UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Monitoring network traffic and detecting attacks using eBPF

Author:
Sofiia TESLIUK

Supervisor:
Halyna BUTOVYCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

I, Sofiia TESLIUK, declare that this thesis titled, "Monitoring network traffic and detecting attacks using eBPF" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Monitoring network traffic and detecting attacks using eBPF

by Sofiia TESLIUK

Abstract

Network safety is a vital aspect of our current life. Lots of servers are becoming targets for attackers who want to reduce their performance or even get access to sensitive data. To prevent attacks, constant monitoring and analysis of network traffic is highly recommended and even required.

eBPF is an interesting technology of Linux that allows investigation and extension of kernel behavior, including access to raw network packets and their processing.

The main goal of this thesis is to explore the possibilities of eBPF in the context of creating a program for network traffic monitoring and analysis for attack prevention.

The project is open-sourced and will be available for further expansion and modification.

Acknowledgements

I want to thank my supervisor Halyna Butovych for guidance through the work on this thesis and for sharing the experience. And I am grateful to Oleg Farenyuk for helping to choose the area of this thesis and meeting me with my supervisor.

Also, I want to thank my family for constant support throughout my life and my friends Anastasiia and Elena for making these four years in Ukrainian Catholic University brighter.

Contents

List of Figures

List of Tables

List of Abbreviations

BPF	**B**erkeley **P**acket **F**ilter
eBPF	extended **B**erkeley **P**acket **F**ilter
DDoS	**D**istributed **D**enial of **S**ervice
SSH	**S**ecure **SH**ell
Rx	**R**eceiver
Tx	**T**ransmitter
CPU	**C**entral **P**rocessing **U**nit
SBA	**S**ignature **B**ased **A**pproach
ABA	**A**nomaly **B**ased **A**pproach
EBA	**E**ntropy **B**ased **A**pproach
SYN	**SYN**chronize
GCC	**GNU C**ompiler **C**ollection
LLVM	**L**ow **L**evel **V**irtual **M**achine
XDP	e**X**press **D**ata **P**ath
IP	**I**nternet **P**rotocol
LPM	**L**ongest **P**refix **M**atch
DAU	**D**aily **A**ctive **U**sers
MAU	**M**onthly **A**ctive **U**sers
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol

Dedicated to people, who haven't heard about eBPF before…

Chapter 1

Introduction

1.1 Motivation

During the exploration of eBPF, I was pleasantly surprised by its possibilities. eBPF can help solve daily tasks that require more information and influence processes run on servers and our personal computers. The range of possibilities goes from packet filtering to catching the event of opening a specific file in a filesystem.

In comparison to other ways to change the kernel behavior, eBPF doesn't require recompiling the kernel or compiling kernel modules and will be safe to run thanks to safety checks by loading verifiers of the code.

1.2 Problem

Today attacks have become a serious problem for everyone who launches a server with public access, which automatically makes a service a potential victim. Over the last years, strategies and tools for creating an attack have been improved drastically. Not only the quality of attacks improves, but the scalability and frequency increase as well.

DoS or DDoS ((Distributed) Denial of Service) is one of the most popular ones type of attack. Their goal is to bombard the target with a lots of requests in one moment, what makes the server unavailable for others. The average size of botnets army 2006, who cause DDoS attacks, is between $10^3 - 10^6$ [1]). Together they produce average load 1 Gbps, which is enough to take most services offline.

According to Cisco Annual Internet Report in the figure 1.1, 9.5 million DDoS attacks happened in 2019 and the number will increase in the next few years [3].

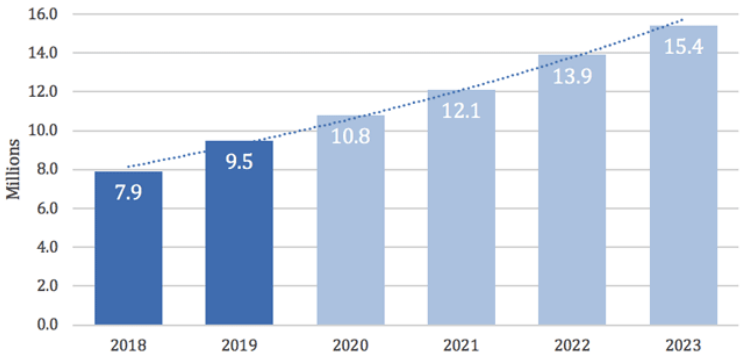

FIGURE 1.1: Data: Cisco Annual Report (2018-2023)[9]

Lots of Ukrainian services were attacked at some point as well. For example, in 2017 the services of "Boryspil" aeroport, Ukrainian postal service, Ukrzaliznytsia and others went offline because malvare that spreaded via service for accounting [11].

