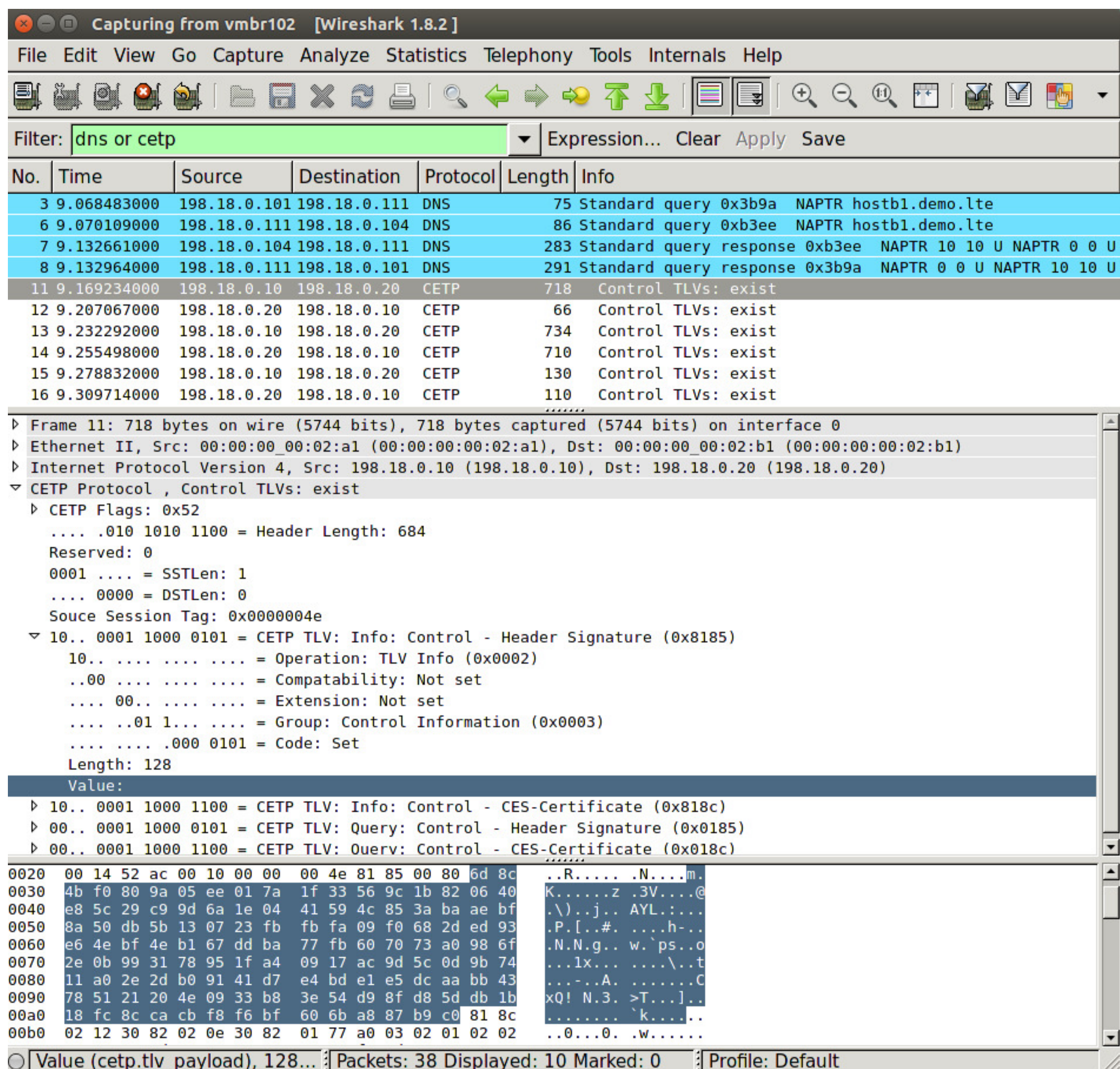

Figure 5.1: Wireshark with CETP dissector plug-in

`reserved` value, `source` and `session tag lengths` and the `source session tag`. After this, there are 4 CETP TLV values. The tool especially shines in easy viewing of bit values in TLVs and their lengths. We can see which flags are set and what type of TLV is in question. Additionally, due to work done on the dissector we can a see simple overview of TLV in the form of `Operation`, `Group` and relating `Code` translated. At the end, we can see the hex dump of the packet, with selected field coloured.

5.3.2 Commands: `time`, `watch` and `tail`

In Listing 5.4 we see an example of the use of time command. The time, in this case, is a build in function of bash command prompt. The time can be used to tell how long it took for a command to run. In the example the command run for 11 minutes 42 seconds. It can be also used as a manual timing tool as the command run can

be terminated by issuing interrupt signal with Ctrl+C combination in the terminal. Time command also shows the system time spent in executing a command so it can potentially be used to discover resources usage.

An example of use of Linux's watch tool is seen in Listing 5.6. Watch tool allows running a command repeatedly. The flag -n detonates an interval in second how often a command is run. The watch runs a command and shows the output.

The tail command allows tracking of lines appended to files. It can either be used to print new lines since the last run or continuously print the added lines to console.

5.3.3 Network bandwidth measurement: `iperf`

`iperf` is a bandwidth measurement tool. It allows testing raw bandwidth between a server and a client. It also supports various options for tuning the testing such as TCP or UDP mode, multiple parallel connections, setting congestion algorithms on Linux, choosing the amount of data transferred and time test runs. An example of a run of the tool can be found in Listing 5.3, where first we see starting the server and its output and the next the running of client and output produced. The addresses and ports used are listed with the time, the transferred data and the bandwidth observed.

```
root@test_gwa:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.0.100 port 5001 connected with 100.64.0.100 port 35120
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  31.7 GBytes  27.2 Gbits/sec

root@public:~# iperf -c test.gwa.demo
------------------------------------------------------------
Client connecting to test.gwa.demo, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 100.64.0.100 port 35120 connected with 100.64.1.134 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  31.7 GBytes  27.2 Gbits/sec
```

Listing 5.3: Use case of: `iperf` server and client

5.3.4 Packet generator and port scanner: `hping3`

`hping3` is a tool allowing the generation of various types of packets including IP, ICMP, UDP and TCP for port scanning and other purposes. It can be used for basic port scanning, or just generating packets in general with various options set at very high speeds and as such being a very good generic tool for testing DoS attacks.

```
root@public1:~# time hping3 hosta1.demo.lte -d 1460 -i u2000 -c 300000
HPING hosta1.demo.lte (wan0 198.18.0.11): NO FLAGS are set, 40 headers +
 1460 data bytes
len=40 ip=198.18.0.11 ttl=63 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=23.5 ms

--- hosta1.demo.lte hping statistic ---
300000 packets transmitted, 1 packets received, 100% packet loss
round-trip min/avg/max = 23.5/23.5/23.5 ms
```

```
real    11m42.188s
user    0m6.730s
sys     0m46.470s
```

Listing 5.4: Use case of: `hping3` tool

An example of a run of the tool is presented in Listing 5.4. In command the `hosta1.demo.lte` refers to target host, `-d 1460` to amount of data sent in a packet, `-i u2000` for the interval between packets in microseconds and `-c 300000` for the number of packets. Interval of 2000 microseconds corresponds to 500 packets in a second.

5.3.5 DNS enumeration tools: `dnsrecon` and dnsenum

`dnsrecon` is a powerful tool allowing brute forcing of records under domain by using a dictionary. Additionally, it allows enumerating srv records, reverse look up of ranges, google searches for sub domains and hosts, performing zone transfers and testing if the domain is under any other top level domain.

Use of the tool to brute force hosts under demo.lte can be found in Listing 5.5, where the time command is used to time how long the process took, -d option for naming the domain, -t brt for brute force and -D for providing our dictionary names.txt.mod. Additionally -n could be used for the address of the nameserver. In the example `dnsrecon` has found the nameservers under the domain and 4 different hosts with AAAA records.

```
root@public1:~# time dnsrecon -d demo.lte -t brt -D ~/names.txt.mod
[*] Performing host and subdomain brute force against demo.lte
[*]      A ns1.demo.lte 198.18.0.111
[*]      A dns.demo.lte 198.18.0.111
[*]      AAAA dns.demo.lte fd00:bbbb::c612:6f
[*]      A hosta2.demo.lte 198.18.0.12
[*]      AAAA hosta2.demo.lte fd00:bbbb::10:11
[*]      A hosta3.demo.lte 198.18.0.13
[*]      AAAA hosta3.demo.lte fd00:bbbb::10:12
[*]      AAAA hosta1.demo.lte fd00:bbbb::10:10
[*]      AAAA hosta4.demo.lte fd00:bbbb::10:13
[*] 9 Records Found

real    2m53.304s
user    2m10.688s
sys     0m27.687s
```

Listing 5.5: Use case of: `dnsrecon` tool

Dnsenum is similar tool which additionally supports multi-threading and automatically performing reverse look ups for found IP addresses. It is also much smarter in real word scenarios where it can detect potential wild card names.

