Samuli Kalliomaa

IP reputation based IPS-system for Unix-routers

Metropolia University of Applied Sciences
Bachelor of Engineering
Information and Communication Technology
Bachelor's Thesis
9 May, 2023

Abstract

Author:	Samuli Kalliomaa
Title:	IP reputation based IPS-system for Unix-routers
Number of Pages:	49 pages
Date:	9.5.2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	IoT and Cloud Computing
Supervisors:	Tapio Wikström, Senior Lecturer

The ever-increasing number of connected devices in home-networks has led to a growing concern of home network security. A typical set of security measures of the average user are lacking, leaving these networks vulnerable to cyber-attacks. This thesis-work was meant to explore the possibility of having the home network gateway check all traffic for known bad IP addresses automatically, allowing for real-time blocking of malicious traffic.

Thesis visits the working principles of antiviruses and firewalls and proposes that IP-address reputation data could be utilized to strengthen the network security significantly – if this could be achieved without affecting the network throughput and stability at a noticeable degree.

Thesis work was carried out as a personal project, and its goals were to produce a proof-of-concept of the software described above in addition to experiment on programming network-software using the Go-programming language, which it was built with.

The project produced a proof-of-concept of a free-of-charge IP reputation based IPS-system for UNIX-routers using Go that does not seem to slow the internet connection at a noticeable level in normal browsing, this should be further studied by testing the effect over longer period. Added latency for opening a new connection turned out to be between 100-200ms, which might be unbearable in some demanding use cases such as competitive gaming.

Keywords: Network Security, UNIX, Go, Golang, Linux, IP, IPS, IoT

Tiivistelmä

Tekijä:	Samuli Kalliomaa
Otsikko:	IP mainepohjainen IPS-järjestelmä Unix-reitittimille
Sivumäärä:	49 sivua
Aika:	9.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja Viestintätekniikka
Ammatillinen pääaine:	Verkot ja pilvipalvelut
Ohjaajat:	Tapio Wikström, Lehtori

IoT-laitteiden yleistyminen on tuonut uusia haasteita kotiverkkojen tietoturvallisuuteen. Tyypillisen kotiverkon turvamekanismit torjuvat yleisimmät uhat, mutta niissä on merkittäviä sokeita pisteitä. Tämän insinöörityön tarkoitus oli tutkia, josko reitittimessä IP-osoitteiden maineeseen perustuva tunkeutumisenestojärjestelmä olisi käytännöllinen ratkaisu kotiverkkoturvallisuuden parantamiseen.

Työ tutustui antivirusten ja palomuurien turvallisuusmekanismeihin ja esitti, että julkisesti saatavilla olevien IP-osoitteiden mainetietojen hyödyntäminen kotiverkon turvallisuuden parantamiseen olisi käytännöllistä, mikäli tämä onnistutaan toteuttamaan siten, ettei verkon nopeus ja luotettavuus mainittavalla tavalla kärsi.

Työ tehtiin henkilökohtaisena projektina ja sen tavoitteina olivat konseptitodistuksen tuottaminen ylläkuvatusta ohjelmistosta sekä verkko-ohjelmistojen ohjelmointiin tutustuminen Go-ohjelmointikielellä.

Projekti tuotti konseptitodistuksen ilmaisesta IP-mainepisteytykseen perustavasta tunkeutumisenestojärjestelmästä UNIX-reitittimille. Järjestelmä ei ensimmäisten testien perusteella hidastanut internetyhteyttä merkittävästi, mutta tämä vaatii pidemmän aikavälin seurantaa tarkemman kuvan saamiseksi. Uuden yhteyden avaamiseen järjestelmä aiheutti noin 100-200 ms ylimääräisen latenssin, tämä saattaa vaikuttaa tiettyihin vaativiin käyttötarkoituksiin, kuten videopelaamiseen.

Avainsanat:	tietoverkko, tietoverkkoturvallisuus, UNIX, go, golang, Linux, IoT, IPS, IP

Contents

List of Abbreviations

LAN:	Local Area Network
OSI:	Open Systems Interconnection
IP:	Internet Protocol
TCP:	Transmission Control Protocol
UDP:	User Datagram Protocol
IOC:	Indicator of Compromise
SOC:	Security Operations Center
DN:	Domain Name
DNS:	Domain Name Service
HTTP:	Hypertext Transfer Protocol
HTTPS:	Hypertext Transfer Protocol Secure
GB:	Gigabyte
CPU:	Central Processing Unit
RAM:	Random Access Memory
IPS:	Intrusion Prevention System
API:	Application Programming Interface

I/O:	Input/Output
VS:	Visual Studio
CLI:	Command Line Interace
WIP:	Work In Progress

1 Introduction

Demand for smart-home appliances has been rapidly growing for the past decade. According to earthweb.com In 2021, there were 258.54 million smart homes globally. As of the time of writing, 23% of broadband household in US had a minimum of three smart-home-appliances and alarmingly 40.8% of smart homes globally have one or more susceptible connected device that an attacker could be able to use as a gateway into the home-network. [1]

For the forementioned surplus of devices, there are quite a few ways to ensure of the confidentiality, integrity and availability of all traffic flowing in the typical modern home-network. Firewalls and antiviruses are well known tools for home-network security, but there is a number of potential security-threats they do not help with. [2] And especially when it comes to smart-home devices – it all might just fall on the router's firewall's ability to block threats, which can be woefully inadequate. [3]

This thesis-project aims to produce an open-source intrusion prevention system (IPS) that utilizes IP-address- reputation as means to identify and block malicious traffic.

2 Home-network security monitoring

Defence-in-depth is a military concept that was coined to a cybersecurity context by the U.S. National Security Agency. It refers to layering defences to mitigate risk hoping that if ill-meaned actor were to penetrate some layers of the defence – other layers might still prevent the intrusion. [4]

The following sections introduce the typical security tools of a home-network, and elaborates on their strengths and their shortcomings, arguing for the need for the IPS-system.

2.1 Antiviruses

Antivirus software is considered as an essential in any end-user device as a protective measure against cyber threats such as spyware or ransomware. Antiviruses come in many shapes and sizes, but they do share not only their purpose, but their means of accomplishing what is expected from them. They work by first detecting malicious files, and then they either quarantine said files; [5] In other words: block their hypothetical execution, [6] or just flat-out delete them.

Historically the forementioned detection has been based on fingerprinting, i.e., checking whether the antivirus-provider has seen that exact piece of malware before. Modern antiviruses have also heuristic capabilities; An example of these would be the ability to analyse program’s behaviour for resemblance to known viruses. [7]

2.2 Firewalls

Another essential tool that most user-devices and home networks have in place in addition to antiviruses are firewalls. They also do come in various forms, but their functionality is mostly concerned on layers from five to three in the OSI-model. [8]

Firewalls are great for blocking most of automated scanning and similar opportunistic threats on the internet. Provided below are the latest firewall logs of the author, comprising only the last minute of blocked traffic as of the time of writing.

```
Apr  4 22:07:11 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=35.204.36.X
DST=(SENSORED_LOCAL_IP) LEN=40 TOS=0x00 PREC=0x00 TTL=59 ID=41255 PROTO=TCP
SPT=56651 DPT=5432 SEQ=3427139256 ACK=0 WINDOW=1024 RES=0x00 SYN URGP=0
MARK=0x8000000
Apr  4 22:07:12 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=62.233.50.X
DST=(SENSORED_LOCAL_IP) LEN=40 TOS=0x00 PREC=0x00 TTL=248 ID=13002 PROTO=TCP
```

```
SPT=49067 DPT=6603 SEQ=1901814182 ACK=0 WINDOW=1024 RES=0x00 SYN URGP=0
MARK=0x8000000
Apr  4 22:07:16 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=183.136.225.X
DST=(SENSORED_LOCAL_IP) LEN=44 TOS=0x00 PREC=0x00 TTL=113 ID=0 PROTO=TCP
SPT=12731 DPT=41795 SEQ=2144109552 ACK=0 WINDOW=29200 RES=0x00 SYN URGP=0 OPT
(020405B4) MARK=0x8000000
Apr  4 22:07:21 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=192.241.221.X
DST=(SENSORED_LOCAL_IP) LEN=40 TOS=0x00 PREC=0x00 TTL=242 ID=54321 PROTO=TCP
SPT=45151 DPT=17185 SEQ=1493399949 ACK=0 WINDOW=65535 RES=0x00 SYN URGP=0
MARK=0x8000000
Apr  4 22:07:39 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=213.226.123.X
DST=(SENSORED_LOCAL_IP) LEN=40 TOS=0x00 PREC=0x00 TTL=248 ID=5933 PROTO=TCP
SPT=59181 DPT=925 SEQ=1931632151 ACK=0 WINDOW=1024 RES=0x00 SYN URGP=0
MARK=0x8000000
Apr  4 22:07:49 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=89.248.163.X
DST=(SENSORED_LOCAL_IP) LEN=40 TOS=0x00 PREC=0x00 TTL=249 ID=26841 PROTO=TCP
SPT=47468 DPT=5008 SEQ=422982126 ACK=0 WINDOW=1024 RES=0x00 SYN URGP=0
MARK=0x8000000
Apr  4 22:07:54 kernel: DROP IN=eth0 OUT= MAC=(SENSORED) SRC=75.80.10.X
DST=(SENSORED_LOCAL_IP) LEN=40 TOS=0x00 PREC=0x00 TTL=53 ID=51928 PROTO=TCP
SPT=32081 DPT=23 SEQ=1536969383 ACK=0 WINDOW=25930 RES=0x00 SYN URGP=0
MARK=0x8000000
```

All the packets in the example above were dropped by Asus-router as being unsolicited by any user-device. The unsolicited nature of these packets could have been derived from the unknown sender IP-addresses, port-numbers, and TCP-sequence numbers. Process of dropping packets based on the mentioned details is called packet filtering. [9]

But if devices inside the local network were to solicit some malicious connections, be it due to some user-error or say malware inside the local network - firewall would let the packets in. [9]

Unlike antiviruses, firewall at the router does protect the whole local network. But as argued earlier, firewalls' means for protection are limited.

2.3 IP and DNS as Indicators of Compromise (IOC)

To summarize previous sections, antiviruses use mainly fingerprinting and behavioural analysis to detect bad intentions. [7] Simple firewalls depend on packet filtering and manual blacklisting of ports and IP-addresses by the administrator. [8]

However, routers do have full visibility to IP-addresses and domain names, which are known to be valuable IOCs [10] and neither antivirus nor your typical home-router firewall takes this into consideration.

There are free tools that analyse both IP-addresses and domain names for malicious content. One of these tools is virustotal.com. It can analyse files, fingerprints, IP-addresses, and URLs and provides an API for programmers to create automation for security analysis. [11]

Services, such as VirusTotal allow for free up-to-date threat intel that could be used at a router to significantly fortify home-network security. However, free version of VirusTotal API allows for maximum of 4 requests per minute. Therefore, should be studied if any other Threat Intel-provider would prove more useful for this use case. [11, 12]

3 Maltrail

The goal of this thesis was to produce an efficient and free-of-charge automated system for home-networks that would use services such as Virustotal to utilize the routers' visibility to IP-details to strengthen home-networks antimalware-capabilities.

There does already exist an open-source project called Maltrail, [13] it has functionalities similar to ones planned for this thesis-project. However - Maltrail requires comparatively lot of resources due to being excessive in features and being implemented using python-programming language, which is known for being inefficient both execution- and memory-wise. [14]

However, since Maltrail is open-source it can be used both as an inspiration and as a reference point for goals regarding memory- and execution-efficiency on the development process of the IPS-system.

One key difference to mention is the fact that Maltrail only logs traffic and events. It alone does not block any traffic but requires the administrator to act on the intelligence it provides. [13] The IPS-system planned to create in this thesis should work as a proxy that will automatically block suspicious traffic.