To successfully deal with attacks, when every millisecond plays its role, we need to detect and react to them very fast. Therefore access to the incoming network packet should be provided at an early stage of its processing.

Even though it would be more convenient to write a program that is being executed completely in user space, by the time we catch a network packet, it will have already gone through the checks done by the kernel. This leads to the conclusion that we can start extracting useful information and in some cases drop the packet at the first steps of kernel processing and won't use unnecessary CPU resources and memory space.

Applying changes to the kernel behavior is not an easy task and sometimes requires recompiling and replacement of kernel modules or the whole kernel. And where an update of the program has so many steps, postponing updates of it is inevitable and could fail to detect freshly introduced vulnerabilities.

eBPF provides an easier solution to change kernel behaviour on response to specific events. eBPF programs can be loaded to kernel at any time and don't require kernel recompilation, which is a great advantage comparing to alternative approaches.

Chapter 2

Related works

2.1 Similar tools for monitoring

The change in behavior in the general state of traffic directed to service can already provide hints about an existing attack. Just by a drastic increase of network bandwidth in 5 seconds, we could guess that something abnormal is happening. To find which information is useful for users, I investigated the functionality of similar tools.

I focused primarily on console tools for network monitoring, as most servers are located remotely and accessed via SSH in the console. The variety of tools for Linux is enormous, so I will list the most popular ones [8] or those that propose more unique features. Some of them are using eBPF as well.

2.1.1 Command-line tools

- **tcpdump** [7]
  - Is probably the most popular tool for network analysis
  - Monitors incoming packets with defined ip address / interface / port
  - Can show payload of the package
  - Supports different modes of shown information
  - Uses **libcap** that itself uses eBPF
- **Nload**
  - Monitors incoming and outgoing traffic separately (by network interface)
  - Shows general stats like:
    - Current network load (MBits/s)
    - Average network load
    - Min network load
    - Max network load
    - Total network load
- **Ifload / Iptraf**
  - Measures data flowing through individual socket connections
  - General stats
- **Nethogs**
  - Shows network bandwidth by an individual process

  - Sorts processes by intensivity

- Others: interesting points
  - Packet level details (tool **bmon**)
    - General number of received bytes
    - Number of errors
    - Number of Compressions
    - Number of dropped packets
    - Number of multicast packets
  - Top Rx/Tx speed (tool **slurm**)
  - Showing total size of the transferred data for whole time after launching recording daemon (tool **vnstat**)
  - Active socket connections (tool **Pktstat**)
    - Displaying the type of the current connection

2.1.2 Advanced tools

Advanced tools for network monitoring are oriented on a group of the servers and quite often appear to be commercial. Those tools commonly have a GUI that shows lots of details.

Interesting information shown in tools that I investigated [12]:

- State of network devices
- Topology of internal network
- CPU load
- Memory usage
- Activity of nodes
- Number of connections on each virtual server
- Monitoring firewall rules
- General statistics

2.2 DDoS attacks detection

2.2.1 Signature Based Approach

The main idea of the SBA approach is in knowing beforehand "bad" values of specific attributes and comparing to patterns that may point to malicious actions.

2.2.2 High traffic load

This type of attack is usually performed by an army of bots, sometimes installed on devices of peaceful citizens. Once the "main" host sends the command to start an attack, the whole army of bots, connected to the network starts bombarding the desired service.

Usually, services end up going offline and losing their reputation of being reliable, but sometimes these high traffic load attacks are a distracting maneuver for getting sensitive data using a found vulnerability.

The approaches Anomaly Based Approach (ABA) and Entropy Based Approach (EBA) are used for detecting hight traffic load attacks. Their main idea lays in comparison of the "normal" traffic to the observed one. In case observed traffic is much more heavy than the "normal" one, it points to an ongoing attack.

Either anomaly can be detected by comparison of traffic load over different segments of time or by comparing different channels at the same moment (for example load in separate socket connections).

To identify the abnormal traffic load over time. We need to collect the average / mean values of traffic load for the long continues time. Once current load is much greater than average traffic load the service should take proper actions.