5.3.6 packet generator: `sendip`

The `sendip` is a tool that allows generating packets with either random data or sending contents of files as packets. It allows simple crafting of packets using its build in module system. Listing 5.6 presents the two use cases. In the first line we use the watch command with 1-second interval to run the tool. The tool itself is set to generate

480 bytes of random data, with using module IPv4, setting protocol to 254, source IP address to 198.18.0.112 and destination IP address of 198.18.0.15. In the second line we use it to send binary data stored in raw.bin containing test data.

```
watch -n 1 sendip -d r480 -p ipv4 -ip 254 -is 198.18.0.112 198.18.0.15

sendip -f raw.bin -p ipv4 -ip 254 -is 198.18.0.112 198.18.0.15
```

Listing 5.6: Use cases of: sendip tool

5.3.7 File retrieval from web: wget

wget is a common tool for scrapping websites. It can recursively download a webpage and any sub-pages or images linked to. In Listing 5.7 we can see how it operates. We provide it the site and flag -r for recursive mode. After this it downloads our site index.php which contains multiple images and we can see how it downloads them too, in listing we have omitted other images, but at the end we can see the number of files, size of download and speed. wget also allows us to download items over FTP which makes it a usable tool for testing network performance.

```
root@public:~# wget -r http://www.test.gwa.demo/index.php
--2017-03-01 12:51:17--  http://www.test.gwa.demo/index.php
Resolving www.test.gwa.demo (www.test.gwa.demo)... 100.64.1.130
Connecting to www.test.gwa.demo (www.test.gwa.demo)|100.64.1.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'www.test.gwa.demo/index.php'

www.test.gwa.demo/index.p     [ <=>                  ]     637  --.-KB/s    in 0s

2017-03-01 12:51:17 (101 MB/s) - 'www.test.gwa.demo/index.php' saved [637]

--2017-03-01 12:51:17--  http://www.test.gwa.demo/aalto.jpg
Reusing existing connection to www.test.gwa.demo:80.
HTTP request sent, awaiting response... 200 OK
Length: 12086 (12K) [image/jpeg]
Saving to: 'www.test.gwa.demo/aalto.jpg'

www.test.gwa.demo/aalto.j 100%[===================>]  11.80K  --.-KB/s    in 0s

2017-03-01 12:51:17 (574 MB/s) - 'www.test.gwa.demo/aalto.jpg' saved [12086/12086]

FINISHED --2017-03-01 12:51:17--
Total wall clock time: 0.02s
Downloaded: 6 files, 827K in 0.001s (866 MB/s)
```

Listing 5.7: Use case of: wget

5.3.8 FTP-client: tnftp

tnftp is an FTP client for Linux ported from NetBSD open-source Unix-like operating system. It was chosen as a test client for its support of IPv6 and support for both active and passive modes of FTP. Supporting rate limitations was also considered a plus in choosing it.

An example of FTP test can be seen in Listing 5.8. Berkeley Software Distribution (BSD) FTP implementation is used. In command option -4A forces use of IPv4

and ACTIVE mode, `-q 5` means timeout after 5 seconds if connection stalls. In `ftp://developer:cesdev@hosta1.demo.lte/ces.log` the `ftp://` is the protocol, the `developer:cesdev` the username and the password, the `@hosta1.demo.lte` the address of the server and the `/ces.log` location, in this case root, and file name `ces.log` retrieved.

```
root@public1:~# ftp -4A -q 5 ftp://developer:cesdev@hosta1.demo.lte/ces.log
Connected to hosta1.demo.lte.
220 (vsFTPd 3.0.2)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
200 Switching to Binary mode.
local: ces.log remote: ces.log
200 EPRT command successful. Consider using EPSV.
150 Opening BINARY mode data connection for ces.log (124977543 bytes).
100% |***************************|   119 MiB   76.47 MiB/s    00:00 ETA
226 Transfer complete.
124977543 bytes received in 00:01 (76.27 MiB/s)
221 Goodbye.
root@public1:~#
```

Listing 5.8: Example of FTP test

In the printout, we can see the process of FTP connection. In addition, it also shows information of the server we are connecting to, the size of the file and how long it took to download it. In this case, connection was initiated from public1 to hosta1, as can be seen from hostname at beginning in bash-prompt.

6 General testing of CESv1

In this chapter, we cover the work done on testing the older implementation of CES. First, we describe the division of scenarios we have chosen. Our work is focused on three areas performance, Denial-of-Service and the DNS. Each area is divided into sections where we describe the methodology, the results and provide some evaluation.

6.1 General

We chose 5 different scenarios for the testing of the system performance and operation.

Scenarios chosen for testing were:

1. CES as a NAT: Connecting from a private network to a public network where a CES operates as a NAT. This is the most common use case in the early adoption. For the user satisfaction, it is important that adoption of the CES does not cause any noticeable negative changes in connectivity.

2. CES as an RGW: Connectivity from a public network to a private network where a CES operated as a RGW with potential help from ALGs. Representing the legacy use case which would be common before wide-scale adoption of the CES. Shows potential of the immediate gains for regular users and is the scenario for the users requiring inbound connectivity.

3. Intra-CES: Connectivity from one host to another with both behind one CES in the same network. As the CES in the environment replaces the router the performance in this capacity is an important factor. This is a potential use case in the corporate networks.

4. Inter-CES: Connectivity from a host in one private network to a host in another private network with both utilizing CES. This obvious scenario where the CES operates as a CES.

5. Legacy: Connecting directly from a host to a different host in public network. This gives a comparative benchmark of the situation where CES is not present.

6.2 Performance testing

6.2.1 Methodology

Performance testing was originally performed with 3 different protocols in the 5 different scenarios presented above. Chosen protocols were FTP, SCP and HTTP, each being a popular point-to-point transport protocol. They were also already found in the environment. Of the three both SCP and FTP allow both download and upload of files in common use. Combinations of IPv4 and IPv6 and the passive and active modes were done with FTP. All scenarios presented above were tested.

For testing, a 100MB file was generated from `dev/urandom`. Random data was chosen as some protocols could utilize compression and raw throughput was the

desired metric. The file generated was distributed to all hosts. Also on each host, a new user was created as only the root accounts were present. This also allowed the use of non-anonymous users in the FTP test case. Two copies of the test file were stored on the machines: one in `/home/developer` for the FTP and the SCP test cases and one in `/usr/share/nginx/html` for the HTTP protocol testing.

Due to slow data rates and automation concerns only the download results were evaluated. This, however, should not have an effect on results as the initial tests point towards similar numbers to be achieve in the upload case. In the test automation, smaller file was chosen for faster test run times.

In addition, `iperf` was used to measure total throughput. The test was repeated 10 times using bash loop and results were collected. The average of measurements with the minimum and the maximum observed in each case can be found in Table 6.1.

6.2.2 Results

Results for `iperf` can be seen in Table 6.1. Observed results from other scenarios were in line with numbers listed here. We will describe the issues relating to them later.

Scenario:	Special:	Average:	Min:	Max:
NAT	`TCP-Splice` ON	NaN	NaN	NaN
NAT	`TCP-Splice` OFF	811	738	852
RGW	`TCP-Splice` ON	1,11	0,99	1,12
RGW	`TCP-Splice` OFF	800	731	842
Intra-CES	-	673	618	703
Inter-CES	Policy `rloc: ipv4`	533	517	554
Inter-CES	Policy `rloc: eth`	1,64	1,48	1,81
Legacy	-	1544	1050	1860

Table 6.1: Results of performance testing using `iperf` in Mbit/sec

The original test plans for the FTP included multiple variables: active or passive mode and IPv4 or IPv6. In addition to these `TCP-Splicing` and `Bot Detection` were variables. However most of the the tested combinations failed. As a result, we do not believe that presenting them here to be productive. The measurements were in line with presented ones when tested.

With the FTP and the HTTP testing, we found `tnftp` client to behave nicely with the CES and not to cause crashes, unlike other programs. `wget` and the SCP required manual limiting of bandwidth and thus made getting exact results harder. Also, the limited test file size made the observations too variable when the system was operating correctly.

`Stall IOError: [Errno 11] Resource temporarily unavailable` from the CES printout was observed to be a common failure mode. This error was noted when transfer rates for SCP were set to above 2,2 Mbit/s in inter-CES testing. Same

error in inter-CES was noted with `wget` and limit above 300 kB/s.

6.2.3 Evaluation

In general, we find the performance metrics to be acceptable where the system operates correctly. The legacy scenario gives us the inter-container average transfer rate of 1,544 Gbit/s which we consider the best case scenario for the test environment. Communication via the CES in the NAT and the RGW scenarios seem to have around 50% lower performance with the intra-CES networking being still slightly lower than this. As such there is a clear performance penalty with CES implementation in use.

We find the performance in the inter-CES scenario `ipv4-rloc` to be acceptable. The loss of performance due to two CES instances being involved. In which case two network translations will happen with both having overhead. The overhead from protocol itself is also to be considered. Overall the performance is acceptable and likely can be improved much further if desired.

The inter-CES communication with the policies using the `eth rloc` seems to be lower than the other scenarios. We believe this to be due to some sort of implementation error. Clear issues with implementation were also observed with the TCP splicing option turned on. From discussions with the developers, it was understood that there was an error in the implementation tested. However, due to moving to a new version this error was not corrected. The manifested in `IOError` likely linked to Ryu as the same message was also observed in tests performed in next section.

6.3 Denial of Service

6.3.1 Methodology

Denial of Service testing was performed with `hping3` tool generating `TCP SYN` packets to varying port numbers. The rate at which system was stable over extended periods of time was noted down. The tests were run from `public1` to `hosta1` with only `CESA` running. Effects of the build in settings of `TCP SPLICING` and bot detection was also used as parameters. The `hping3` tool is presented in more detail in Section 5.3. Packet size of 1460 bytes was used and the packet interval was varied until the stable operating point was found. In addition, we run the test with both the `CESA` and `CESB` running to observe if it would have any effects on results.