3.1 Requirements and benchmarking

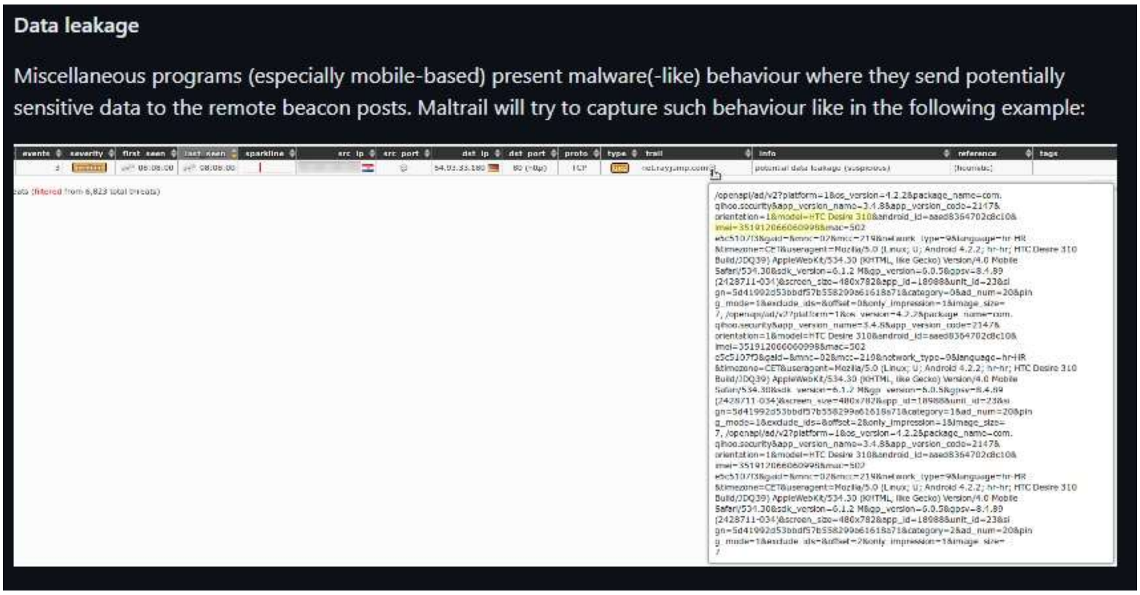
Maltrail's Github-documentation states that it requires a minimum of 1GB of RAM to run in a single-process mode. [13] A running instance of Maltrail should be benchmarked to investigate the resource-usage further.

3.2 Features

Maltrail has a range of features for both the detection and for the security event management on the provided web-interface.

3.2.1 Application layer-based features

Maltrail can detect and analyse application layer- protocols, such as HTTP and detect suspicious file downloads, vulnerability-scans, and data leakage. All the mentioned threats and how they are mitigated are described in the Maltrail's documentation. [13]

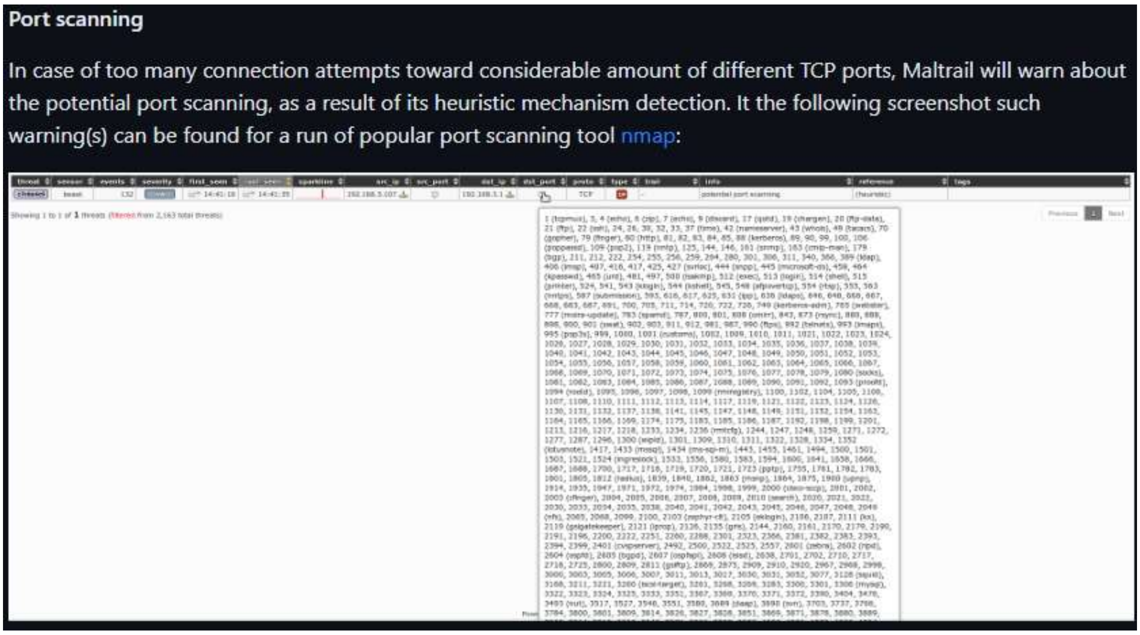


(Screenshot, https://github.com/stamparm/maltrail, Data leakage, accessed 2.5.2023)

Another application-layer protocol well utilized in security monitoring is DNS. A typical malware behavior is to use DNS while checking for victim IP-address with some IPinfo-service. Also, malware’s communications with some command and control-server may be identified based on suspicious DNS-requests. [13]

3.2.2 Transport and Network-layer based features

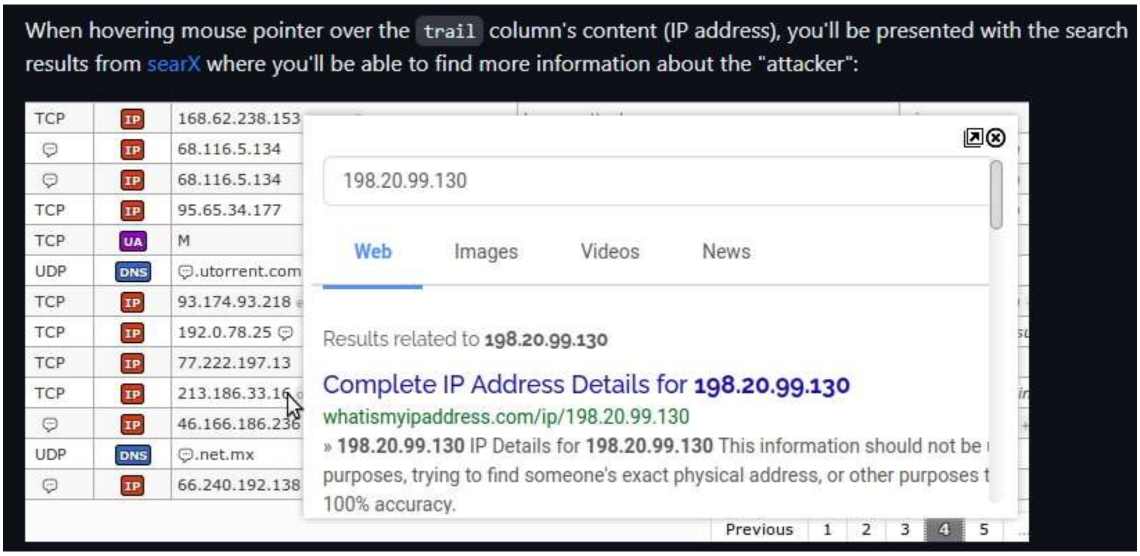
Similar to what was planned in this thesis – Maltrail checks IP-addresses for known bad actors. In addition, it can detect port scanning using a heuristic mechanism. Some light is shed upon the mechanism this is based on, but details are left unclear in the document. [13]



(Screenshot, https://github.com/stamparm/maltrail, Port scanning, accessed 2.5.2023)

3.2.3 The web-interface

As previously mentioned, Maltrail provides quite a sophisticated web-interface, which allows for regular expression-based searching on the security events and some Open-source intelligence automation on attacker IP-addresses just to name a few of the capabilities. [13]



(Screenshot, https://github.com/stamparm/maltrail, IP-address-lookup, accessed 2.5.2023)

3.3 Conclusions

Maltrail is a versatile and rather easy to use intrusion detection system, it has an excessive feature-list that strengthens network-administrator's visibility into the network with a multitude. However, it does not provide intrusion prevention capabilities and thus requires swift hands and active participation from the administrator to maintain network security in a case of a security incident.

The IP-monitoring capabilities are similar to what was planned for this thesis-work.

4 Golang

Go-language (later referred to as simply "Go") is an open-source programming language launched in 2009 by Rob Pike, Ken Thompson and Robert Griesemer at Google. Go is a procedural language for server software that promises a concurrency implemented via "goroutines" instead of traditional multiprocessor-threading. This is said to drastically decrease the memory consumption compared to other multithreaded high-level languages such as python while being flexible, dynamic, and easily maintainable. [15] The main philosophy behind Go was to be C-like in terms of efficiency in execution, yet to be scalable and easy to use and maintain such as python - this makes Go an attractive option for developing production software. [16]

4.1 Go Syntax

As mentioned earlier – Go is portrayed as dynamic, easily scalable, and maintainable language for software production. The following code-snippet is a generic example of how it looks. [17]

```
package main

import (                            // Multiple imports are defined as such
     "fmt"
     "time"
)


func printString(str string) {      // Define a function
     fmt.Println(str)               // Print to terminal using fmt-package
}

func main() {


     var x int                      // Declare single var with explicit type
     y := 0    // Declare single var - implicit type and initialization
     var z, q int = 1, 2  /* multiple variables - explicit type (int) */

     timeVar := time.Now()     // Function-call from the imported time-library

     for index := 0; index < 10; i++{          // for-loop
          fmt.Println("Index is %d\n", index)
     }


     index := 0
     for index < 10 {                     // while-loop
          fmt.Println("Index is %d\n", index)
     }

     printString("Hello World")           // Call a function
     go printString("Hello goroutine")    // Call a function as a goroutine
}
```

4.2 Pointers, references, and memory optimization

Go has both implicit and explicit typing, and pointers. A pointer is a variable holding a memory-address of a value rather than holding the value itself, this address is usually called “a reference”. This is most efficient when dealing with large structs, as only the memory-address (usually 64bits) rather than the whole object needs to be copied around. [18]

```
func main() {
    frstVar := 5           // the variable is declared and set to 5
    frstVar = multiply(5)  // the result (10) is returned and assigned to frstVar
    scndVar := 5           // the variable is declared and set to 5
    multiplyWithPointer(&scndVar) // a reference is given as an input and the
                                  // value is multiplied in the function.
    fmt.Println(frstVar)          // Will print "10"
    fmt.Println(scndVar)          // Will print "10"
}

func multiply(input int) {
    return input*2
}

func multiplyWithPointer(input *int) {
    *input = *input*2
}
```

In the example above the multiply()-function takes in the variable input. The value in function-call is copied from the main-functions stack-memory frame to another temporary instance in the memory frame of the multiply-function. When implemented as such, program will have the variable frstVar in memory twice when executing multiply().

However, in the function-call of the multiplyWithPointer(), the function takes in a reference to input-value. This value is then dereferenced and multiplied in the function using the original value in memory; Thus, the effect from the function (multiplication in this context) continues to exists outside of the function, and both frstVar and scndVar will hold value of 10 when printed in the last lines of main().

Again - when dealing with simple integers, this does not really matter, but say if the input-variable were a slice of 3000-string-values; Using pointers and references would become a necessity.