The method called Fast Entropy Approach on Flow-Based Network Traffic is used to detect an attack by comparison of traffic load in active socket connections. It calculates the entropy of flow count during two adjusted segments of time for each socket that as well depends on flow count on other sockets. The calculated entropy value will drastically decrease for large flow counts [5]. This method doesn't require any additional information from the past and therefore requires relatively small resources and time for detection.

The SYN (synchronize) flood attack is one of the types of DDoS attacks. This attack is breaking the rule of TCP handshake by simply to responding to the server with ACK (Acknowledge) for SYN-ACK request. While server is waiting for the response from all bombarding bots, the memory space that keeps the information about each started connections may run out. This causes service unresponsiveness for valid users [4].

2.2.3 Malicious packet content

Revealing this type of attack is much harder than checking a source IP address in blacklist or finding anomalies in the number of load/connections. For this case, packets should go through a full examination of payload, which requires additional time and resources. Usually, such attacks are becoming noticeable much later after they have happened.

Chapter 3

eBPF

3.1 eBPF general overview

BPF is a tool available in Linux that allows executing custom programs in response to kernel events. BPF programs can be loaded from user-space with safety checks and be executed in kernel space, which provides a convenient updating of existing programs and access to more raw data.

The journey of BPF started in 1992 when it was introduced as a convenient way to catch packets for observation from user-level. It had a list of advantages and was declared to work 20 times faster than other ways of packet processing from user-space in those days [2].

The main points that made BPF an efficient way to make changes to the kernel behavior are:

- BPF is designed to run as a separate Virtual Machine, optimized to work with register-based CPUs.
- BPF programs use per-application buffers, which allows to avoid copying packet information, needed for processing.
- Before loading the BPF program to the kernel space, the verifier performs various checks to identify that the program is safe to execute and won't get into a never-ending loop.

Over the years Linux enthusiasts expanded BPF functionality to the extended version (also known as eBPF). Now eBPF works with 64-bit registers, instead of 32-bits. eBPF programs can be attached to other kernel events that are not only related to package receiving (for example, the start of executing a program or connecting a new device by USB). And eBPF functionality is more exposed to user-space for inserting custom actions.

A typical eBPF usage:

1. Writing a script that operates with data, available in the scope of the attached event.
2. Compiling this script (it can be done with standard compilers like GCC or LLVM).
3. Loading a compiled program to the kernel space, in case of successful checks by the verifier.
4. Attaching the loaded BPF program to the specific network interface/file/etc. On which it should be triggered.

Currently, the size of loaded BPF program is limited to 4Kb. In case user wants to expand a program, it can be done by creating an ordered chain of loaded BPF programs with the same type, where every program can continue execution in the scope of the next program.

Logs in the BPF program can be created with **bpf_trace_printk** and found in **debugfs**.

3.2 XDP program type

XDP is one of the few BPF program types. It is executed on the early stage of network package arrival when we can extract needed information. To mark the BPF program as XDP one, on loading the program to the kernel space user should add a flag BPF_PROG_TYPE_XDP.

The XDP program type also supports performing specific actions to the package itself. The most common ones, which I have used as well, are:

- XDP_PASS – pass the network package for further procession
- XDP_DROP – drop the package and terminate its execution at this moment

In more unique situations, the following ones can be used:

- XDP_ABORTED – terminate package procession with tracepoint exception
- XDP_TX – send back by the same network interface, from which it came
- XDP_REDIRECT – redirect package to another network interface

Also, the XDP type of BPF programs provides helpful structures for unwrapping network packets, such as **ethhdr** (for Ethernet packet) and **iphdr** (for IP packet).

After loading a XDP program to the kernel space, it can be attached to the network interface which we want to observe, or even to several ones. Detachment of the program from the network interface can be done at any time with a helper tool **bpftool**. In case users would like to try to attach several XDP programs to the same network interface, they would fail to do so since only one program is allowed to be attached to the same network device.

XDP programs are probably the most common ones. But it is worth mentioning the existence of such types as “Socket Filter Programs” and “Socket Option Programs” that allow observation of flowed packets and modifying options of the specific socket connection as well as give an even more detailed overview of a network.

3.3 eBPF maps

Once a packet is successfully processed and we have received all the needed information in kernel space, the question about passing the collected information to the user space comes up. And that is the moment when eBPF maps are used.