6.3.2 Results

Results for RGW can be seen in Table 6.2, where the interval is inter-packet delay in microseconds, the rate is packets per second, the time is runtime of test or until a crash was observed, the splicing and the detection corresponding to the state of respective settings and the result is observation of result of test.

Stable operation was observed at packet interval of 2000 microseconds in all scenarios. As seen the settings had an effect on results with the system performing

better without them. In addition to this, all test cases were tested with option –flood for tool to generate the maximum number of packets per second.

Interval:	Rate:	Time:	Splicing:	Detection:	Result:
2000	500	20 min	YES	YES	Stable
1500	666	7.5 min	YES	YES	Crash
1200	833	0.08 min	YES	YES	Crash
2000	500	24 min	YES	NO	Stable
1500	666	7.6 min	YES	NO	Crash
1200	833	0.013 min	YES	NO	Crash
2000	500	18 min	NO	YES	Stable
1500	666	14 min	NO	YES	Stable
1200	833	0.08 min	NO	YES	Crash
2000	500	5.5 min	NO	NO	Stable
1500	666	12.5 min	NO	NO	Stable
1000	1000	6.75 min	NO	NO	Stable
800	1250	0.013 min	NO	NO	Crash

Table 6.2: Results of stress testing RGW using `hping3`

In an additional test case where both `CESA` and `CESB` were running, we noted a crash after 30 seconds with the rate of 500 packets per second, the stable point for all setting combinations tested with single CES instance running. Further testing with multiple sources of attack pointed towards same hard limits listed here.

Implementation in the test environment was always stable to around 500 TCP SYN packets per second overall with the connections coming from public network to CES. With the maximum payload of the system at 1540 bytes per packet at 500 packets a second the throughput would be 752 kB/s.

In the NAT scenario we saw stability at around 7500 to 10000 microseconds intra-packet delay. Corresponding to 100 to 133 packets per second. In measuring intra-ces scenario we noted that inter-packet delay of 30000 microseconds resulted in crash after half a minute. With the delay of 50000 microseconds system was observed to be stable after 12,5 minutes. The values would translate to 33 packets a second and 20 packets per second.

For inter-CES scenarios, it appears that if `ipv4 rloc` is used once the connection is established the system can handle the maximum rate which hosta1 can send. However, before the connection is established DoS can affect the system. With `eth rloc` very poor performance was observed.

6.3.3 Evaluation and suggestions

We found Ryu-component of CES-implementation to be susceptible to crashing under certain kinds of load. During the testing, high CPU loads for container `cesa` was observed. In the tests running multiple instances of CES similar behaviour was noted.

This lead us to believe that the performance is constrained by CPU capacity. We also observed the possibility of memory leaks during testing as memory use on container kept increasing while running the test. Killing the CES process freed this memory so we believe CES to allocate large quantities of memory in certain situations. Extra CPU load from swapping and other kernel processes after memory was full is one likely cause of pushing CPU use to unsustainable level. This will result in a crash even if the system would otherwise be able to handle the load.

No significant correlation between packet sizes and throughput of packets were observed in additional testing. The most likely culprit is the inefficient handling of headers and routing inside the CES. The fact that host behind CES can also effectively DoS CES points towards implementation errors. The alternative is that the chosen technology is just not able to handle the traffic.

We do not believe that the tested implementation provides good enough performance. Some type of filtering using either IDS or firewall would likely mitigate part of the problems in attack scenarios. Iptables would be a suitable option for basic filtering. Other alternative is the use of an efficient SYN-proxy which would mitigate the problem, but at the same time option for splicing connections would be lost. Generally, a more performant system is needed.

6.4 DNS attacks

6.4.1 Methodology

Basic dictionary attacks against the DNS were performed utilizing `dnsrecon` tool, a multi-threaded DNS scan utility. A dictionary file of over 100000 words was used and seeded with know hostnames, as the hostnames were not in the dictionary initially. The attack was repeated against found hostnames to find possible services on hosts. Also, we tested blacklisting functionality with a dictionary containing multiple instances of known hostnames.

The goal of dictionary attack was to discover if some of the hosts behind RGW could be effectively discovered. In addition, we wanted to test the effectiveness of blacklisting techniques to discover if it was possible to circumvent blacklisting. This was achieved by getting host blacklisted by repeatedly performing a query for know FQDN. After the server was blacklisted we wanted to check how widely the blacklisting applied.

6.4.2 Results

We found that the dictionary attacks were effective in gathering information of the test-network and of possible services in it. We also noted that performance of the DNS server connected to CES was relatively low compared to the `bind9` running on the `public-dns`. We estimated `dnsrecon` with 6 threads to be able to query about 1250 domains per second against `public-dns` and 662 sub-domains of `hosta1`.

In addition, we tested the DNS blacklisting functionality of CES. After 5 queries the resolver was entered on the blacklist and no further `A` records were returned. However, we noted that replies with `RCODE 3` (`NXDOMAIN`) were still sent for names

that did not exist. Additionally, the blacklisting did not seem to affect sub-domains or `AAAA` records.

In some further testing in the environment, it was noted that `bind9` service running on `public-dns` did cache records with ttl value of 0 for approximately half a second and did not perform recursive queries. This was replicated by using `nc` command for trying to connect to non-open port on tested host. We also found the `dnsrelay` running on `cesa-ext` to be susceptible to DoS with a high level of traffic generation.

6.4.3 Evaluation

General performance seems to be bad for our chosen tool `dnsrecon`. The difference between direct queries and using `public-dns` as resolver is notable, however. As such we believe there is a performance issue with DNS relay in `cesa-ext`. A better solution and more efficient system is required.

Blacklisting behaviour is working on the basic level. However, an attacker aware of system behaviour could discern the existing host names even if they are blacklisted. In such an attack hacker would be able to track which domains are confirmed to not exist and ignore them while gathering a list of domains which no reply is generated for or one is provided. Additionally blacklisting should extend to sub-domains and `AAAA` which can be seen as an oversight.

One possibility to counter these types of attacks is to not send any `NXDOIMAIN` replies. This would prevent the attacker from discerning existing hosts. This is not desired behaviour from a DNS server but at least with blacklisted hosts this would be acceptable behaviour. If outright lying to another server is acceptable a `ICMP Destination Unreachable` is one option to prevent further traffic. We also suggest not to use common host names found in dictionaries available.

7 Fuzzing CES version 1 and CETP version 2

7.1 General

We understood that the most interesting use case of fuzz testing in CES was CETP. As such attacks against protocol were considered the most important surface of attack. This work could theoretically be divided in white-, grey- and black-box testing. However no clear authoritative specification of the protocol was available. This was to do with how the protocol and the implementation were still evolving. As a result, effectively we were limited to white-box testing. This was especially true later when certain implementation details needed to be verified from the source code of CES.

After inspection of the protocol format from available sources, we were able to make a choice between fuzz testing methodologies. We were also able to modify available Wireshark dissector for the new protocol, as described in Chapter 5. One of the key problems was choosing of the tooling. Due to CETP being transport layer protocol and not utilizing the common TCP or UDP our options were very limited inside the original design goal of building a system using Robot Framework. Effective options could be found if the protocol had used TCP or UDP. The simplest way to start testing was found to be simple random data approach with the recording of potential responses from the system. From just these responses multiple clear implementation errors were found.

Next, the author decided that even if it was possible to use existing implementation for CETP packet generation this was potentially wrong as any errors in the implementation could not be found. As such goal to create an alternative implementation for the simple generation, sending, receiving and parsing of CETP packets was set. The Robot Framework was chosen as the platform. For an extension of the past work as this was one of the original goals. For this purpose CETPLibrary described in more detail in page 38 was written.

7.2 Fuzz testing CETP version 2

For the generation of random data a tool called `sendip`. discussed in Section 5.3.6. was chosen. `sendip` allowed generation of IP packets containing pseudorandom data of select length with chosen header information such as the addresses and the protocol. Packets were generated once a second using the watch command. Traffic was monitored and gathered using Wireshark. The output of the CES was logged and any processing of packets was noted and tracked to the matching packet. The contents of packets noticed to generate response were copied from capture and saved in files for replay.