But it is not that simple efficiency-wise - when one passes a variable to a function in Go, Go copies this variable to stack-memory; But whenever one passes a reference to a function in Go, Go does an escape analysis to see if the data should be stored in heap-memory or in stack-memory. This adds buffer to the execution. In addition to that, having larger heap adds buffer to garbage collection-processes of Go. [18]

However, all this extra overhead may be leveled out by the fact that potential large structs are processed as pointers, hence the only way to know for sure whether to use pointers or not is to benchmark the software. [18]

4.3 Goroutines

In the end of the code-snippet in the beginning of this chapter, the printString(str)-function was called both normally and as a goroutine. Using a goroutine multiplexes all independent executions, or “coroutines” onto a set of threads. Whenever such function executes a blocking system call, the run-time moves all other coroutines to other non-blocked threads allowing them to keep running regardless of the blocking call. [17]

Each goroutine is allocated a miniscule amount of resizable and bounded stack-memory. The buffer for CPU is said to average on around three machine-code instruction for each function call and this efficiency allows for as much as six-figure sums of goroutines to coexist in the same address space. With traditional multithreading this would be unthinkable because of the inevitable resource shortage. [17]

4.4 Go-modules

Dependency management in Golang is done using a go.mod-file. It is used to maintain all modules used in the project in question. Said file can be initialized using “go mod init”-command. Whenever new packages are imported, “go get X” can be executed to download the imported library. Another tool to know is “go mod tidy" which synchronizes the go.mod-file and the packages used in the project. [19]

5 Project plan

This project aimed to create an IPS system that would consist of a transparent proxy that all IP-traffic is routed through. If the remote address is whitelisted the packet should pass. On the other hand, if the remote address is blacklisted the packet should be dropped.

The program should check all non-white- nor blacklisted remote IP-addresses with threat intel API, to see if the system is connecting to known suspicious addresses. If threat intel API marks the address as suspicious, that IP-address should be blacklisted. If address comes out clean, this address should be whitelisted.

The original project plan is broadly broken down into a bullet-point list of tasks in this section. There might have come some minor tweaking along the line.

- Setup a development system

  A virtual Linux-system for the development should be created.

- Study appropriate libraries for the implementation

  A study should be conducted to find out what libraries could be utilized to create the IPS-system

- Implement: Capture traffic on network-interface (IPv4)

  Program should be able to capture traffic on network-interface to a network-listener.

- Implement: Interactive mode

Program should be able to be executed in a manner where IPS-functionality can be started and stopped. In a real scenario this could be left running on a system in a detached terminal.

- Implement: Logging

Program should be able to write events to a log-file.

- Implement: Manipulate firewall-rules to proxy traffic to IPS

Program should be able to set the system to proxy all traffic to a network-listener

- Implement: Ability to pass on traffic

Program should be able to send allowed traffic to the recipient.

- Implement: Ability to drop traffic (+ logging details)

Program should be able to drop blocked traffic and log details on the event.

- Implement: White- and blacklist functionalities

Program should have a list of allowed and a list of blocked addresses

- Implement: Check with threat intel API

Program should be able to ask the threat intel API to provide threat intel on the IP-addresses of interest.

- Implement: IPv6 support

Program should be able to use IPv6-addresses as well.

- Implement: Set verbosity levels

User should be able to set verbosity level of logging.

- Implement: Add revocation-loop to whitelist

Whitelist should be broken down to two different lists, a permanent user-created whitelist of fixed good addresses, and automatically created whitelist of checked addresses. The items on the latter one should be periodically revocated, to ensure that the system has up-to-date intel on foreign addresses.

6 Project diary

The project began by setting up a virtual development environment. This process is documented in the following section.

6.1 Setting up the development system

The first step was to setup a virtual instance of the latest Ubuntu-Linux, which was 22.04 at the time of writing.

```
samulikalliomaa@Thesis-buntu:~$ uname -a
Linux Thesis-buntu 5.15.0-69-generic #76-Ubuntu SMP Fri Mar 17 17:19:29 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
samulikalliomaa@Thesis-buntu:~$
```

(Screenshot, bash-terminal, system details, 13.04.2023)

6.1.1 Visual Studio Code

Visual Studio Code was used as an IDE for this thesis project, which was available as an .dep-package to be installed in Debian-based Linux-systems, such as Ubuntu. [20]

Said package was installed using dpkg-command with -I (install) option and sudo-privileges in Ubuntu.

```
samulikalliomaa@Thesis-buntu:~/Downloads$ sudo dpkg -i code_1.77.3-1681292746_amd64.deb
[sudo] password for samulikalliomaa:
Selecting previously unselected package code.
(Reading database ... 221440 files and directories currently installed.)
Preparing to unpack code_1.77.3-1681292746_amd64.deb ...
Unpacking code (1.77.3-1681292746) ...
Setting up code (1.77.3-1681292746) ...
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
Processing triggers for gnome-menus (3.36.0-1ubuntu3) ...
Processing triggers for desktop-file-utils (0.26-1ubuntu3) ...
Processing triggers for shared-mime-info (2.1-2) ...
```

(Screenshot, bash-terminal, Install VScode, 13.04.2023)

Installation was verified by running code-command using --v (version) option.

```
samulikalliomaa@Thesis-buntu:~/Downloads$ code --v
1.77.3
704ed70d4fd1c6bd6342c436f1ede30d1cff4710
x64
```

(Screenshot, bash-terminal, Verify VScode installation, 13.04.2023)

The output above proves that the installation was successful and that the installed version was 1.77.3. In addition, output consisted of the 40 character long hexadecimal value stating the software build-id and of the system architecture, which was 64-bit Intel x86.

6.1.2 Git

Git and Github were used for version control on this project. First requirement was to install the git-package using apt-package manager.

```
samulikalliomaa@Thesis-buntu:~$ whatis git
git: nothing appropriate.
samulikalliomaa@Thesis-buntu:~$ sudo apt install git
[sudo] password for samulikalliomaa:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 4 121 kB of archives.
After this operation, 20,9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://fi.archive.ubuntu.com/ubuntu jammy/main amd64 liberror-perl all 0.17029-1 [26,5 kB]
Get:2 http://fi.archive.ubuntu.com/ubuntu jammy-updates/main amd64 git-man all 1:2.34.1-1ubuntu1.8 [953 kB]
Get:3 http://fi.archive.ubuntu.com/ubuntu jammy-updates/main amd64 git amd64 1:2.34.1-1ubuntu1.8 [3 141 kB]
Fetched 4 121 kB in 3s (1 283 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 202506 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-1_all.deb ...
Unpacking liberror-perl (0.17029-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.34.1-1ubuntu1.8_all.deb ...
Unpacking git-man (1:2.34.1-1ubuntu1.8) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.34.1-1ubuntu1.8_amd64.deb ...
Unpacking git (1:2.34.1-1ubuntu1.8) ...
Setting up liberror-perl (0.17029-1) ...
Setting up git-man (1:2.34.1-1ubuntu1.8) ...
Setting up git (1:2.34.1-1ubuntu1.8) ...
Processing triggers for man-db (2.10.2-1) ...
samulikalliomaa@Thesis-buntu:~$
```

(Screenshot, bash-terminal, Installing git, 18.04.2023)

An SSH-keypair was created as means for authentication between development environment and Github. ED25519 signature-system was used for key-generation, as it was the de facto secure and efficient signature system at the time of writing. [21]

6.1.3 Golang

Comprehensive instructions for installing the latest version of Go can be found from Go's website https://go.dev/doc/install. A binary release for Linux was used in this project.

The steps were:

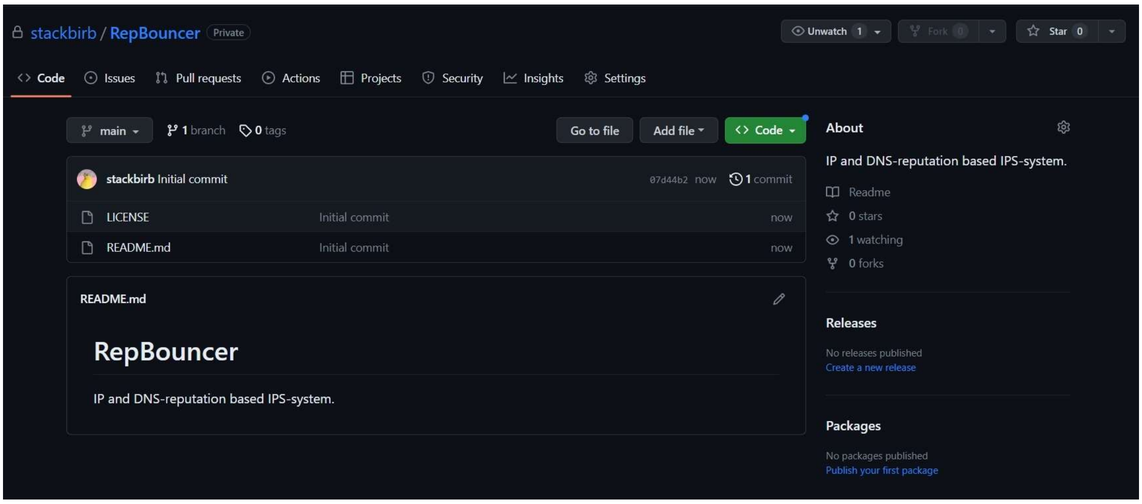
1. Download the correct archive, which was go1.20.3.linux-amd64.tar.gz
2. execute "rm -rf /usr/local/go && tar -C /usr/local -xzf go1.20.3.linux-amd64.tar.gz", to remove any hypothetical previous installation of Go and extract the downloaded archive to /usr/local. Needs sudo-privileges.
3. Add Go's binary-directory to the $PATH environment variable by adding "export PATH=$PATH:/usr/local/go/bin" to ~/.profile
   Small optional logic was added to ensure the path exists before adding it to $PATH:
   ```
   # USER PATHS:
   if [ -d "/usr/local/go/bin" ] ; then
       export PATH=$PATH:/usr/local/go/bin
   fi
   ```
   (screenshot, vim, if-statement for $PATH, 18.04.2023)

4. Execute the export-command above for immediate application.
5. Verify installation by executing "go version"

6.1.4 Creating the project-repository

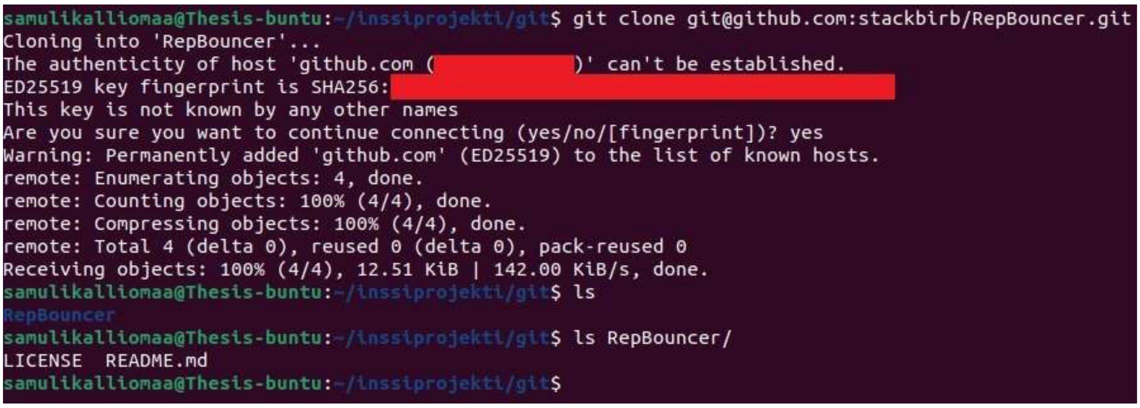
As mentioned earlier, Github was used to store the project-code. A working title of "RepBouncer" was chosen. Project was published under GNU General Public License v3.0 to allow for free usage and modification of the project for anyone,

but to leave a minimum amount of room for privatizing any component of the project compared to say MIT license, which is another well-known open license model, but less strict on usage-policies. [22]
Should be added that project was kept private until a minimum value product-state.