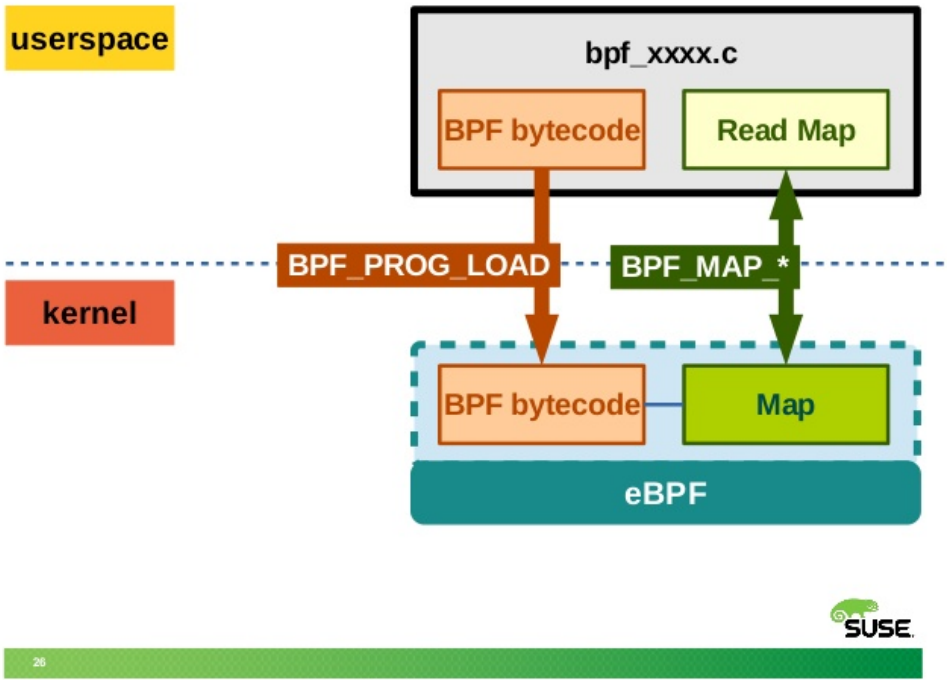

FIGURE 3.1: Location of eBPF maps [6]

eBPF maps are data structures that are accessible from user space and from BPF programs in kernel space 3.1. They behave as key-value storage for different purposes. For example, as hash-table, array, queue, stack, or even for more specific purposes as LPM tries (Longest Prefix Match) etc. But the size of eBPF maps should be predefined on the load of the BPF program and it sums up to the general limitation of 4KB of the loaded BPF program.

Most variations of eBPF maps support 3 actions, such as:

- Lookup for the value by provided key
- Updating the value by provided key
- Deleting the whole element by provided key

From userspace, one more action is available – getting the next key of the map. Overall, the current possibilities of eBPF maps are enough to fulfill general needs.

Chapter 4

Background information

4.1 HyperLogLog

HyperLogLog, the descendant of the LogLog algorithm, is a great way for estimating the distinct values over a data stream with repeated values. Compared to other count distinct algorithms over a data stream, HyperLogLog promises to reach a high accuracy < 2% for cardinality values $N = 10^9$, while using relatively small memory space [10]. The count distinct problem is often used for estimation of unique IP addresses or other attributes from constant flow.

Let's introduce a hashing function $h(v)$, that converts a value from the observed data stream to a binary sequence, and a function $r(x)$, called rank, that finds the position of rightmost 1-bit out of provided binary sequence.

The main idea of the HyperLogLog algorithm and its relatives is in finding the estimate of the number of unique value in data-stream by known probability for appearance of each value of the rank.

For example, from uniformly distributed hashes, the probability to get $r(x) = 1$ is 50%, for $r(x) = 2$ is 25%, for $r(x) = 3$ is 6.52%, and so on. If we remember only the highest value of the rank function, we can already extract an estimation $E = 2^{max(r)}$. Of course, with such approximate estimation, some outliers will make the difference between estimation and actual number way too big.

The HyperLogLog includes a lot of improvements that make the estimation more accurate. First of all, we need to keep in memory more than 1 rank value. Secondly, the calculation of estimation contains more steps to make it closer to the actual number.

To split the hashed values into separate groups, within which their ranks will be compared, we will introduce the function that converts first b bits into value, called index $j(x)$. For this, we need to keep in memory $m = 2^b$ values for comparisons. Let's introduce an array $M[1]$, $M[2]$ ... $M[m]$, called registers in which we will keep comparing values identified by their index.