In Figure 7.1 we can see an example of this random test data case. At the top, we can see how the watch and the sendip is used to generate packets with the IP protocol value of 254 and a length of 480 bytes. Multiple captured packets can be seen in the Wireshark. The marked one has generated a reply from the CES. Dissect of that packet and of the reply packet can be also seen. It is noted that even if the

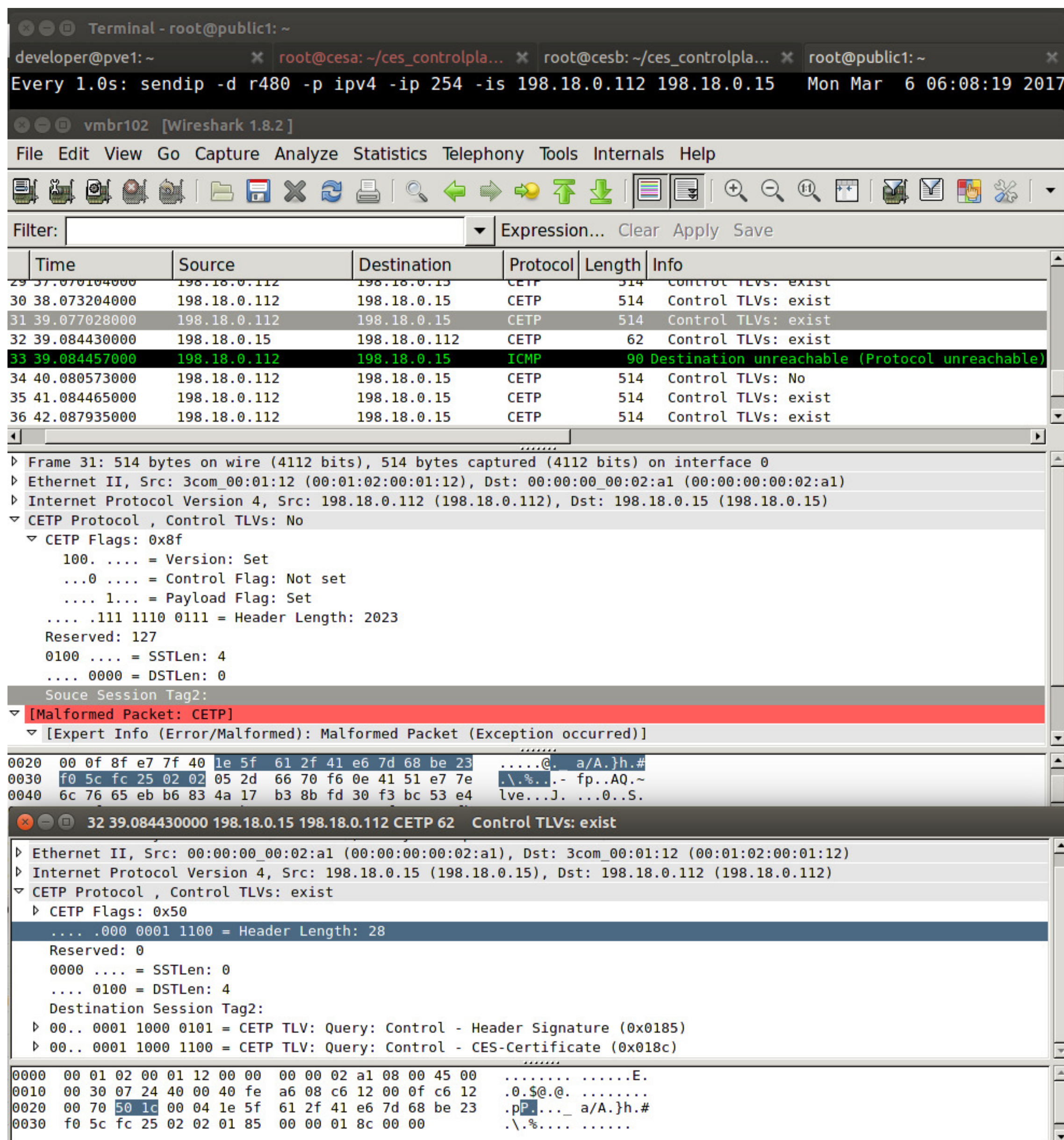

Figure 7.1: Use of `sendip` for random packet generation and Wireshark capture of process

content of the packet is malformed for dissector some of the values from the header have still been transformed to the reply. One example is the `SST` in the first packet which can be also seen as the `DST` in the reply packet. The capture also present many errors which we will discuss later.

7.2.1 Example of work: CETP header size field iterator

An example of work done can be found in Appendix C, where we test effects of different values of CETP packet header fields. The process generally follows the one

presented for `Perform Security Handshake` in Section 5.2.3. The main difference is that we iterate over multiple values and modify the `header length` field after we have generated the correct signature. Additionally, we read any potential output from CES process.

7.3 Found issues

Figure 7.1 presents some of the implementation issues found. For the CETP handling, we can see that the `version` value in the header is `100 (4)` and not `010 (2)` which is the value used by this version of the protocol. This proves that the tested implementation does not validate the version field. The next potential error in is how the `Control` Flag is not set, but the `Payload` Flag is and the CES expects the negotiation to follow from receiving this packet. Further on the `header length` field is set to `2023` even if the length of the whole received IP packet is only `514` bytes and CETP packet length is `480` bytes. Also, an issue with IP addresses is present in this capture.

Figure 7.2 presents one more implementation issue. It turned out that CES instances process and reply to any traffic received. In this capture we have replayed the packet from Figure 7.1 using `sendip`. The packet is sent to `198.18.0.15` an IP address belonging to `CESA`'s circular pool but is not an address advertised as a CES end point address. We can note that the MAC address of one host is `00:00:00:00:02:a1` which is also used in the packet. Also, we can see that we get a reply from IP address `.15` with the same MAC address, however, we get a second reply from `00:00:00:00:02:b1` the mac address of `CESB` from the same IP address `.15`. In this case, the ICMP replies are generated by the `public1` host as it does not have an open socket for the protocol as the packets are generated directly to the line. This behaviour was hard to track as the the virtual bridge involved eventually discovers the interface with the correct MAC and stops switching traffic to other hosts.

```
CETP Protocol , Control TLVs: exist
    CETP Flags: 0x16
    .... .110 1010 1000 = Header Length: 1704
    Reserved: 242
    0101 .... = SSTLen: 5
    .... 0000 = DSTLen: 0
    Souce Session Tag2:
    11.. 1001 0110 1100 = CETP TLV: Unknown (0xc96c)
        11.. .... .... .... = Operation: Ack to Response (0x0003)
        ..01 .... .... .... = Compatability: Set
        .... 10.. .... .... = Extension: Set
        .... ..01 0... .... = Group: RLOC types (0x0002)
        .... .... .110 1100 = Code: Set
        Length: 25983
        Value:
```

Listing 7.1: Dissection of CETP message causing KeyError

In the testing, it was also noted that the CES implementation does not handle wrong values gracefully, but instead throws a keyerror in the log. This happens when it receives a TLV with, for example, code value outside what it expects. An example of a packet causing this can be found in Listing 7.1 and the log output in Listing 7.2.

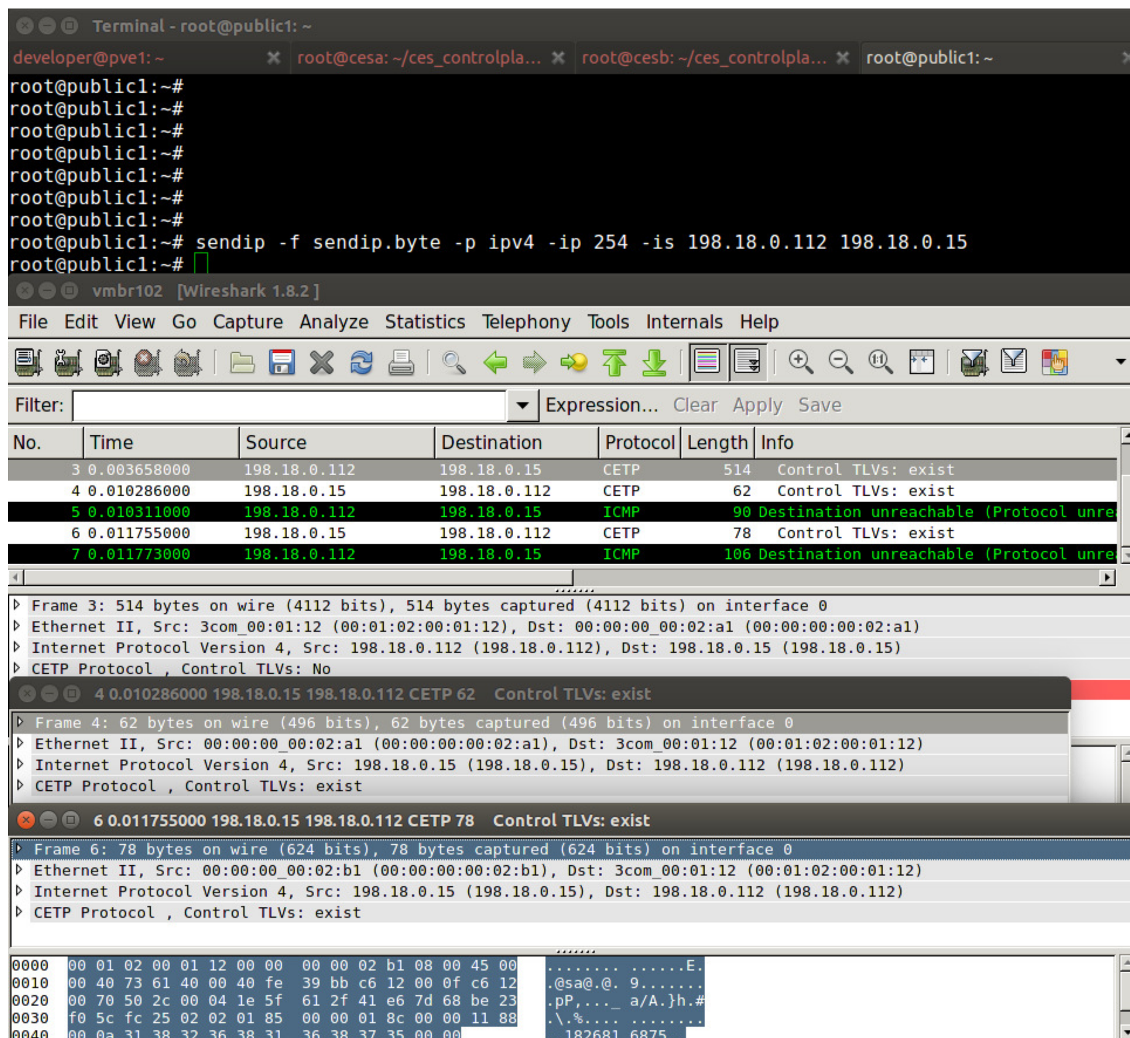

Figure 7.2: Use of `sendip` for packet replay and Wireshark capture of results

In the listing we can see the error being `108L` and we find the same parameters in code section of dissected message.

```
##############################
Processing a total of 1 Control TLV(s)
calling: Utils.exception_handler('Packet Relay: process packet', ex)
Exception in Packet Relay: process packet: 108L
Traceback (most recent call last):
  File "/root/ces_controlplane_pox/src/CETPTLV.py", line 353, in _unpack_value
    ret = self._unpack_rloc(value, group, code)

  File "/root/ces_controlplane_pox/src/CETPTLV.py", line 397, in _unpack_rloc
    code = TLV_CODE_RLOC[code]

KeyError: 108L
```