(Screenshot, https://github.com/stackbirb/RepBouncer, accessed 18.04.2023)

As the repository was now created, it was time to verify the connection and authentication between the git-CLI and github.com, by cloning the RepBouncer-repository to the local development-system.



(Screenshot, bash-terminal, git clone, 18.04.2023)

It worked, the public-key of github.com was added to local known_hosts-file, and a local clone of RepBouncer-repository was created into ~/inssiprojekti/git/-directory.

6.2 The development

Initially a new development branch was created to begin developing the program.

```
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ git checkout -b RB_1_First_Testing
Switched to a new branch 'RB_1_First_Testing'
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ git log
commit 07d44b292a5ddc3471e4a0e17e2069d70b0a2efb (HEAD -> RB_1_First_Testing, origin/main, origin/HEAD, main)
Author: StackBirb <43564716+stackbirb@users.noreply.github.com>
Date:   Tue Apr 18 15:43:09 2023 +0300

    Initial commit
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ git status
On branch RB_1_First_Testing
nothing to commit, working tree clean
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$
```

(Screenshot, bash-terminal, git-status, 18.04.2023)

When building software using go-lang, first thing is to initialize a module-file using the “go mod init”-command as described earlier in the Golang-chapter.

6.2.1 Feature: Open capture on network-interface

First planned feature was ability to read from network-interface. This began by studying the libraries needed, they are listed and briefly described in the following bullet-point list:

- “fmt” - a go standard library for formatted I/O functionalities.
- “log” - a go standard library for simple logging functionalities, also provides error-handling.
- “net” – a standard library for network I/O.
- “syscall” – a standard library for using system-calls. [23]

Google provides a library called “gopacket” that provides packet capture and inspection capabilities. [24] This library was cloned from Github for further studying.

The library is broadly documented in the fascinating doc.go-files provided in the repository for each package and sub-package individually.

One sub-package of interest to mention was pcap, which was advertised as “C bindings to use libpcap to read packets off the wire”. This allows for reading live packets from a network interface using pcap-class’s method called “OpenLive”.

```
Reading Live Packets

The following code can be used to read in data from a live device, in this case
"eth0". Be aware, that OpenLive only supports microsecond resolution.

 if handle, err := pcap.OpenLive("eth0", 1600, true, pcap.BlockForever); err != nil {
   panic(err)
 } else if err := handle.SetBPFFilter("tcp and port 80"); err != nil {  // optional
   panic(err)
 } else {
   packetSource := gopacket.NewPacketSource(handle, handle.LinkType())
   for packet := range packetSource.Packets() {
     handlePacket(packet)  // Do something with a packet here.
   }
 }
```

(Screenshot, VScode, gopacket/pcap/doc.go, 18.04.2023)

Next the pcap-package was imported using go get-command.

```
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ go get github.com/google/gopacket/pcap
go: added github.com/google/gopacket v1.1.19
go: added golang.org/x/sys v0.0.0-20190412213103-97732733099d
```

(Screenshot, bash-terminal, go get-command, 20.04.2024)

And the very first executable was written, this version just wrote a welcoming message to std.out and opened and closed a network-interface for capturing. Retrospectively, this opening should have been verified by adding an else-statement on row 21 with some print in, but luckily one was not necessary.

```
RepBouncer.go M ×    go.mod M
cmd > RepBouncer.go > main
You, 9 minutes ago | 1 author (You)
package main

import (
    "fmt"
    "time"

    "github.com/google/gopacket/pcap"
)

func main() {
    fmt.Println("Welcome to RepBouncer!")
    device := "enp0s3"
    snaplen := int32(65535)
    promisc := false
    timeout := time.Duration(0)

    handle, err := pcap.OpenLive(device, snaplen, promisc, timeout)
    if err != nil {
        fmt.Printf("Error opening device: %v\n", err)
        return        You, 3 seconds ago • Uncommitted changes
    }
    defer handle.Close()
}
```

(Screenshot, VScode, First WIP, 20.04.2023)

Trying to execute the first WIP-software revealed that there is some error in importing the pcap-library.

```
# github.com/google/gopacket/pcap
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:30:22: undefined: pcapErrorNotActivated
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:52:17: undefined: pcapTPtr
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:64:10: undefined: pcapPkthdr
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:102:7: undefined: pcapBpfProgram
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:103:7: undefined: pcapPkthdr
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:261:33: undefined: pcapErrorActivated
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:262:33: undefined: pcapWarningPromisc
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:263:33: undefined: pcapErrorNoSuchDevice
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:264:33: undefined: pcapErrorDenied
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:738:14: undefined: pcapTPtr
../../../go/pkg/mod/github.com/google/gopacket@v1.1.16/pcap/pcap.go:264:33: too many errors
```

(Screenshot, bash-terminal, go run- output, 20.04.2023)

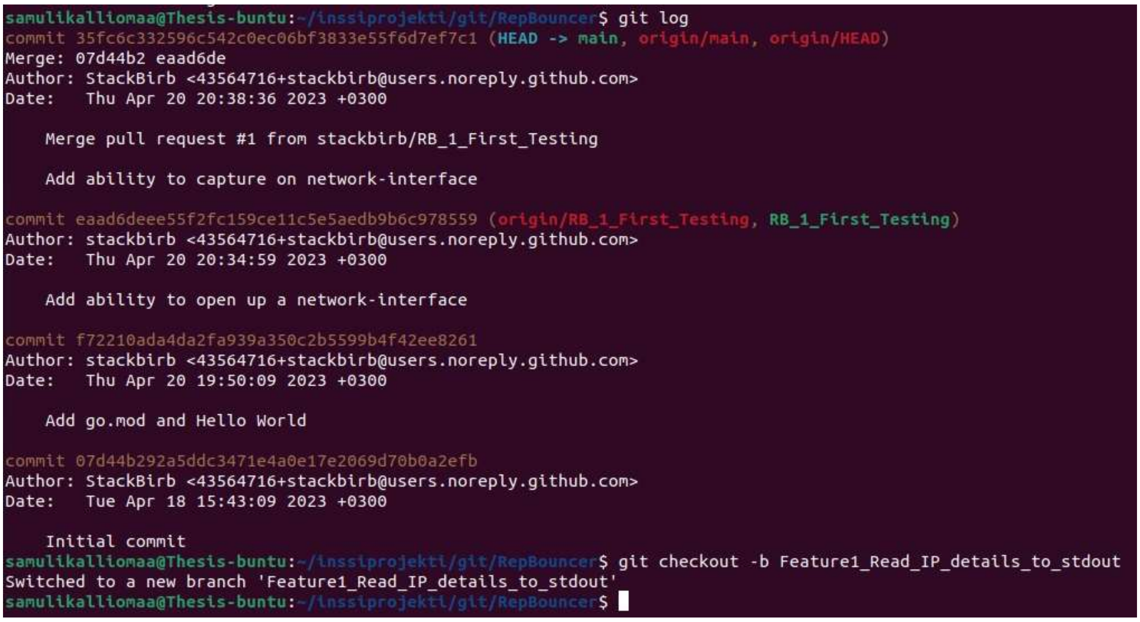
After some research turned out other developers have had similar issues, and these had been resolved by setting the $CGO_ENABLED-environmental variable to 1. [25] In addition to setting said variable immediately, said set was added to ~/.bashrc to stop the need for running that with each login. Turned out the system was missing GCC as well, therefore that had to be installed.

And of course, capturing on network interface requires sudo-privileges, with that in mind the execution produced no errors!

```
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ sudo /usr/local/go/bin/go run cmd/RepBouncer.go
go: downloading github.com/google/gopacket v1.1.19
Welcome to RepBouncer!
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$
```

(Screenshot, bash-terminal, successful execution, 20.04.2023)

The development-branch was pushed to GitHub, merged to main and new development-branch was checked out with plan to implement the ability to read IP-addresses to std.out from traffic.



```
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ git log
commit 35fc6c332596c542c0ec06bf3833e55f6d7ef7c1 (HEAD -> main, origin/main, origin/HEAD)
Merge: 07d44b2 eaad6de
Author: StackBirb <43564716+stackbirb@users.noreply.github.com>
Date:   Thu Apr 20 20:38:36 2023 +0300

    Merge pull request #1 from stackbirb/RB_1_First_Testing

    Add ability to capture on network-interface

commit eaad6deee55f2fc159ce11c5e5aedb9b6c978559 (origin/RB_1_First_Testing, RB_1_First_Testing)
Author: stackbirb <43564716+stackbirb@users.noreply.github.com>
Date:   Thu Apr 20 20:34:59 2023 +0300

    Add ability to open up a network-interface

commit f72210ada4da2fa939a350c2b5599b4f42ee8261
Author: stackbirb <43564716+stackbirb@users.noreply.github.com>
Date:   Thu Apr 20 19:50:09 2023 +0300

    Add go.mod and Hello World

commit 07d44b292a5ddc3471e4a0e17e2069d70b0a2efb
Author: StackBirb <43564716+stackbirb@users.noreply.github.com>
Date:   Tue Apr 18 15:43:09 2023 +0300

    Initial commit
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$ git checkout -b Feature1_Read_IP_details_to_stdout
Switched to a new branch 'Feature1_Read_IP_details_to_stdout'
samulikalliomaa@Thesis-buntu:~/inssiprojekti/git/RepBouncer$
```

(Screenshot, bash-terminal, Git log after first merge, 20.04.2023)

Similar git-operations were executed after each feature implementation onwards and will not be mentioned further.

6.2.2 Feature: Extract IP-addresses from traffic

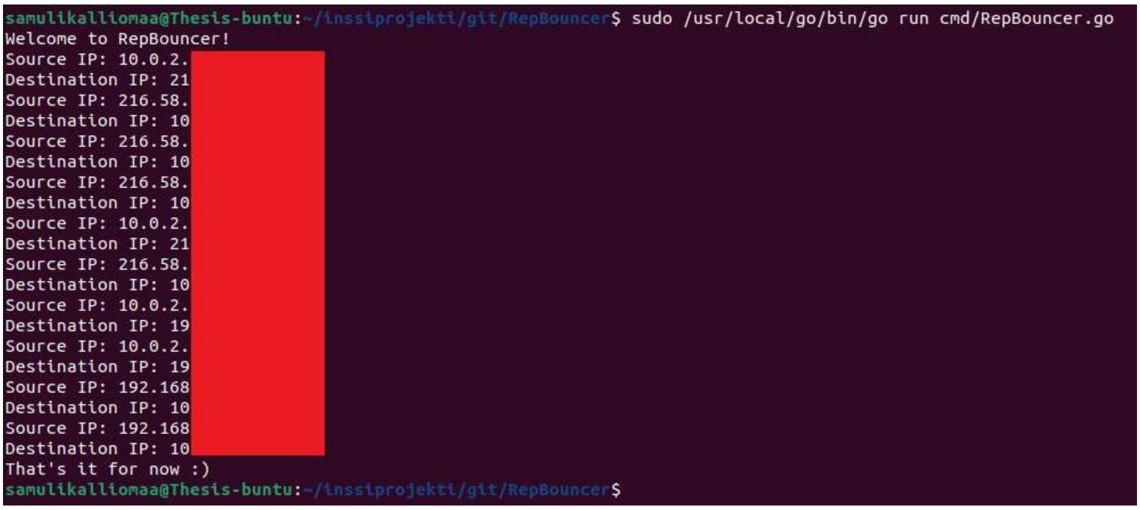
Next step was to implement the extraction of IP-addresses from traffic. This was done using the layers-subpackage.

```go
func main() {
    fmt.Println("Welcome to RepBouncer!")
    device := "enp0s3" // TODO: create ability to set this dynamically
    snaplen := int32(65535)
    promisc := false // Probably needs to be on in router
    timeout := pcap.BlockForever
    amountOfPackets := 0

    handle, err := pcap.OpenLive(device, snaplen, promisc, timeout)
    if err != nil {
        log.Fatalf("Error opening device: %s: %s\n", device, err)
        return
    } else {
        packetSource := gopacket.NewPacketSource(handle, handle.LinkType())
        for packet := range packetSource.Packets() {
            readIpDetails(packet)
            amountOfPackets += 1
            if amountOfPackets >= 10 {
                fmt.Println("That's it for now :)")
                handle.Close()      You, now • Uncommitted changes
                os.Exit(0)
            }
        }
    }
    defer handle.Close()
}

// Read Ip-addresses from packets
func readIpDetails(packet gopacket.Packet) {
    ipLayer := packet.Layer(layers.LayerTypeIPv4)
    if ipLayer == nil {
        return // Skip non-IPv4 packets for now TODO: implement IPv6 support
    }
    ipPacket := ipLayer.(*layers.IPv4)
    fmt.Println("Source IP:", ipPacket.SrcIP.String())
    fmt.Println("Destination IP:", ipPacket.DstIP.String())
}
```