From this point, once an element arrives from observed data stream, the next process starts:

1. Calculating the hash of the arrived element $x = h(v)$, t – total number of bits in x
2. Finding index value out of first b bits of the hashed element $i = j(x[0 : b - 1])$

3. Finding the rank of last $(t - b)$ bits – $r(x[b : t - 1])$

4. Assign to $M[i]$ the higher value between $M[i]$ and rank

In the end, to find the estimation, we can simply calculate the harmonic mean of values in the registers array and multiply it by correcting coefficients ($a_m * m^2$ below).

The final result:

$$E = \frac{a_m \cdot m^2}{\sum_{i=1}^{m} 2^{-M[i]}}$$

Where:

$$a_m = \left(m \int_0^\infty \left(log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}$$

4.2 Knuth Multiplicative method

Knuth multiplicative method is a simple hashing function for integers. By multiplying the value to the golden ratio of 2^{32} (= 2654435761). To keep the result limited to 32 bits, we can take the last 32-bits of the hashed value.

$$f(x) = (x \cdot 2654435761) \ mod \ 2^{32}$$

This hashing method gives uniformly distributed results. And its simplicity saves time.

Chapter 5

Implementation

The created program, named monitorBX, consists of two parts – the user space and kernel space (in the context of eBPF) parts. eBPF part extracts the needed information from packets and passes it to the user-space part, while the user-space part of the program collects data, processes it, and outputs results to the user.

The program is written in C, to simplify the preparation process for its execution. For C eBPF features are available after installation **libbpf** library, while other languages require installing additional libraries.

Since eBPF is a subject of constant improvement and its API is changing with kernel versions, the BPF part of the code is compatible with Linux kernel version 5.10.3.

Currently, monitorBX has two modes of execution:

- General monitoring
- Estimating the number of unique IP addresses

Before the start of the execution, user can define:

- The mode in which monitorBX should be launched
- Index of network interface, over which the traffic will be monitored
- Path to files, where information will be stored

To find index of network device that will be observed, you can simply find a first number appeared next to network device description with command **ip addr**.

5.1 General monitoring

General statistics in output includes:

- Number of received packets
- Number of dropped packets
- Speed of received traffic
- Number of unique source IP addresses
- Number of unique destinations ports

- Overview of source IP addresses and destination ports – how many times each IP address / port was met (the collected overview won't be shown in the console, but will be saved to the file)
- Number of packet with TCP/UDP/Other protocol

The information points above should be enough to detect some anomalies in the traffic. Special mode for saving each package will allow custom detailed exploration of the attack's existence by the user.

5.1.1 eBPF part

The eBPF part of the program is responsible for extracting useful information from the caught packets. At this stage we look for this information in the packet:

- Source IP address
- Destination port
- Protocol

To share the information with user-space part, eBPF maps are defined as in table 5.1, where maps include:

- conf_map
  - to drop packages
- values_map
  - Total size of flow
  - Number of passed packets
  - Number of dropped packets
  - Number of packets with TCP protocol
  - Number of packets with UDP protocol
  - Number of packets with Other protocol

Name	Type	Size
conf_map	BPF_MAP_TYPE_ARRAY	1
values_map	BPF_MAP_TYPE_ARRAY	6
ip_map	BPF_MAP_TYPE_HASH	100
port_map	BPF_MAP_TYPE_HASH	20

TABLE 5.1: General monitoring: defined eBPF maps

In XDP program type, the packet comes as Ethernet packet along with Ethernet interfaces. The structure of the Ethernet packet with TCP protocol is shown in 5.3. To extract the source IP address and destination port, as mentioned earlier the next helping structures are used: **ethhdr**, **iphdr**, **tcphdr**, **udphdr**.

The steps of extracting the needed information are shown in 5.2.