Listing 7.2: Truncated part of CETP log message in case of KeyError

7.3.1 Security handling

Minor details of implementation were also left unclear in previous work. This resulted in a need to find information in the code. One such detail was that SHA-digest signed in the signature TLV is a hex character string instead of bytes.

We found that using a incorrect certificate in the security handshake caused the connecting CES to get blacklisted. We believe that this is an effective avenue of attack as single spoofed packet can prevent further negotiation happening, thus preventing the legitimate instance from establishing required trust.

```
CETP Protocol , Control TLVs: exist
    CETP Flags: 0x53
    .... .011 1000 0100 = Header Length: 900
    Reserved: 0
    0001 .... = SSTLen: 1
    .... 0000 = DSTLen: 0
    Souce Session Tag: 0x31000000
    10.. 0001 1000 0101 = CETP TLV: Info: Control - Header Signature (0x8185)
    10.. 0001 1000 1100 = CETP TLV: Info: Control - CES-Certificate (0x818c)
    00.. 0001 1000 1100 = CETP TLV: Query: Control - CES-Certificate (0x018c)
    00.. 0001 1000 0101 = CETP TLV: Query: Control - Header Signature (0x0185)
[Malformed Packet: CETP]
```

Listing 7.3: CETP details for packet containing security requirements and incorrect header length

Additionally, it was noted that signatures of incoming packets were incorrectly handled. In Listing 7.3 we provide a dissection of send capture. Where the `Header Signature` provided is calculated from correct values for the header, but later the `length` field in the header is modified before sending the packet. Applying the code provided in Appendix C. In Listing 7.4 we can see the output from CES. Here we can note that the signature is accepted even if the information it hashes does not match the header.

```
##############################
Processing a total of 4 Control TLV(s)
Received TLVs #1. [info.cmp_notset.e_notset.control.headersignature -
(128 bytes of headersignature) ]
Received TLVs #2. [info.cmp_notset.e_notset.control.certificate -
(533 bytes of certificate) ]
Received TLVs #3. [query.cmp_notset.e_notset.control.certificate - ]
Received TLVs #4. [query.cmp_notset.e_notset.control.headersignature - ]
CES-certificate verification succeeded with CA
response_ctrl_certificate
response_ctrl_headersignature
CES Security policies are successfully negotiated
Sender '198.18.0.112' is admitted legitimate CES security requirements
Computing CETP HeaderSignature
Processing a total of 2 Control TLV(s)
Reply TLV #1. [response.cmp_notset.e_notset.control.certificate -
(530 bytes of certificate) ]
Reply TLV #2. [response.cmp_notset.e_notset.control.headersignature -
(128 bytes of headersignature) ]
start_transaction()
```

Listing 7.4: Output from CES when accepting CES security requirements

7.4 Evaluation issues

We believe that not handling the `version` and the `header length` correctly are clear implementation issues which should be corrected. The CES should always validate the version in a header it receives. Not doing the validations might mean failure of communicating with a host which has further capabilities, but at the same time supports the older version of the protocol.

The `header length` field is not compared to the length of received message. As the CES protocol lacks any error checking this means that incomplete packets are likely to be processed. This can cause complex issues in policy negotiation. Also from the discussion with the developers, we understood that header length is used in certain scenarios, but only as an upper limit in storage handling.

We discovered that the current implementation of CES effectively spoofs responses from any packets it receives. We can show this in two different scenarios. One is where traffic is received by the correct CES node, but to a wrong address and another where even the MAC address does not match the CES device. In the later case, the reply is produced from an IP address that does not even belong to host. In general, it is still an undesired behaviour for the CES to use address not configured for the purpose and advertised when queried.

Minor error in implementation was that certain error cases were not handled gracefully. Providing a value outside the expected range produced error. The preferred way would be to catch these errors and handle them better. This can be achieved easily by use of `try-catch` blocks.

We discussed the signature issue with developers and checked the code. We understood that signature handling for the security handshake is incorrect. The TLVs and Session tags are read from the header and then used in reconstructing of the message from which the signature TLV is removed and then a hash is calculated and verified against one provided. In further testing, it was found that also other fields like `version` and `reserved` were ignored in this process. The correct, implementation would read the whole message in memory and then remove `signature TLV` after which the signature hash would be calculated and verified.

From our understanding, the protocol lacks any sort of error-checking mechanism. This is a desired feature from transport layer protocols which CETP in this incarnation desires to be. For rudimentary error checking, the reserved field can be used or extra mandatory TLV can be introduced. This is especially needed if protocol aims to fully support IPv6 which does not have any error-checking but instead relies on further layers. Overall we were unable to locate any extreme exploits in the code. Further work could be done with the focus on lower level components as there is also potential to find exploits there. However, as the implementation is in Python we consider finding any errors unlikely.

7.5 Evaluation of tools and methodologies

In retrospect, the use of Robot Framework for testing CETP protocol itself was a poor choice. Robot Framework excels in the type of testing previously done and

automating some of the performed attacks. For those we found it sufficient and easy to use option. As such the work expanding it in those scenario can be found in Section 8.5,

However, we did not find its ecosystem to support the type of protocol testing needed in our case. Key problems were protocol's utilization of raw sockets and special cases of networking such as encapsulation. We found it to be possible to implement basic negotiation and message generation with our custom `CETPLibrary`. Due to the nature of the protocol, however, use of raw sockets was needed, which required running software with very high privileges. This itself made the code iteration unnecessarily complex. Furthermore, the effective message generation for the payload was found complex.

Some fuzz testing scenarios were found possible with the implemented system. It was also effective in the simple generation of messages for testing the edge-cases. However, automating testing was not possible as the only method of monitoring CES was to observe the log. Improved logging and communication of error cases could solve this problem. The quiet failing of the system was also an observed behaviour. With the potential of blacklisting in these cases complex iteration was not possible. With these types of systems better monitoring and integration of tools to the system under test is likely the best way of moving forward.

8 General testing of CESv2 and porting of RF tests

8.1 General

In Section 6.1 we presented 5 scenarios for testing. At the time of writing of this thesis CES to CES functionality was not available in the new version and as such many of the scenarios previously presented could not be used for testing. We present the scenarios and changes to them below.

1. CES as a NAT: Connectivity from a private network to a public network where the CES operates as a NAT. We repeated tests where applicable against the new RGW implementation.

2. CES as an RGW: Connectivity from a public network to private network where the CES operates as a PRGW with possible ALG. We repeated tests where applicable against the new RGW implementation.

3. Intra-CES: Connectivity from one host to another behind single CES with both hosts in the same private network. This scenario was not repeatable as the architecture of the test environment was changed and all hosts were simulated on a single host. However, this scenario should not differ from the Legacy scenario.

4. Inter-CES: Connectivity from one private network to another private network with both utilizing CES. This scenario was not repeated as CES to CES functionality was not available for testing in the new version.

5. Legacy: Connectivity directly from host to host in a network. We repeated tests where applicable to get the comparative baseline.

From the above we can see that the Inter-CES scenario is the only one not currently available. The Intra-CES scenario is possible, but current test environment is not set up the same way as the old one. However, we believe it to be sufficiently close to Legacy scenario.

8.2 Performance testing

8.2.1 Methodology

The test performed against old version were redone against the new iptables based version. Due to missing CES functionality only NAT, RGW and legacy tests were repeated. In addition to these, we present two legacy scenarios one between `public` and `router` and one between `public` and `gwa`. These tests aim to provide a picture of performance lost in general when iptables is used in routing. The `public - gwa` should offer performance equal or slightly lower to the old intra-CES scenario.

Tests were run using iperf with default settings. The command was repeated 10 times and results were collected. FTP testing also consisted of 10 runs and both active and passive mode were tested.

8.2.2 Results

Results from the iperf tests are presented in Table 8.1 with measurements in megabits per second. Considerable CPU loads were observed during the testing. All tests with iperf run without any noted errors.