(Screenshot, VScode, reading IP-addresses, 20.04.2023)

Decoding packets with gopacket/layers was straightforward. The ipLayer-variable in the code above is a gopacket.layer-object and ipPacket object is derived from that. Then source and destination IP-addresses can be accessed as net.IP-objects and said object-class has a .String()-method which casts the net.IP-object to human-readable strings that can be printed to std.out using the fmt-library.

An iterator was added to packet-reading loop to exit the program in a clean and timely manner.



(Screenshot, bash-terminal, reading IP-addresses, 20.04.2023)

6.2.3 Feature: Support Interactive mode

Ability to execute program in an interactive mode where user can interactively launch the IPS-functionality and set settings etc. was built next. Idiomatic way to parse CLAs in golang is using the flag-builtin package, this was implemented as well.

```go
// MAIN  //
func main() {
    // "GLOBAL" VARIABLES //
    welcomeTitle := "\n\nRepBouncer v0.02 - by Samuli Kalliomaa github.com/stackbirb\n\n"
    device := "enp0s3" // TODO: create ability to set this dynamically
    snaplen := int64(65535)
    promisc := true // Probably needs to be on in router
    timeout := pcap.BlockForever
    choice := 0
    captureOn := false
    interactiveMode := false
    verboseLevel := 0 //TODO: implement
    exitCapturing := true
    loggingVar := []string{}
    loggingVar = append(loggingVar, "Logs:")
    loggingVarMaxLength := 20
      You, 3 days ago • Add InteractiveMode and CLA-parsing …
    //main-loop
    read_command_line_arguments_and_set_settings(&device, &snaplen, &promisc, &interactiveMode, &verboseLevel)
    if interactiveMode {
        main_menu(&device, &snaplen, &promisc, &timeout, &interactiveMode, &verboseLevel, &choice, &captureOn, &exitCapturing, &welcomeTitle, &
    } else {
        fmt.Println(welcomeTitle)
        openNetworkInterfaceAndCapturePackages(&device, &snaplen, &promisc, &timeout, &exitCapturing, &loggingVar, &loggingVarMaxLength)
    }
}

///   FUNCTIONS   ///

// CLA-parsing
func read_command_line_arguments_and_set_settings(device *string, snaplen *int64, promisc *bool, interactiveMode *bool, verboseLevel *int) {
    flag.StringVar(device, "device", "enp0s3", "The network-interface to use (default: enp0s3)")
    flag.Int64Var(snaplen, "snaplength", int64(65535), "the Snaplength to use (default: 65535)")
    flag.BoolVar(promisc, "promisc", true, "Promiscuous mode (default: true)")
    flag.BoolVar(interactiveMode, "interactive", false, "Interactive-mode (default: false)")
    // flag.IntVar(verboseLevel, "verbose", 0, "verbose-level: 0:silent 1:verbose, 2:very verbose(for debugging), (default: 0)")
    flag.Parse()
}
```

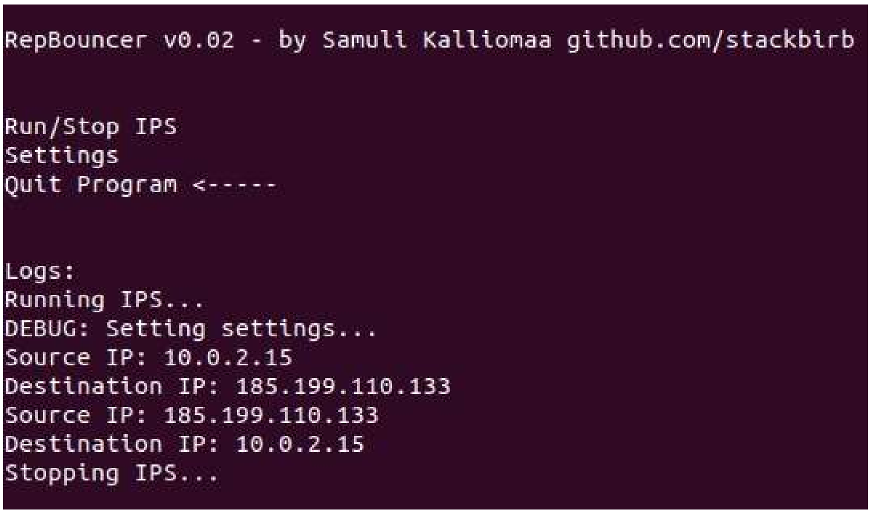
(Screenshot, VScode, RepBouncer main and CLA-parsing, 25.04.2023)

Functionalities so far were broken down into functions and library called termbox was added to allow for reading keystrokes from user while in interactive mode.

```go
        //Print-menu
        printMenu(&menuChoicesList, &chosenItem, welcomeTitle)
        printLoggingVar(loggingVar)

        //Menu-Control
        switch menuEvent := termbox.PollEvent(); menuEvent.Type {
        case termbox.EventKey:
            if menuEvent.Key == termbox.KeyArrowUp {
                if chosenItem > 0 {
                    chosenItem -= 1
                }
            } else if menuEvent.Key == termbox.KeyArrowDown {
                if chosenItem < len(menuChoicesList)-1 {
                    chosenItem += 1
                }
            } else if menuEvent.Key == termbox.KeyEnter || menuEvent.Key == termbox.KeySpace {
                termbox.Close()
                execute_menu_item(&chosenItem, &menuChoicesList, device, snaplen, promisc, timeo
            }
        case termbox.EventError:
            log.Fatal(menuEvent.Err)
        }
    }

}
```

(Screenshot, VScode, RepBouncer menu-control, 25.04.2023)

Interactive mode was executed using –interactive flag.



(Screenshot, bash-terminal, execution in interactive mode, 25.04.2023)

6.2.4 Feature: Transparent proxy

At the time being the RepBouncer only read packets off the wire – in other words took copies of traffic passing by. For the intended functionality the traffic should have been proxied to sockets controlled by RepBouncer and then it would have automatically decided what to do with the packets. All this should have been transparent to both the client and any server communicating with the client. Such network function is called transparent proxy. [26] Turned out that go has package for exactly that called tproxy, [27] and that was implemented to do the listening instead of the one previously implemented.

(Screenshot, https://pkg.go.dev/github.com/LiamHaworth/go-tproxy#section-readme, Golang TProxy documentation, accessed 29.04.2023)

The example use case offered in tproxy's github-page influenced the implementation heavily. Own listeners were implemented for both UDP and TCP connections.

```go
func listenForTraffic() {

    updateLoggingVar("DEBUG: in ListenForTCPTraffic...")
    var err error

    tcpListener, err = tproxy.ListenTCP("tcp", &net.TCPAddr{IP: net.ParseIP("127.0.0.1"), Port: 12347})
    if err != nil {
        log.Fatalf("ERROR: Could not bind TCP listener %s", err)
    }
    go listenTCP()

    udpListener, err = tproxy.ListenUDP("udp", &net.UDPAddr{IP: net.ParseIP("127.0.0.1"), Port: 12348})
    if err != nil {
        log.Fatalf("ERROR: Could not bind UDP listener %s", err)
        return
    }

    go listenUDP()
```

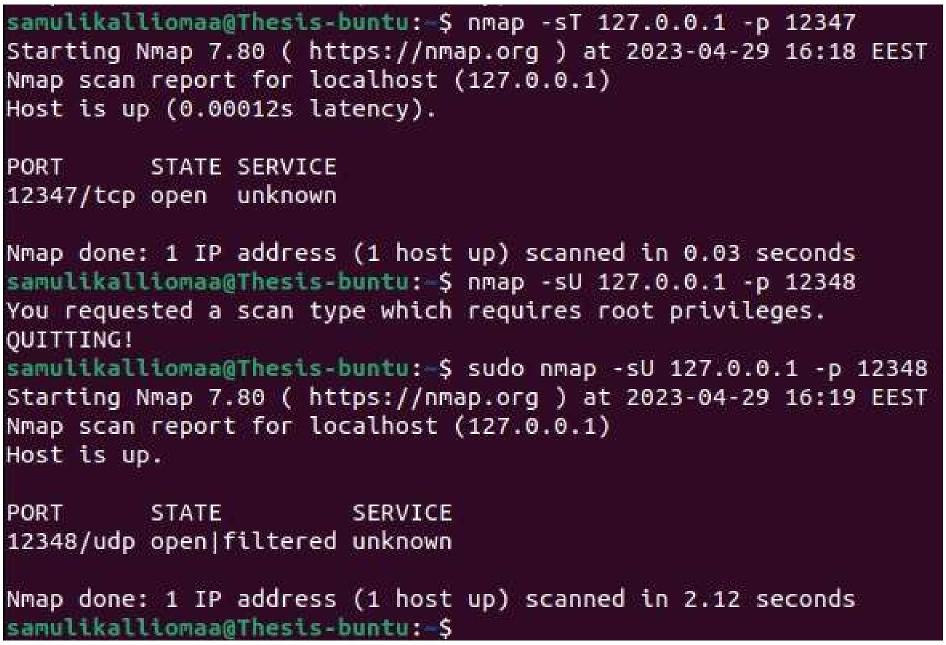
(Screenshot, VScode, listenForTraffic(), 29.04.2023)

To be able to stop listeners once they are activated, a go-native channel-implementation was created to achieve that for both TCP and UDP connections, the screenshot from below is an example from listenTCP()-function.

```go
// listenTCP runs in a routine to
// accept TCP connections and hand them
// off into their own routines for handling
func listenTCP() {
    for {
        select {
        default:
            conn, err := tcpListener.Accept()
            if err != nil {
                if netErr, ok := err.(net.Error); ok && netErr.Temporary() {
                    updateLoggingVar("Temporary error while accepting connection: " + netErr.Error())
                }
                log.Fatalf("Unrecoverable error while accepting connection: %s", err)
                return
            } else {
                go handleTCPConn(conn)
            }
        case <-stopChannelTCP:
            updateLoggingVar("DEBUG: TCP listener closing...")
            tcpListener.Close()
            updateLoggingVar("DEBUG: TCP listener closed...")
            return
        }
    }
}
```

(Screenshot, VScode, Select-statement with <-stopChannelTCP case, 29.04.2023)