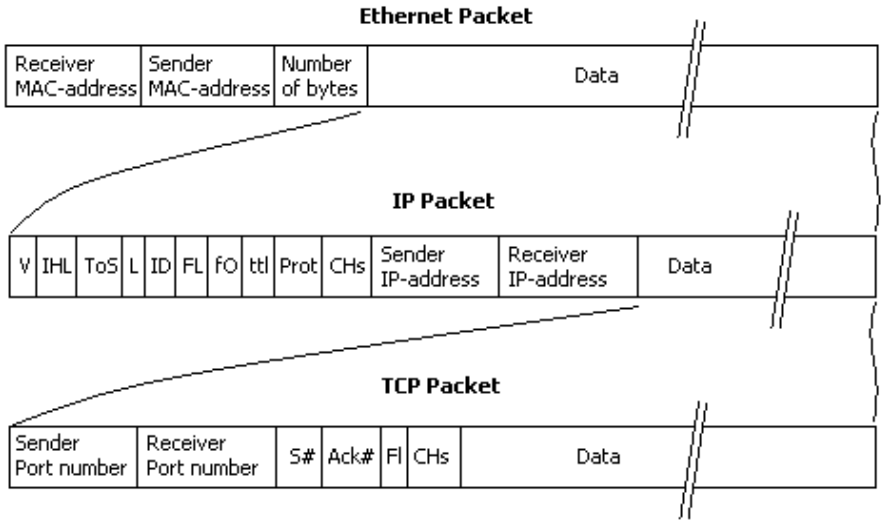

FIGURE 5.1: TCP packet structure

5.1.2 User-space part

User space part is the maintainer of the process. It loads the eBPF part to the kernel space, attaches it the chosen by user network device and accessed the maps shared with the loaded eBPF program. After this, it periodically checks the state of the maps, aggregated data and outputs it either to console or provided data file.

5.2 Estimating the number of unique visitors

An important metric for server maintainers to know is the number of users of the service over time. Usually, such metrics as DAU (Daily Active Users) and MAU (Monthly Active Users) are tracked by the service itself. For web servers, the uniqueness of the user is determined by the event of the authentication / information in the header of the HTTP request or by cookie. In case you don't want to introduce any major changes with tracking features to the service, but still want to collect metrics, the use of eBPF and estimation algorithms can help with estimating the number of unique users that will be identified by their IP address or a combination of other flow parameters as IP + port, session ids etc.

The straightforward solution, for counting unique users, would be saving every new ip address to the data storage, and periodically count the number of elements in used data storage. This method will provide accurate results, but as long as the service works, the memory used for data storage will increase as well. This might become a big deal if the server is running for a long time or for the one which serves a lot of users.

As mentioned in the earlier section, there exists HyperLogLog algorithms for estimation of distinct values in a large data stream, by using relatively small data storage. I added this estimation as well, since it can be a good way to save memory space to get the needed metrics.