Scenario:	Hosts:	Average:	Min:	Max:
NAT	public - test_gwa	30 070	28 100	31 300
RGW	test_gwa - public	30 670	29 500	31 800
Legacy	public - router	45 720	42 000	50 200
Legacy	public - gwa	36 340	34 400	37 500

Table 8.1: Results of performance testing using iperf in Mbit/sec

FTP test were repeated in NAT and RGW scenarios. Performance measured for RGW was an average of 959 MB/s, minimum of 855 MB/s and maximum of 1 016 MB/s in active mode. For NAT in passive mode results were an average of 952 MB/s, minimum of 876 MB/s and maximum of 1 014 MB/s. For RGW passive mode did not work and for NAT active mode did not work.

8.2.3 Evaluation

From the results presented above, we can see that the performance is excellent in all of the scenarios. The observed test result seems to be order of an order of magnitude higher than with those presented in Table 6.1. The RGW and NAT scenarios appear to offer same performance number, as the difference of 2% can be explained by high loads observed and general variability in the testing.

We can note there to be a general penalty from routing, as the best case scenario of two hosts communicating with each other is 45,7 Gb/s. The public - gwa was measured at 36,34 Gb/s for a route of public - router - gwa where traffic between hosts is routed at the router. Further performance penalty can be observed from the route public - test_gwa at rate of 30,07 GB/s where the longer route of public - router - gwa - test_gwa exist and additionally NATing is done.

The performance results from the FTP point towards 7,6 Gb/s speeds. Considering the small test file size and overheads involved with dealing with the filesystem, the performance is very good. The CES also appears to offer very good performance in test scenarios that are closer to reality. However, lack of ALG for the FTP is noted, which can be seen from the operation, where no extra connections from outside are possible. This will stop various protocols and applications from operating correctly with RGW.

For comparison of the best case legacy scenarios between CESv1, see Table 6.1, and CESv2 we have the numbers of 1 544 Mbit/s and 45 720 Mbit/s. From this we can calculate a ratio of 29,6 or about 30 between the environments. With this ratio we can do some comparisons between the systems. The NAT scenario where the old system offered 811,8 Mbit/s would equal about 24 Gbit/s rate, RGW would be the same. Compared to new versions circa 30 Gbit/s rates we can estimate the performance gain from the new system to be around 20%.

Overall, we believe that this version offers reasonable opportunities in real world use if the features of RGW are desired. We do not have the performance numbers from CES, but we are doubtful that they would be anywhere near the performance presented. This is due to planned encapsulation of data for CES-to-CES communication and overheads involved in this process. However, we do not believe the current implementation to be the bottleneck for any such system.

Iptables is mature, optimised and understood technology and as such we see its adoption as a very practical choice. Building on it allows utilization of much of the existing work.

8.3 Denial of Service

We repeated the experiments presented in Section 6.3 where applicable. Scenarios of RGW and NAT were used. `hping3` was used to generate traffic with –flood option from `public` to `test_gwa` and from `test_gwa` to `public`. No issues were noted during the running of the test.

We found the new version to handle SYN-flood with grace and continued to operate well. We believe this to be due to SYN-proxy utilized. Further on, the design using iptables to handle the traffic is also good, as the userspace component is loaded only by the new connections. Limiting the DNS rate also is working and effective way for stopping DoS. As a result we believe that the system should not face loads it cannot handle gracefully.

8.4 DNS attacks

DNS attacks presented in Section 6.4 were repeated. However, the new system does not have the blacklisting functionality of older version. As such the correct operation of blacklisting could not be tested. Major DNS traffic was generated with dnsenum tool.

The new iptables based RGW has a DNS rate limitation system and this was found operating correctly when tested. During generation of DNS traffic, the system was found stable under the load of thousands of queries per seconds. With further and more extensive DNS query generation we found the system to still be stable. As such we do not expect the system to be susceptible to DNS DoS.

Repeating the test of same queries with short timing differences against `dnsmasq` did not produce similar response as `bind9` did. Both queries were recursively handled and resulted in new allocations from the circular pool. It appears that `dnsmasq` has the desired behaviour in our test environment.

Vulnerability in dnsmasq server used to effectively route queries at router was found. By repeating zone transfer queries from public it was possible to crash the dnsmasq. This was believed to be due to the interaction between dnsmasq and bind9 running on the same host where these zone transfers were forwarded. The issue was not considered critical or possible in systems that do not use this combination.

8.5 Porting of Robot Framework test cases

At midway of the thesis process, the iptables version of CES was deemed testable. At this point, only RGW functionality was testable. As the old version was not going to be developed further, it was decided to move efforts to testing of the new version. For this, the existing test cases were to be ported over to the new version.

In this process, it was found that existing tests were largely not suitable to be used in the new version, being extraneous or too specific. As such starting from fresh and only bringing basics and some of the specific applicable test cases over was considered as the best option

We believed that test suites should test both the environment, orchestrated with scripts and functionality of RGW and CES. As such we did not want to limit test suites just to testing of CES as an issue in the environment would likely have effect on tests too.

In general, the current approach in testing is to cover needed regressions to make sure that, as the system is updated and build up, no changes affect the operation. In the process there has been multiple occasions where incorrect operation or breaking changes in the test environment was detected.

The main goal of the new test design was to heavily rely on building things from smaller keywords to larger test keywords so the code could be reused. Additionally, we were able to leverage the design of the Robot Framework by setting our own parameters for different system behaviour. Two more interesting use cases of these were a parameter for either running all of the test cases in a test suite within a single state or restarting the CES instance after each test case. With this, we also designed a way for us to log related output from the CES to each test case.

8.5.1 General structure

In the design of the tests, it was deemed that some logical division was needed and we considered folders to be the suitable option for this. Scenarios presented deemed the clearest division of responsibility. In addition to scenarios, it was considered good practise to separate the used keyword libraries and resources. The following folder structure was chosen:

1. Libraries: libraries, resource and keyword files.
2. Connectivitytest: test suite for testing connectivity to general Internet. Additionally could be expanded to test needed to run against environment.
3. NATTest: test suites for testing RGW's NAT component with multiple protocols, including general connectivity.

4. RGWTest: test suites for testing RGW's RGW component with multiple protocols.

In addition, each test written was tagged with relevant tags such as nat, wget, hosta, http and public. These tags usually contained the scenario, hosts involved, program used and relevant protocol.

8.5.2 Libraries files

Libraries folder contains the generic keywords used in multiple test suites. This allows modification in single place to change behaviour in all of the test cases. Each general use case is separated into an individual robot file.

In Libraries global_lib.robot contains keywords for logging in on host either using username and password or RSA-keys. In addition, it contains keywords for starting and stopping CES instances and checking its state. It also provides keywords for logging CES output. Global_variables.robot contains variables for storing names, ip addresses and other information relevant to the whole environment.

Utility_lib.robot contains general keywords for executing commands on the command line, in various ways, and checking the status of these executions. dns, ftp, iperf and ncat_lib.robot contain generic keywords for more specific test and general tests keywords for launching client and servers on hosts provided as arguments.

8.5.3 Connectivitytest test suites

Connectivitytest contains legacytest.robot test suite. The test suite contains tests for general connectivity to general Internet. This suite exist to prove that environment works correctly and failures in other suites such as RGW are not errors in RGW, but of the environment itself. Example found of such case in testing is missing NATing in the host VM.

The tests included pinging of well know hosts such as Google. Using various tools such as wget and curl to retrieve websites from general Internet. Downloading file using ftp, doing DNS query and performing repository update.

8.5.4 NATTest test suites

NATTest contains dhcptest.robot, legacytest.robot, nattest.robot and nstest.robot test suite files. The aim of these test suites is to test that NAT component of RGW allows correct communication with hosts inside the environment and the general Internet. As RGW should not have negative effects on the host.

Suite legacytest.robot contains the same test suite as legacytest.robot in Connectivitytest. This allows to test that RGW does not stop any normal communication to general Internet. Suite nattest.robot contains various tests for communicating with mainly host public inside the environment. Correct operation of various protocols and the firewall rules in RGW for outbound traffic are tested.

Suites `dhcptest.robot` and `nstest.robot` are currently bare, but allow building on the testing of network namespaces. Network namespaces were chosen to allow simulating multiple hosts using a single container. `Dhcptest.robot` contains initial support for testing namespaces on `test_gwa` querying for configuration via DHCP. `Nstest.robot` contains test for custom policies uploaded from machine running to `gwa` operating correctly and allowing the namespaces to be communicated to. In addition the policy uploaded contains some rules not covered in standard policies.

8.5.5 `RGWTest` test suites

`RGWTest` contains test suites for testing RGW components operation. It should allow seamless connectivity from outside to host behind it using FQDNs. RGW is also going to contain various ALGs and HTTP proxy.

Test suite `rgwtest.robot` contains individual test cases to assert that various protocols and applications such as ping, http, `iperf`, `iperf3`, dns, tcp, udp and sctp operate correctly. Also rules for inbound traffic with mentioned protocols are tested. `FTPTest.robot` is an extension for this test suite containing the test suite for FTP. More clear direction if test suite should be separated into multiple suites further or if `FTPTest.robot` should be merged back into `rgwtest.robot` should be discussed in future.

`NSPolicyTest.robot` is a special test suite for testing network namespaces like `nstest.robot`. Its main goal is to make sure that services such as `nginx`, a web server, can be run inside a namespaces and they can be communicated to. In addition, the test its checks that group policies defined for hosts in the policy files are operating correctly and only allow defined type of traffic through.