The proxies were tested and indeed they got connections. This was verified using nmap.



```
samulikalliomaa@Thesis-buntu:~$ nmap -sT 127.0.0.1 -p 12347
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-29 16:18 EEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00012s latency).

PORT      STATE SERVICE
12347/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.03 seconds
samulikalliomaa@Thesis-buntu:~$ nmap -sU 127.0.0.1 -p 12348
You requested a scan type which requires root privileges.
QUITTING!
samulikalliomaa@Thesis-buntu:~$ sudo nmap -sU 127.0.0.1 -p 12348
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-29 16:19 EEST
Nmap scan report for localhost (127.0.0.1)
Host is up.

PORT      STATE         SERVICE
12348/udp open|filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 2.12 seconds
samulikalliomaa@Thesis-buntu:~$
```

(Screenshot, bash-terminal, nmap scans to TCP and UDP proxies, 29.04.2023)

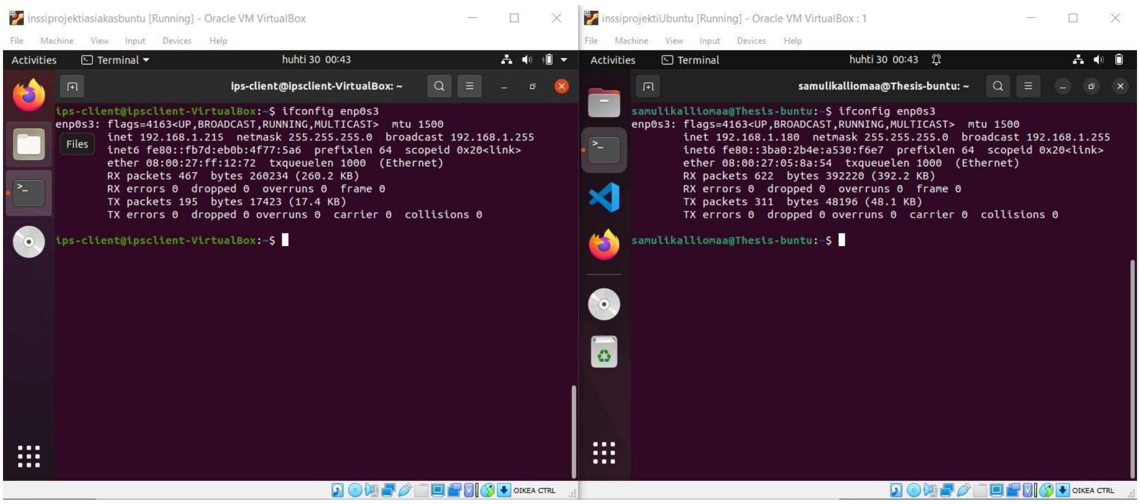
These scans were printed to screen on the running instance of RepBouncer. Connections did fail as can be seen from the screenshot below, but that could have been due to the way of testing. This would be further studied in the following chapters.

```
RepBouncer v0.03 - by Samuli Kalliomaa github.com/stackbirb


Run/Stop IPS <-----
Settings
Quit Program


DEBUG: CLAs parsed
Pre-existing iptable-rule-snapshot found, using that as a backup :)
DEBUG: IPTABLE SAVED
DEBUG: IN INTERACTIVE MODE
DEBUG: Checking exitCapturing
DEBUG: We are not exiting
Running IPS...
Forwarding successfully allowed for ipv4 traffic...
TCP proxy is set
UDP proxy is set
DEBUG: in ListenForTCPTraffic...
Accepting TCP connection from 127.0.0.1:37090 with destination of 127.0.0.1:12347
Accepting TCP connection from 127.0.0.1:37090 with destination of 127.0.0.1:12347
Failed to connect to original destination [127.0.0.1:12347]: dial: socket connect: cannot assign requested address
Accepting UDP connection from 127.0.0.1:33305 with destination of 127.0.0.1:12348
Accepting UDP connection from 127.0.0.1:33305 with destination of 127.0.0.1:12348
Failed to connect to original UDP source [127.0.0.1:33305]: dial: socket bind: address already in use
Accepting UDP connection from 127.0.0.1:33306 with destination of 127.0.0.1:12348
Accepting UDP connection from 127.0.0.1:33306 with destination of 127.0.0.1:12348
Failed to connect to original UDP source [127.0.0.1:33306]: dial: socket bind: address already in use
DEBUG: Checking exitCapturing
DEBUG: We are exiting
Stopping IPS...
```

(Screenshot, bash-terminal, listening works, 29.04.2023)

6.2.5 Feature: Manipulate Firewall to route everything to the transparent proxy

To better simulate the IPS-functionality, a new virtual system was created to act as the user device, all traffic from user were to be router via the IPS-system by first setting them both to the same virtual network and then setting the default gateway on the client system to point to the IPS-system.



(Screenshot, both virtual systems side by side, 30.04.2023)

```
ips-client@ipsclient-VirtualBox:~$ sudo ip route add default via 192.168.1.180
dev enp0s3
[sudo] password for ips-client:
ips-client@ipsclient-VirtualBox:~$ ip route show
default via 192.168.1.180 dev enp0s3
default via 192.168.1.1 dev enp0s3 proto dhcp metric 100
169.254.0.0/16 dev enp0s3 scope link metric 1000
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.215 metric 100
ips-client@ipsclient-VirtualBox:~$
```

(Screenshot, bash-terminal, set default gateway, 30.04.2023)

Plan was to try setting the routing manually via terminal on the IPS-system, and once a working set of configurations would be found, these settings would be forced in RepBouncer whenever the IPS would be executed, and vice versa previous settings would be forced back when IPS would be turned off.

A working set of commands was found, first the IP-forwarding was allowed on the system, this was achieved with command:
sysctl net.ipv4.ip_forward=1

Executing said command without the “=1” checks for current value, a logic was introduced that does the checking and sets the value if it is off.

```
func setProxyAllTrafficToIPTables() {
    cmdForIpForward := exec.Command("sysctl", "net.ipv4.ip_forward")
    output, err := cmdForIpForward.Output()
    if err != nil {
        log.Fatal("ERROR: Couldn't read ip-forwarding rules :( error: ", err)
    }
    if string(output) != "net.ipv4.ip_forward = 1" {
        cmdForSetIpForward := exec.Command("sysctl", "-w", "net.ipv4.ip_forward=1")
        if err := cmdForSetIpForward.Start(); err != nil {
            log.Fatal("ERROR: Could not set ipv4 forward, error: ", err)
        }
        if err := cmdForSetIpForward.Wait(); err != nil {
            log.Fatal("ERROR: while waiting for set ipv4 forward, error: ", err)
        }
        updateLoggingVar("Forwarding successfully allowed for ipv4 traffic...")
    } else {
        //verbose1
        updateLoggingVar("ipv4 forwarding already allowed, leaving that as it is :)")
    }
```

(Screenshot, VSCode, allowing ip_forward, 1.5.2023)

To route traffic to transparent proxies, iptables were configured.

```
	//sudo iptables -t nat -A PREROUTING -p tcp -j REDIRECT --to-port 12347
	cmdForTCPProxy := exec.Command("iptables", "-t", "nat", "-A", "PREROUTING", "-p", "tcp", "-j", "REDIRECT", "--to-port", "12347")
	if err := cmdForTCPProxy.Start(); err != nil {
		log.Fatal("ERROR: while setting TCP-proxy! error: ", err)
	}
	if err := cmdForTCPProxy.Wait(); err != nil {
		log.Fatal("ERROR: while waiting for setting TCP-proxy! error: ", err)
	}
	updateLoggingVar("TCP proxy is set")

	////sudo iptables -t nat -A PREROUTING -p udp -j REDIRECT --to-port 12348
	cmdForUDPProxy := exec.Command("iptables", "-t", "nat", "-A", "PREROUTING", "-p", "udp", "-j", "REDIRECT", "--to-port", "12348")
	if err := cmdForUDPProxy.Start(); err != nil {
		log.Fatal("ERROR: while setting UDP-proxy! error: ", err)
	}
	if err := cmdForUDPProxy.Wait(); err != nil {
		log.Fatal("ERROR: while waiting for setting UDP-proxy! error: ", err)
	}
	updateLoggingVar("UDP proxy is set")
}
```

(Screenshot, VSCode, setting iptables-rules, 1.5.2023)

These settings allowed for routing traffic from the client via the IPS-system. However, seemed as if the UDP-forwarding was broken. TCP-did not produce any error logs, but UDP did the very same ones that were visible in the nmap-testing.
This needed further studying, but first thing was to create proper logging.

6.2.6 Feature: Write logs to file

Before further studying the UDP-issues and other oncoming issues, a persistent log with timestamps was required. This was implemented as follows - logs were written to string-slice variable until it reaches a size-limit of 150 items. Then all those would be appended to the log-file and the variable would be re-initialized as empty. The reasoning behind this implementation was to minimize the times writing is done to persistent memory.

```go
func updateLoggingVar(stringToUpdate string) {
    if len(loggingVar) >= loggingVarMaxLength {
        writeLogsToFile()
        loggingVar = []string{}
    }
    t := time.Now()
    loggingVar = append(loggingVar, t.Format("2006-01-02 15:04:05.000")+"\t"+stringToUpdate)
}

func writeLogsToFile() {
    file, err := os.OpenFile("repbouncer.log", os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil {
        log.Fatal("ERROR: Could not open log-file for writing", err)
    }
    defer file.Close()

    writer := bufio.NewWriter(file)
    for _, line := range loggingVar {
        _, err := writer.WriteString(line + "\n")
        if err != nil {
            log.Fatal("ERROR: while writing to file, ", err)
        }
    }

    // flush the writer to ensure all data is written to file
    err = writer.Flush()
    if err != nil {
        log.Fatal(err)
    }
}
```

(Screenshot, VSCode, logging-functions, 1.5.2023)

The screenshot above was taken before this change was made, but later the log-file writing was temporary done to an absolute path of /run/repbouncer.log to store and write the logs on RAM-memory, due to the number of debug-messages per second being excessive while still in testing phase.

6.2.7 Study: Fixing connection issues

The error message from UDP-connections suggested that the issue was in trying to use the same socket for multiple calls at once.

```
2023-05-01 20:39:25.608 Accepting UDP connection from 192.168.1.139:5353 with destination of 192.168.1.180:12348
2023-05-01 20:39:25.608 Failed to connect to original UDP source [192.168.1.139:5353]: dial: socket bind: address already in use
2023-05-01 20:39:26.612 Accepting UDP connection from 192.168.1.139:5353 with destination of 192.168.1.180:12348
2023-05-01 20:39:26.612 DEBUG: Checking UDP connection...
2023-05-01 20:39:26.612 Accepting UDP connection from 192.168.1.139:5353 with destination of 192.168.1.180:12348
2023-05-01 20:39:26.612 Failed to connect to original UDP source [192.168.1.139:5353]: dial: socket bind: address already in use
2023-05-01 20:39:29.675 Accepting UDP connection from 192.168.1.139:5353 with destination of 192.168.1.180:12348
```

(Screenshot, Bash-terminal, udp-errorlogs, 1.5.2023)

The error message came from tproxy.DialUDP()-call in handleUDPConn()-function, visible in the screenshot below on row 471. The error-message is created on row 473.

```go
462 // handleUDPConn will open a connection
463 // to the original destination pretending
464 // to be the client. It will when right
465 // the received data to the remote host
466 // and wait a few seconds for any possible
467 // response data
468 func handleUDPConn(data []byte, srcAddr, dstAddr *net.UDPAddr) {
469     updateLoggingVar(fmt.Sprintf("Accepting UDP connection from %s with destination of %s", srcAddr, dstAddr))
470 
471     localConn, err := tproxy.DialUDP("udp", dstAddr, srcAddr)
472     if err != nil {
473         updateLoggingVar(fmt.Sprintf("UDP: Failed to connect to original UDP source [%s]: %s", srcAddr.String(), err))
474         return
475     } else {
476         updateLoggingVar(fmt.Sprintf("UDP: Successfully connected to original UDP source [%s]", srcAddr.String()))
477     }
478     defer localConn.Close()
```