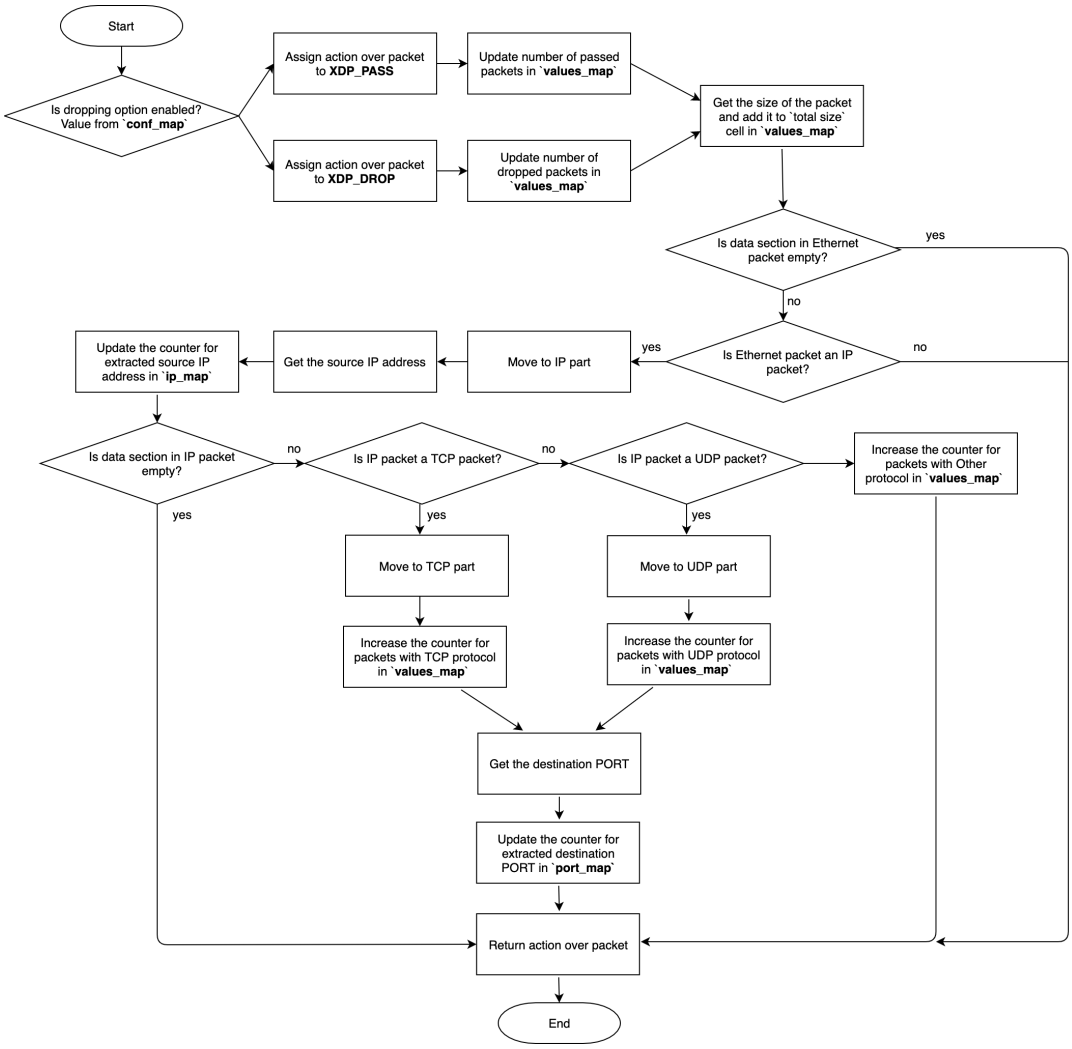

FIGURE 5.2: General monitoring: eBPF part block scheme

5.2.1 eBPF part

At first, we need to have an array of size that is power of 2 defined as m. A BPF map of type array is used as this required data storage, so later we will have access to it from user-space. The defined map is shown in the picture 5.2

Name	Type	Size
registers	BPF_MAP_TYPE_ARRAY	100

TABLE 5.2: Estimating the number of unique visitors: defined eBPF maps

The next step was to choose a hash function that would convert a received IP address to a hash that is uniformly distributed over 32-bit integers. For this purpose, the Knuth's Multiplicative Method is used.

The block scheme of is shown in figure 5.2.

The part with extracting source IP address from the packet can be found in 5.1.

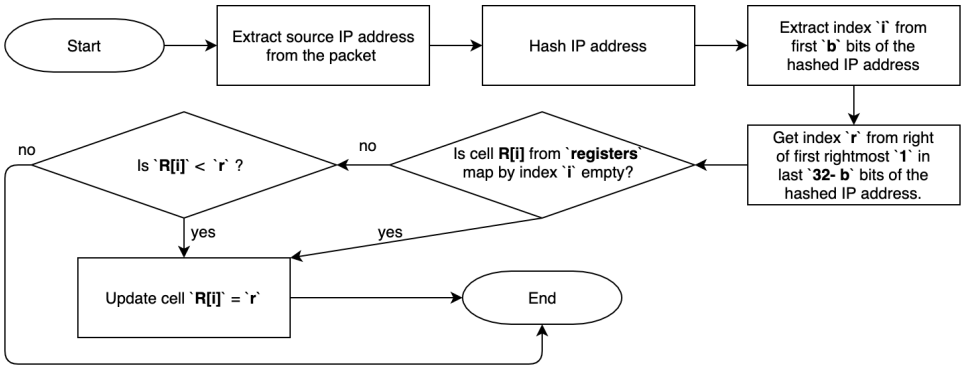

FIGURE 5.3: Estimating the number of unique visitors: eBPF part block sheme

5.2.2 User-space part

The beginning of user-space part in count distinct mode includes loading BPF part, finding maps and attaching eBPF part to the network interface as well as it is done in general monitoring mode. After that It iterates over registers map periodically and counts the Harmonic mean of ranks. Estimation is calculated by multiplication of Harmonic mean of the ranks to correcting coefficient.

5.3 Testing

To test both modes of monitorBX, I needed to emulate an active network flow. To do this, I wrote a python script that uses **scapy** library for packet generation and sending.

The script sent generated Ethernet packets to the defined destination IP address with configured frequency. Generated packets are with spoofed source IP addresses.