8.6 Future work

The author recommends expanding the above with test suites relating to the CES operation when the features in the new version are available and the interactions between components are testable. The author's opinion would be the division of the test suites based on layers in the case of CES. Host to host connectivity over CES would be one clear scenario. Correct usage of certificates and other such issues in general security can also be considered as a separate scenario. The use of wrong certificate was one test in the old framework. From the discussions about the new version, there is a new components being developed. These are likely to offer new and interesting opportunities for fuzz testing.

9 Conclusions

We tested two different versions of CES. For this purpose, we established 5 different scenarios. We believe these scenarios to provide s good framework for testing and believe that they should be reused in the future. We found the older version to offer acceptable implementation in many areas, but we also found multiple faults which could have been corrected. We believe that the new version of CES offers s great base to build on. For best product, the blacklisting should be reintroduced in some form with the ALGs for relevant protocols.

In the general operation, the version 1 seems to perform well. However, we noted that there are some scenarios where some type of implementation error hinders the work. In future, we hope that team can iterate over these sort of issues in their work. We found the DNS blacklisting to be an interesting idea and believe the found issues of blacklisting not applying to all types of traffic to be simple fixes. However, the most critical issue of the system from our viewpoint is the susceptibility to DoS attacks. We found the system to be extremely weak against even relatively low rate attacks in many scenarios. However, we were genuinely surprised by the scenario involving established CES data plane traffic, which offered very promising results.

In our work with the CETP protocol, we found multiple issues with implementation. We observed both clear lack of validation the received data and implementation details that could be handled better. We believe the most critical issue is the implementations lack of traffic filtration and using any destination address of a received packet as the source address in the replies. The protocol itself lacks any error detection facilities, which combined with the handling of erroneous inputs is an issue that we hope will be rectified in the future versions.

CES version 2 showed a lot of promise when we performed the test done against the old versions. The performance was excellent and we did not find any effective attacks against the system. With the reintroduction of some of the ideas from version 1 we believe that it can be developed to an effective system. On the environment side, we found the orchestration system very easy and pleasant to work with. It could allow great options in the future for automatizing builds and testing.

We hope that our work in building the testing framework for CES will provide the direction forwards. The Robot Framework combined with above-mentioned orchestration system should allow easy transfer and testing of the system around for development. Our work should provide a logical base to build up from when more features become available and need testing. For this purpose, we believe that Robot Framework is an excellent choice and believe that it will serve well.

In the field of fuzz, testing we found the Robot Framework to be a sub-optimal solution. While it was possible to generate needed signalling traffic the nature of the protocol made testing unnecessarily complex. Extension to full workable implementation is likely possible, but suboptimal. Additionally, we did not find the system to offer good enough debugging facilities for effective testing of the implementation. For further work in this field, I believe some sort of facilities should be provided or at least have access to improved logging of critical failures. With proper facilities use of some other system for testing could provide interesting results.

Another issue with our work was the lack of proper documentation. We believe that this type of work should not be done before there is a specification available and implementation is near completion. However, this type of testing should be done as critical errors in design or implementation are likely to be found and any such issues should be corrected in an iterative process.

Looking at the process in retrospect, we believe that we started this work at a wrong time. At starting of the work the old system was no longer under development, but the new version was promised to be available soon very soon. This itself was clearly a bad starting point, testing a system that is going to be replaced does not have great value. This is especially valid in our case as the design of the new CETP protocol itself has been entirely changed in nature from a custom binary protocol operating over IP to a text-based protocol over TCP utilizing standard protocols. At a time of writing this the author's understanding is that work on the new protocol and implementation is still under way. We are however thankful that new version offering interesting new challenges and opportunities became available, albeit late, but at the midway of the thesis process and it was decided to focus on the new version.

References

[1] R. Braden, "Requirements for internet hosts - communication layers," STD 3, RFC Editor, October 1989.

[2] I. Standardization, "Iso/iec 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model," tech. rep., 1996.

[3] J. Postel, "Internet protocol," STD 5, RFC Editor, September 1981.

[4] J. Postel, "Dod standard internet protocol," RFC 760, RFC Editor, January 1980.

[5] S. E. Deering and R. M. Hinden, "Internet protocol, version 6 (ipv6) specification," RFC 2460, RFC Editor, December 1998.

[6] K. Nichols, S. Blake, F. Baker, and D. L. Black, "Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers," RFC 2474, RFC Editor, December 1998.

[7] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," RFC 3168, RFC Editor, September 2001.

[8] E. Alliance and B. Kohl, "Ethernet jumbo frames," 2009.

[9] J. Touch, "Updated specification of the ipv4 id field," RFC 6864, RFC Editor, February 2013.

[10] J. Postel, "Assigned numbers," RFC 790, RFC Editor, September 1981.

[11] V. Fuller and T. Li, "Classless inter-domain routing (cidr): The internet address assignment and aggregation plan," BCP 122, RFC Editor, August 2006.

[12] M. Cotton, L. Vegoda, R. Bonica, and B. Haberman, "Special-purpose ip address registries," BCP 153, RFC Editor, April 2013.

[13] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger, "Iana-reserved ipv4 prefix for shared address space," BCP 153, RFC Editor, April 2012.

[14] J. McCann, S. E. Deering, and J. Mogul, "Path mtu discovery for ip version 6," RFC 1981, RFC Editor, August 1996.

[15] M. Mathis and J. Heffner, "Packetization layer path mtu discovery," RFC 4821, RFC Editor, March 2007.

[16] J. Postel, "Transmission control protocol," STD 7, RFC Editor, September 1981.

[17] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry," BCP 165, RFC Editor, August 2011.

[18] J. Postel, "User datagram protocol," STD 6, RFC Editor, August 1980.

[19] J. Postel, "Internet control message protocol," STD 5, RFC Editor, September 1981.

[20] F. Gont and C. Pignataro, "Formally deprecating some icmpv4 message types," RFC 6918, RFC Editor, April 2013.

[21] "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture," tech. rep., June 2014.

[22] D. C. Plummer, "Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware," STD 37, RFC Editor, November 1982.

[23] P. Mockapetris, "Domain names - concepts and facilities," STD 13, RFC Editor, November 1987.

[24] P. Mockapetris, "Domain names - implementation and specification," STD 13, RFC Editor, November 1987.

[25] R. Elz and R. Bush, "Clarifications to the dns specification," RFC 2181, RFC Editor, July 1997.

[26] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Message syntax and routing," RFC 7230, RFC Editor, June 2014.

[27] J. Postel and J. Reynolds, "File transfer protocol," STD 9, RFC Editor, October 1985.

[28] P. Srisuresh and M. Holdrege, "Ip network address translator (nat) terminology and considerations," RFC 2663, RFC Editor, August 1999.

[29] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session traversal utilities for nat (stun)," RFC 5389, RFC Editor, October 2008.

[30] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," RFC 5766, RFC Editor, April 2010.

[31] J. Rosenberg, "Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols," RFC 5245, RFC Editor, April 2010.

[32] R. Droms, "Dynamic host configuration protocol," RFC 2131, RFC Editor, March 1997.

[33] L. Rudman and B. Irwin, "Characterization and analysis of ntp amplification based ddos attacks," in *Information Security for South Africa (ISSA), 2015*, pp. 1–5, Aug 2015.

[34] S. Josefsson, “The base16, base32, and base64 data encodings,” RFC 4648, RFC Editor, October 2006.

[35] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing.* John Wiley & Sons, 2011.

[36] J. Llorente Santos, “Yksityisen alueen yhdyskäytävä; private realm gateway,” g2 pro gradu, diplomityö, 2012.

[37] M. Pahlevan, “Signaling and policy enforcement for co-operative firewalls,” g2 pro gradu, diplomityö, 2013-04-22.

[38] H. Kabir, “Security mechanisms for a cooperative firewall,” g2 pro gradu, diplomityö, 2014-03-31.

[39] R. Kantola, J. Llorente Santos, and N. Beijar, “Policy-based communications for 5g mobile with customer edge switching,” *Security and Communication Networks*, 2015.

[40] H. Kabir, R. Kantola, and J. L. Santos, “Security mechanisms for a cooperative firewall,” in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pp. 814–818, Aug 2014.

[41] J. L. Santos and R. Kantola, “Transition to ipv6 with realm gateway 64,” in *2015 IEEE International Conference on Communications (ICC)*, pp. 5614–5620, June 2015.

[42] H. Kabir, J. L. Santos, and R. Kantola, “Securing the private realm gateway,” in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 243–251, May 2016.

[43] L. Pekka, “Data-driven and keyword-driven test automation frameworks,” *Master’s Thesis, Software Business and Engineering Institute, Department of Computer Science and Engineering, Helsinki University of Technology*, 2006.

[44] “Robot framework user guide.” Accessed: 2017-03-20 http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html.

[45] K. Amir, “Automated testing of customer edge switching architecture,” master’s thesis, 2016-10-27.