(Screenshot, VSCode, handleUDPConn(), 1.5.2023)

This handleUDPConn()-function was called as a goroutine from listenUDP()-function (on row 421) so it was likely to have had multiple simultaneous threads running at once.

```go
403 // listenUDP runs in a routine to
404 // accept UDP connections and hand them
405 // off into their own routines for handling
406 func listenUDP() {
407     for {
408         select {
409         default:
410             updateLoggingVar("DEBUG: Checking UDP connection...")
411             buff := make([]byte, 1024)
412             n, srcAddr, dstAddr, err := tproxy.ReadFromUDP(udpListener, buff)
413             if err != nil {
414                 if netErr, ok := err.(net.Error); ok && netErr.Temporary() {
415                     updateLoggingVar("Temporary error while reading data: " + netErr.Error())
416                 }
417                 log.Fatalf("Unrecoverable error while reading data: %s", err)
418                 return
419             }
420             updateLoggingVar("Accepting UDP connection from " + srcAddr.String() + " with destination of " + dstAddr.String())
421             go handleUDPConn(buff[:n], srcAddr, dstAddr)
422         case <-stopChannelUDP:
423             updateLoggingVar("DEBUG: UDP listener closing...")
424             udpListener.Close()
425             updateLoggingVar("DEBUG: UDP listener closed...")
426         }
427     }
428 }
```

(Screenshot, VSCode, listenUDP(), 1.5.2023)

Assuming that the issue was the first UDP-call hanging up the UDP-port, as seemed likely - this could have theoretically been resolved by running it via a normal function call, but this could have ended up creating quite a bottleneck on the network-throughput as only one UDP-connection could have been open at a time. Another solution could have been to have a pool of UDP-ports available that would have been iterated over until an open one was found every time UDP-connection needed to be created. A decision was made that the UDP-functionalities would be implemented once a working product was produced with only TCP-support. For now - all UDP-related functionalities were commented out.

TCP-proxy's functionality was verified, and a new problem surfaced – RepBouncer did otherwise work as expected when only TCP-connections were monitored, but when the original destination address was parsed from the net.Conn-object in the handleTCPConnection()-function, the parsing would return the IPS-system's address and connect to that. The solution was easy to describe at a high level, but potentially harder to fix – the destination address parsing should be fixed to point to the original destination.

```go
func handleTCPConn(conn net.Conn) {

    updateLoggingVar(fmt.Sprintf("Accepting TCP connection between %s and %s", conn.RemoteAddr().String(), conn.LocalAddr().String()))
    defer conn.Close()

    // FIX TODO: get the real original destination address      You, 1 second ago • Uncommitted changes

    if isAllowedIP(conn.RemoteAddr().(*net.TCPAddr).IP) {
        updateLoggingVar("Connection was allowed! Sending the packets along!")
        remoteConn, err := conn.(*tproxy.Conn).DialOriginalDestination(false)
        if err != nil {
            updateLoggingVar(fmt.Sprintf("TCP: Failed to connect to original destination [%s]: %s", conn.LocalAddr().String(), err))
            return
        } else {
            updateLoggingVar(fmt.Sprintf("TCP: Successfully connected to original destination [%s]", conn.LocalAddr().String()))
        }
        defer remoteConn.Close()

        var streamWait sync.WaitGroup
        streamWait.Add(2)

        streamConn := func(dst io.Writer, src io.Reader) {
            io.Copy(dst, src)
            streamWait.Done()
        }

        go streamConn(remoteConn, conn)
        go streamConn(conn, remoteConn)

        streamWait.Wait()
    } else {
        conn.Close()
        updateLoggingVar("Connection wad dropped due to banned address!")
    }
}
```

(Screenshot, VScode, handleTCPConn(), 3.5.2023)

The DialOriginalDestionation()-function was supposed to be able to parse the original destination, but for some reason it did not work as documented in this use case.

```
func (*tproxy.Conn).DialOriginalDestination(dontAssumeRemote bool) (*net.TCPConn, error)
DialOriginalDestination will open a TCP connection to the original destination that the client was trying to connect to before being intercepted.
When `dontAssumeRemote` is false, the connection will originate from the IP address and port that the client used when making the connection.
Otherwise, when true, the connection will originate from an IP address and port assigned by the Linux kernel that is owned by the operating system
 (tproxy.Conn).DialOriginalDestination on pkg.go.dev
DialOriginalDestination(false)
```

(Screenshot, VScode, DialOriginalDestination()-inline documentation, 3.5.2023)

The solution was setting the dontAssumeRemote to true, which seemed counterintuitive, but seemed that in this context "the client" in the documentation referred to the iptables. and the IP-address and port from the Linux kernel were the ones holding the true remote address. Perhaps incorrect firewall-settings were the root cause for this and for the UDP issues as well. But for the time being - changing that Boolean value allowed the TCP connections to pass at some volume as can be interpreted from the packet-capture from IPS-system that was taken after implementing the changes. These connections were visible in the repbouncer.log as well. Therefore, RepBouncer was now proven to be able to proxy the connection from client to server and vice versa.

```
1 0.000000000   192.168.1.215   145.131.132.90   TCP   76 43508 → 443 [SYN] Seq=0 Win=64240 I
2 0.000057972   145.131.132.90  192.168.1.215    TCP   76 443 → 43508 [SYN, ACK] Seq=0 Ack=1
3 0.000433072   192.168.1.215   145.131.132.90   TCP   68 43508 → 443 [ACK] Seq=1 Ack=1 Win=(
```

(Screenshot, wireshark, Successful TCP-handshake, 3.5.2023)

6.2.8 Feature: Implement white- and blacklist usage

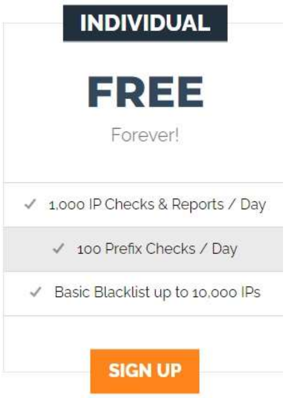
White- and blacklist usage was implemented next. The lists were implemented as global variables – On execution, the initial lists were fetched from white.list and black.list files and new findings were to be written to either of lists based on the Threat Intel API's response. For memory optimization purposes the lists should probably be fetched from files on demand in some way, as in this version the whole lists were constantly kept in heap-memory.

```go
func isInList(listName string, ip net.IP) bool {
    if listName == whiteListName {
        for _, ipadr := range whiteList {
            if ip.String() == ipadr {
                return true
            }
        }
    } else if listName == blackListName {
        for _, ipadr := range blackList {
            if ip.String() == ipadr {
                return true
            }
        }
    } else {
        log.Fatal("ERROR: unknown list given in isInList()")
    }

    return false
}

func isAllowedIP(ip net.IP) bool {
    if isInList(whiteListName, ip) {
        updateLoggingVar("Was on whitelist, letting pass...")
        return true
    } else if !isInList(blackListName, ip) {
        updateLoggingVar("Was NOT on blacklist, checking with ThreatIntelAPI...")        You, now • Uncommitted changes
        if isOKAccordingToThreatIntelAPI() {
            updateLoggingVar("Was OK according to ThreatIntelAPI, adding to whitelist")
            addToList(whiteListName, ip)
            return true
        } else {
            updateLoggingVar("Was not OK according to ThreatIntelAPI, adding to blacklist + dropping packets")
            addToList(blackListName, ip)
        }
    } else {
        updateLoggingVar("Was on blacklist - dropping packet!")
    }
    return false
}
```

(Screenshot, VScode, White-/Blacklist usage, 4.5.2023)

6.2.9 Feature: Implement Threat Intel API usage

VirusTotal API had the previously mentioned limits on the API-usage, with the most restricting one for this use case being the limit for maximum of 4 requests per minute. Hence the AbuseIPDB-API was used instead, which allows for 1000-ip checks a day using a free account.



(Screenshot, https://www.abuseipdb.com/pricing, Accessed 4.5.2023)

The API-usage was studied from the AbuseIPDB's documentation. Requests were sent to the check-endpoint using HTTP and the responses were JSON-encoded. The parameter of interest was called AbuseConfidenceScore which was an integer value between 0 and 100, where 0 was the most trustworthy and 100 the least. [28]

Function isOKAccordingToThreatIntelAPI() began by fetching the personal API-key from file. Then it sent the requests. All this is visible in the screenshot below.

```go
func isOKAccordingToThreatIntelAPI(ip net.IP) bool {
    //updateLoggingVar("DEBUG: In isOKAccordingToThreatIntelAPI")      You, 12 hours ago • Implement AbuseI
    cmdForGetAPIKey := exec.Command("bash", "-c", "cat abuseipdb_api_key")

    var stdout bytes.Buffer
    cmdForGetAPIKey.Stdout = &stdout

    err := cmdForGetAPIKey.Run()
    if err != nil {
        log.Fatal("ERROR: Could not get the api-key!", err)
    }

    apiKey := stdout.String()
    //updateLoggingVar("API_KEY was: " + apiKey)

    url := fmt.Sprintf("https://api.abuseipdb.com/api/v2/check?ipAddress=%s&maxAgeInDays=90", ip.String())
    req, err := http.NewRequest("GET", url, nil)
    if err != nil {
        log.Fatal("ERROR: Could not create the API request, error: ", err)
    }
    req.Header.Set("Key", apiKey)
    req.Header.Set("Accept", "application/json")

    // Send the request
    client := http.DefaultClient
    resp, err := client.Do(req)
    if err != nil {
        log.Fatal("ERROR: Could not send the API request! error: ", err)
    }

    // Read the response body
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatal("ERROR: Could not read the API response! error: ", err)
    }
```

(Screenshot, VScode, Sending the request to AbuseIPDB, 5.5.2023)

Afterwards the response was parsed and the AbuseConfidenceScore was read to determine if the address is malicious, a value of 10 was used as an initial threshold.

```go
// Read the response body
defer resp.Body.Close()
body, err := ioutil.ReadAll(resp.Body)
if err != nil {
    log.Fatal("ERROR: Could not read the API response! error: ", err)
}

// Parse the JSON response into a struct
type Response struct {
    Data struct {
        IpAddress            string `json:"ipAddress"`
        AbuseConfidenceScore int    `json:"abuseConfidenceScore"`
        CountryCode          string `json:"countryCode"`
        Domain               string `json:"domain"`
        Isp                  string `json:"isp"`
        IsWhitelisted        bool   `json:"isWhitelisted"`
        UsageType            string `json:"usageType"`
    } `json:"data"`
}
var response Response
err = json.Unmarshal(body, &response)
if err != nil {
    log.Fatal("ERROR: could not parse the API response! error: ", err)
}
if response.Data.AbuseConfidenceScore <= 10 {
    updateLoggingVar("AbuseConfidenceScore was: " + strconv.Itoa(response.Data.AbuseConfidenceScore) + " for: " + ip.String() + ", allowing...")
    return true
}
updateLoggingVar("AbuseConfidenceScore was: " + strconv.Itoa(response.Data.AbuseConfidenceScore) + " for: " + ip.String() + ", blocking...")
return false
```

(Screenshot, VScode, Parsing the API response, 5.5.2023)