Chapter 6

Results and Summary

6.1 Results

Currently the tool for network monitoring is ready to use to get a general overview of the current network state. The user have an access to the current and collected before stats. This allows user custom investigation of possible past anomalies.

The exprolation of HyperLoglog algorithm did not give that great results for smaller cardinality ($N = 10^3 - 10^6$ (for an average size of botnets), as it was promised for cardinality $N = 10^9$. The estimation is compared to the actual number in table 6.1, for size of registers array $m = 128$, an $b = 7$. Nevertheless, this method might be used for much major attacks, but they are less likely to happen. For sure the accuracy the HyperLogLog method is not enough for getting metrics about active users with a balanced every-day flow.

Actual number A	Estimate E	Relative error $\epsilon_A = \frac{E-A}{A}$
1000	3989	2.99
10000	23622	1.36
100000	46154	0.538

TABLE 6.1: Estimation results for m = 128, b = 7

Estimation of unique IP addresses requires further investigation and mixing with other known approaches for counting distinct values. For example, the program can start estimating with methods for smaller cardinality, like Linear Counting, but once the greater cardinality is detected, the other approach can be enabled. Or by using the Sliding HyperLogLog [13] which adjust to detected cardinality.

6.2 Possible improvements

The field of improvement in aspect of estimating features and attacks detection is wide and can be improved into two directions:

- Metrics estimation
- Attacks detection

Attacks detection highly depends on metrics and that is the reason why they observed alongside.

Detailed points for improvement:

- Finding more accurate method for estimation specific parameters
- Automatization of attack identification and notification system for administration with alarms about possible attack

- Analysis of collected data for servers which are very rarely turned off
- Change the configuration of executed program during runtime (for this, user won't need to stop the execution and launching with other parameters)
- More user-friendly interface

Also, as eBPF allows operating with socket connections as well, there is a possibility to add analysis of the traffic flow in socket connections as well.

6.3 Summary

In this thesis I investigated the possibilities of eBPF and approaches for attacks detection in context of creating a tool for network monitoring. It gives a general overview of the current state and access to collected data that can be used for further attack analysis.

The source code is available here `https://github.com/sofiia-tesliuk/monitorBX`

Bibliography

[1] *Botnet @ Radware*. 2021. URL: https://www.radware.com/security/ddos-knowledge-center/ddospedia/botnet.

[2] David Calavera and Lorenzo Fontana. *Linux Observability with BPF*. 1005 Gravenstein Highway North, Sebastor, CA 96472: O'Relly Media, Inc., 2020.

[3] *Cisco Annual Internet Report (2018-2023)*. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[4] imperva. *TCP SYN Flood*. URL: https://www.imperva.com/learn/ddos/syn-flood/.

[5] Ciza Thomas Jisa David. *DDoS Attack Detection using Fast Entropy Approach on Flow-Based Network Traffic*. 2015. URL: https://core.ac.uk/download/pdf/82617302.pdf.

[6] Al Cho @ SUSE Labs. *eBPF maps 101 (Presentation)*. 2018. URL: https://www.slideshare.net/suselab/ebpf-maps-101.

[7] *Man page of TCPDUMP*. URL: https://www.tcpdump.org/manpages/tcpdump.1.html.

[8] Silver Moon. *18 Commands to Monitor Network Bandwidth on Linux server*. 2020. URL: https://www.binarytides.com/linux-commands-monitor-network/.

[9] Paul Nicholson. *Five most Famous DDoS Attacks and Then Some*. 2020. URL: https://www.a10networks.com/blog/5-most-famous-ddos-attacks/.

[10] Olivier Gandouet Philippe Flajolet Eric Fusy and Frederic Meunier. *HyperLogLog: the analysis of near-optimal cardinality estimation algorithm*. 2007. URL: http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf.

[11] RBC.UA. *Networks attacks: how dangerous is new cyberthreat is for Ukraine*. 2020. URL: https://daily.rbc.ua/ukr/show/ataki-set-naskolko-opasna-novaya-ugroza-ukrainy-1595856741.html.

[12] Marc Wilson. *11 Best Network Monitoring Tools Software of 2021*. 2021. URL: https://www.pcwdld.com/best-network-monitoring-tools-and-software.

[13] Georges Hébrail Yousra Chabchoub. *Sliding HyperLogLog: Estimating cardinality in a data stream*. 2010. URL: https://hal.archives-ouvertes.fr/hal-00465313/file/sliding_HyperLogLog.pdf.