A Appendix: CETPLibrary documentation

```
CETPLibrary
===========
Scope:            test case
Named arguments:  supported

Library of objects and functions for use of testing CETP remotely.

Append Certificate Tlv
----------------------
Arguments:  [certFileName, tlvs=None]

Appends certificate to tlv list.

Append Signature Tlv
--------------------
Arguments:  [header, keyFileName, tlvs=None]

Appends signature to provided tlv list.

Append Tlv
----------
Arguments:  [tlv, tlvs=None]

Appends TLV dictionary to given list of TLVs, if no list provides creates new one

Bytes To B64
------------
Arguments:  [buff]

Bytes To CETP Message
---------------------
Arguments:  [packet]

Attempts convert a received packet to header and TLV list

Bytes To TLVs
-------------
Arguments:  [buff, offset]

Attempts to convert given string with offset to multiple TLVs

Fuzz Header Field
-----------------
Arguments:  [header, field]

Fuzzes given field in header

Fuzz Tlv Field
--------------
Arguments:  [tlv, field]

Fuzzes given field in tlv

Header To Base64
----------------
Arguments:  [header]

Returns base64 encode of the header

Header To Bytes
---------------
Arguments:  [header]

Converts the given header to struct presented as string suitable to be send
```

```
Modify Field Header
-------------------
Arguments:  [header, field, value]

Modifies single field in header dictionary

Modify Field Tlv
----------------
Arguments:  [tlv, field, value]

Modifies single field in tlv dictionary

Modify Header Value B64
-----------------------
Arguments:  [header, b64]

Modifies value field in header dictionary
to provided b64 encoded string

New Header Dictionary
---------------------
Arguments:  [version=3, control=False, payload=False,
                         length=0, SST=, DST=, value=]

Returns a dictionary containing values relevant to header.
Fields: control, payload, length(size of message), SST, DST, value

New Socket
----------
Arguments:  []

Creates a new raw IP socket

New Tlv Dictionary
------------------
Arguments:  [operation=0, compatibility=0, extension=0, group=0,
                        code=0, TLVvalue=]

Returns a dictionary containing values relevant to single TLV.
Fields: operation, compatibility, extension, group, code and TLVvalue

New Tlvpayload Dictionary
-------------------------
Arguments:  [operation=0, compatibility=0, extension=0, group=0,
                code=0,mobility=0, DQ=0, TTL=0, proto=0, payload=]

Returns a dictionary containing values relevant to single payload TLV.
Fields: operation, compatibility, extension, group, code,
mobility, DQ, TTL, Proto and TLVvalue

Padded Len
----------
Arguments:  [len]

Calculates padded length.
Returns same or next number divisible by 4.

Query NAPTR
-----------
Arguments:  [hostname]

Queries given hostname for NAPTR records.

Randomize List
--------------
Arguments:  [tlvs]
```

```
Suffles the given list

Receive Via Socket
------------------
Arguments:  []

Receive up to 2048 byte from scoket

Seed Random
-----------
Arguments:  [seed]

Seeds then random number generator with provided seed.

Send B64 Via Socket
-------------------
Arguments:  [src, dst, b64]

Sends IP datagram with decoded byte64 string as payload.

Send Via Socket
---------------
Arguments:  [src, dst, buff]

Sends IP datagram via socket with provided buffer as payload.

Signed B64 Message
------------------
Arguments:  [header, keyFileName, tlvs=None]

Returns a byte64 encoded header with signature TLV appended.

Tlv To Bytes
------------
Arguments:  [tlv]

Converts given TLV to struct presented as base64 encoded string.

Tlvpayload To Bytes
-------------------
Arguments:  [tlv]

Converts payload TLV to struct presented as string to be send

Tlvs To Bytes
-------------
Arguments:  [tlvs]

Convert multiple TLVs from a list to struct presented
as string to be send.

Update Header Length
--------------------
Arguments:  [header]

Updates the header length field to match length of content.
```

B Appendix: CETP_lib.robot

```
*** Settings ***
Library  Remote  http://192.168.254.112:25400

*** Variables ***

#Constants for use of CETP

#Operations
${OPEQuery}  ${0}
${OPEResponse}  ${1}
${OPEInfo}    ${2}

#"Compatibility"
${CMPNava}  ${0}
${CMPAva}  ${1}
${CMPnreq}  ${2}
${CMPreq}  ${3}

#Extensions
${Ext}  ${0}

#Group
${GRPId}  ${0}
${GRPPayload}  ${1}
${GRPRloc}  ${2}
${GRPControl}  ${3}
${GRPMobility}  ${4}

#C RLOC
${CRLOCIPv4}  ${1}
${CRLOCIPv6}  ${2}
${CRLOCEth}  ${3}

#C Payload
${CPLIPv4}  ${1}
${CPLIPv6}  ${2}
${CPLEth}  ${3}

#C Control
${CCTRLCESID}  ${0}
${CCTRLDStep}  ${1}
${CCTRLTerm}  ${2}
${CCTRLWarn}  ${3}
${CCTRLRateLim}  ${4}
${CCTRLHeadSig}  ${5}
${CCTRLCACES}  ${6}
${CCTRLCAEP}  ${7}
${CCTRLACK}  ${8}
${CCTRLTAG}  ${9}
${CCTRLTTL}  ${10}
${CCTRLPOW}  ${11}
${CCTRLCERT}  ${12}

#C ID
${CIDFQDN}  ${1}
${CIDMAID}  ${2}
${CIDMOC}  ${3}
${CIDHASH}  ${4}
${CIDTEMP}  ${5}
${CIDRandom}  ${6}
${CIDBBBBid}  ${7}
${CIDMSisdn}  ${8}
```

```
*** Keywords ***
Send message
    [Arguments]  ${header}  ${src}  ${dst}
    ${bytes}  Header to bytes  ${Header}
    ${b64}  bytes to b64  ${bytes}
    Send b64 via socket  ${src}  ${dst}  ${b64}

Add TLVs
    [Arguments]  ${header}  ${tlvs}
    ${tlvsBytes}  TLvs to bytes  ${tlvs}
    ${header}  Modify Field Header  ${header}  value  ${tlvsbytes}
    ${header}  Update Header Length  ${header}
    [Return]  ${header}


Perform Security Handshake
    [Arguments]  ${src}  ${dst}  ${certFileName}  ${keyFileName}
    ${header}  New Header Dictionary  version=${2}
    ...  control=${TRUE}  payload=${FALSE}  SST=1
    ${tlvHSReq}  New TLV Dictionary  operation=${OPEQuery}
    ...  group=${GRPControl}  code=${CCTRLHeadSig}
    ${tlvCertReq}  New TLV Dictionary  operation=${OPEQuery}
    ...  group=${GRPControl}  code=${CCTRLCert}

    ${tlvs}  Append Certificate TLV  ${certFileName}
    ${tlvs}  Append TLV  ${tlvCertReq}  ${tlvs}
    ${tlvs}  Append TLV  ${tlvHSReq}  ${tlvs}

    ${header}  Add tlvs and signature  ${header}  ${tlvs}  ${keyFileName}
    Send Message  ${header}  198.18.0.112  198.18.0.10
    ${packet}  receive via socket
    ${header}  ${tlvs}  Bytes to CETP Message  ${packet}

Setup
    New Socket

Add tlvs and signature
    [Arguments]  ${header}  ${tlvs}  ${keyFileName}
    ${header}  Add TLVs  ${header}  ${tlvs}
    ${tlvs}  Append Signature TLV  ${header}  ${keyFileName}  ${tlvs}
    ${header}  Add TLVs  ${header}  ${tlvs}
    [Return]  ${header}
```

C Appendix: Test case for RF CETP Security Handshake with header size-field iteration

```
*** Settings ***

#Relevant parameters to be used with CETP protocol dictionaries
#Relevant IP addresses and Domain names
#Helper methods common for CES testing
Resource  ../global_lib.robot
Resource  ../global_variables.robot
Resource  ../CETP_lib.robot

Library  SSHLibrary

#Remote library on public host1
Library  Remote  http://192.168.254.112:25400
#Library  CETPLibrary
Suite Setup  New Socket

*** Variables ***
${keyFileName}  /root/CETPLibrary/public1.demo.lte.private
${certFileName}  /root/CETPLibrary/public1.demo.lte.certificate.der

*** Test Cases ***
Handskahe
    [Documentation]  Simple policy test
    ${conn}  Log In   ${cesa}  ${username}  ${password}
        write  tail -F /root/ces_controlplane_pox/src/ces.log
    ${output}  read
    : FOR  ${INDEX}  IN RANGE  13  1000
    \  ${header}  New Header Dictionary  version=${2}
    \  ...  control=${TRUE}  payload=${FALSE}  SST=1
    \  ${tlvHSReq}  New TLV Dictionary  operation=${OPEQuery}
    \  ...  group=${GRPControl}  code=${CCTRLHeadSig}
    \  ${tlvCertReq}  New TLV Dictionary  operation=${OPEQuery}
    \  ...  group=${GRPControl}  code=${CCTRLCert}

    \  ${tlvs}  Append Certificate TLV  ${certFileName}
    \  ${tlvs}  Append TLV  ${tlvCertReq}  ${tlvs}
    \  ${tlvs}  Append TLV  ${tlvHSReq}  ${tlvs}

    \  ${header}  Add TLVs  ${header}  ${tlvs}
    \  ${tlvs}  Append Signature TLV  ${header}  ${keyFileName}  ${tlvs}
    \  ${header}  Add TLVs  ${header}  ${tlvs}

    \  ${header}  Modify field Header  ${header}  length  ${INDEX}
    \  Send Message  ${header}  198.18.0.112  198.18.0.10
    \  ${packet}  receive via socket
    \  ${header}  ${tlvs}  Bytes to CETP Message  ${packet}

    \  Switch Connection  ${conn}
    \  ${output}  Read  loglevel=WARN
```