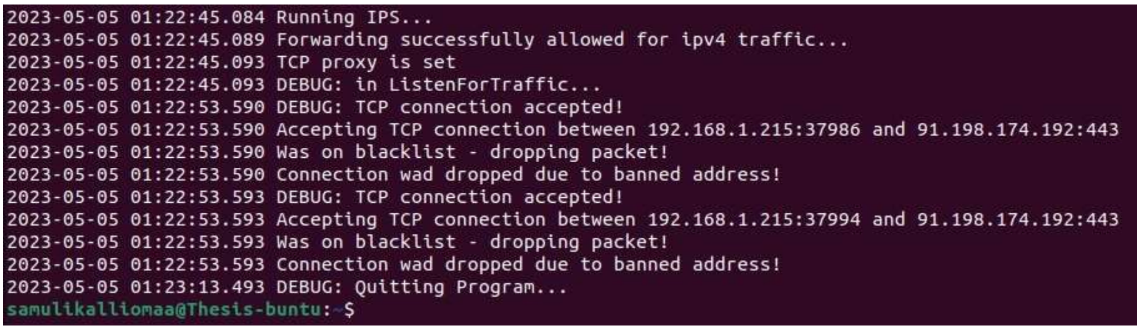
6.3 Final Testing

Now every functionality required for proof-of-concept was implemented, and it was time to verify the program works as expected.

```
2023-05-05 01:15:13.362 Running IPS...
2023-05-05 01:15:13.364 Forwarding successfully allowed for ipv4 traffic...
2023-05-05 01:15:13.365 TCP proxy is set
2023-05-05 01:15:13.365 DEBUG: in ListenForTraffic...
2023-05-05 01:15:21.220 DEBUG: TCP connection accepted!
2023-05-05 01:15:21.220 Accepting TCP connection between 192.168.1.215:40980 and 91.198.174.192:443
2023-05-05 01:15:21.220 Was NOT on either of lists, checking with ThreatIntelAPI...
2023-05-05 01:15:21.391 AbuseConfidenceScore was: 0 for: 91.198.174.192, allowing...
2023-05-05 01:15:21.391 Was OK according to ThreatIntelAPI, adding to whitelist
2023-05-05 01:15:21.392 TCP: Successfully connected to original destination [192.168.1.215:40980]
2023-05-05 01:15:23.653 DEBUG: TCP connection accepted!
2023-05-05 01:15:23.653 Accepting TCP connection between 192.168.1.215:54314 and 91.198.174.192:443
2023-05-05 01:15:23.653 TCP: Successfully connected to original destination [192.168.1.215:54314]
2023-05-05 01:15:25.673 DEBUG: Quitting Program...
samulikalliomaa@Thesis-buntu:~$
```

(Screenshot, bash-terminal, Successful AbuseIPDB-check, 5.5.2023)

Blocking malicious sites was tested by manually adding safe IP-addresses to blacklist to be connected to.



```
2023-05-05 01:22:45.084 Running IPS...
2023-05-05 01:22:45.089 Forwarding successfully allowed for ipv4 traffic...
2023-05-05 01:22:45.093 TCP proxy is set
2023-05-05 01:22:45.093 DEBUG: in ListenForTraffic...
2023-05-05 01:22:53.590 DEBUG: TCP connection accepted!
2023-05-05 01:22:53.590 Accepting TCP connection between 192.168.1.215:37986 and 91.198.174.192:443
2023-05-05 01:22:53.590 Was on blacklist - dropping packet!
2023-05-05 01:22:53.590 Connection wad dropped due to banned address!
2023-05-05 01:22:53.593 DEBUG: TCP connection accepted!
2023-05-05 01:22:53.593 Accepting TCP connection between 192.168.1.215:37994 and 91.198.174.192:443
2023-05-05 01:22:53.593 Was on blacklist - dropping packet!
2023-05-05 01:22:53.593 Connection wad dropped due to banned address!
2023-05-05 01:23:13.493 DEBUG: Quitting Program...
samulikalliomaa@Thesis-buntu:~$
```

(Screenshot, bash-terminal, Successful banning of blacklisted traffic, 5.5.2023)

Another point to mention was the timestamps – The delay of waiting for AbuseIPDB-response was around 170 milliseconds, which is very acceptable – especially as this would have to be executed only once for each new IP-addresses the client is connecting to. The overall internet speed should be tested using some online tester to determine the effect of the IPS-system in general.

6.4 Future Development

There were still numerous obvious features to have in this IPS-system, some of them are listed below.

- IPv6 support
- UDP support
- DNS support for domain name reputation as well
- White- and blacklist revocation loop
  The lists should be revocated with a period such as 30 days to ensure that the system has up to date threat intel.
- Unit-tests for all software-components
- Feature tests
- Verbosity levels configurable via CLAs
- Installation script that takes care of the configurations and dependencies for as far as possible
- Proper documentation for everything required to set this on a personal router.

7 Conclusions

This thesis was a versatile introduction into programming network-automation with Go for the author, similarly it could be used as such for anyone reading it.

Unfortunately, as the software was not yet developed with test-driven-development paradigm – there might be some uncertainty in some of the reasoning in the project diary.

This thesis proved the flexibility of Go in server-software and produced a proof-of-concept of IP-reputation based IPS-system for UNIX-routers.

The AbuseIPDB’s limit of 1000 requests per day did set some boundaries of usage of such system, but otherwise this project proved that it was possible to have free-of-charge IPS-system on a home-network with tolerable burden on the network speed.

Unfortunately, this system does require manual configurations that requires solid familiarity with computer networking and hence this does not seem like a viable resolution to IoT’s security risks at a larger scale.

Overall, this thesis and the project associated with it was a fun, challenging and educational experience for the author.

References

1 The Earthweb. Smart Home Statistics 2023: How many smart homes are there? [Internet]. [place unknown]: Wise; 2023 [updated 2023 Apr 08; cited 2023 Apr 12]. Available from: https://earthweb.com/smart-home-statistics/

2 Spiceworks. Malware threats can easily bypass antivirus software (know the limits of antivirus) [Internet]. [place unknown]: Locutus; 2020 [updated 2020 Oct 20; cited 2023 Apr 14]. Available from: https://www.spiceworks.com/it-security/data-security/articles/malware-threats-can-easily-bypass-antivirus-software/

3 Dryden Municipal Telephone System. What can’t a firewall protect against? [Internet]. Dryden (CA): Dryden Municipal Telephone System; [cited 2023 Apr 14]. Available from: https://www.dmts.biz/faqs-firewalls/what-cant-a-firewall-protect-against/

4 Cyberark. What is Defence-in-Depth? [Internet]. [place unknown]: Cyberark; [cited 2023 Apr 15]. Available from: https://www.cyberark.com/what-is/defense-in-depth/

5 National Cyber Security Centre. What is an antivirus product? Do I need one? [Internet]. UK: National Cyber Security Centre; 2019 Jan 21 [cited 2023 Apr 15]. Available from: https://www.ncsc.gov.uk/guidance/what-is-an-antivirus-product

6 Safety Detectives. How does antivirus quarantine work? [Internet]. [place unknown]: Glamoslija; [cited 2023 Apr 15]. Available from: https://www.safetydetectives.com/blog/how-does-antivirus-quarantine-work/

7 Safety Detectives. How does antivirus software work in 2023? [Internet]. [place unknown]: Kane; [cited 2023 Apr 15]. Available from: https://www.safetydetectives.com/blog/how-does-antivirus-software-work/

8 Fortinet. What is a Firewall? [Internet]. [place unknown]: Fortinet, inc.; [cited 2023 Apr 17]. Available from: https://www.fortinet.com/resources/cyberglossary/how-does-a-firewall-work

9 Techtarget. Definition: packet filtering [Internet]. [place unknown]: Wright; [updated 2023 Mar; cited 2023 Apr 17]. Available from: https://www.techtarget.com/searchnetworking/definition/packet-filtering

10 Securelist. Indicators of compromise (IOCs): how we collect and use them [Internet]. [place unknown]: Nazarov, Delcher, Sapronov; 2022 Dec [cited 2023 Apr 19]. Available from: https://securelist.com/how-to-collect-and-use-indicators-of-compromise/108184/

11 Virustotal. How it works [Internet]. [place unknown]: Virustotal; [cited 2023 Apr 20]. Available from: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

12 Cyber Management Alliance. Why is IP address data important for Cybersecurity & threat intel [Internet]. [place unknown]: Bishop; 2022 Jun [cited 2023 Apr 20]. Available from: https://www.cm-alliance.com/cybersecurity-blog/why-is-ip-address-important-for-cybersecurity-threat-intel

13 Github. Maltrail – Malicious traffic detection system [internet]. [place unknown]: stamparm; 2016 [updated 2023 Mar 1; cited 2023 Apr 21]. Available from: https://github.com/stamparm/maltrail

14 DOIT software. Go vs Python in 2023: Which one to choose? [Internet]. [place unknown]: Osadchuckto; 2023 Mar 26 [cited 2023 Apr 24]. Available from: https://doit.software/blog/go-vs-python#screen8

15 Geeksforgeeks. Go programming language (introduction) [Internet]. [place unknown]: Yadav; [updated 2023 Apr 24; cited 2023 Apr 24]. Available from: https://www.geeksforgeeks.org/go-programming-language-introduction/

16 Google. Frequently Asked Questions (FAQ) [Internet]. [place unknown]: Google; [cited 2023 Apr 24]. Available from: https://go.dev/doc/faq

17 Go by Example. Go by Example: Goroutines [Internet]. [place unknown]: McGranaghan, Bendersky; [cited 2023 Apr 24]. Available from: https://gobyexample.com/goroutines

18 Medium. When to use pointers in Go [Internet]. [place unknown]: Meeus; 2019 Nov 17 [cited 2023 Apr 24]. Available from: https://medium.com/@meeusdylan/when-to-use-pointers-in-go-44c15fe04eac

19 FAUN Publication. Understanding go.mod and go.sum [Internet]. [place unknown]: Janteshital; 2021 Jun 23 [cited 2023 Apr 24]. Available from: https://faun.pub/understanding-go-mod-and-go-sum-5fd7ec9bcc34

20 Microsoft. Visual studio code [Internet]. [place unknown]: Microsoft; [cited 2023 Apr 18]. Available from: https://code.visualstudio.com/

21 Ed25519: high-speed high-security signatures. Introduction [Internet]. [place unknown]: Bernstein, Duif, Lange, Schwabe, Yang; [cited 2023 Apr 18]. Available from: https://ed25519.cr.yp.to/

22 Exygy. Which License Should I Use? MIT vs. Apache vs. GPL [Internet]. [place unknown]: Morris; 2016 Jun 21 [cited 2023 Apr 18]. Available from: https://www.exygy.com/blog/which-license-should-i-use-mit-vs-apache-vs-gpl

23 Google. Standard library (go1.20.3) [Internet]. [place unknown]: Google; 2023 Apr 4 [cited 2023 Apr 18]. Available from: https://pkg.go.dev/std@go1.20.3

24 Github. GoPacket [internet]. [place unknown]: Google; [updated 2022 Aug 10; cited 2023 Apr 18]. Available from: https://github.com/google/gopacket

25 Github. gopacket linux undefined: pcapErrorNotActivated . Windows has no problem #629 [internet]. [place unknown]: Hilisecs; 2019 Apr 5 [cited 2023 Apr 20]. Available from: https://github.com/google/gopacket/issues/629

26 Fortinet. What is a Transparent Proxy? [Internet]. [place unknown]: Fortinet, inc.; [cited 2023 Apr 29]. Available from: https://www.fortinet.com/resources/cyberglossary/transparent-proxy

27 Google. Golang TProxy [Internet]. [place unknown]: Haworth; 2019 Jul 26 [cited 2023 Apr 29]. Available from: https://pkg.go.dev/github.com/LiamHaworth/go-tproxy#section-readme

28 AbuseIPDB. CHECK Endpoint [internet]. [place unknown]: AbuseIPDB; [cited 2023 May 4]. Available from: https://docs.abuseipdb.com/#check-endpoint