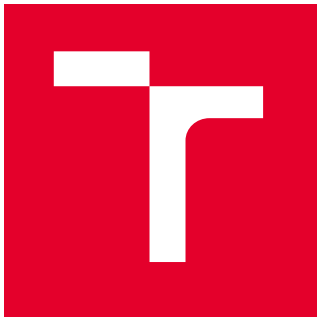
BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2022 Bc. MICHAEL JUREK



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

DETECTION OF MODERN SLOW DOS ATTACKS

DETEKCE MODERNÍCH SLOW DOS ÚTOKŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. Michael Jurek

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Marek Sikora

BRNO 2022

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study program **Information Security**

Department of Telecommunications

Student: Bc. Michael Jurek ***ID:*** 182503

Year of study: 2 ***Academic year:*** 2021/22

TITLE OF THESIS:

Detection of modern Slow DoS attacks

INSTRUCTION:

The SlowDrop and Slow Next attacks are among the latest so-called slow DoS attacks. Their main characteristic is a very faithful imitation of legitimate users with a slow internet connection. The attacks do not contain any invalid communication, nor do they show any significant signature. Therefore, addressing the effective detection of these attacks is a major challenge for security professionals.

The task of the diploma thesis is to study in detail the characteristics of SlowDrop and Slow Next attacks, to design and create a laboratory network with legitimate users, to use available attack generators and to apply them in a laboratory network. Another task is to design and implement the most appropriate methodology for detecting these attacks. The aim of this work is to create a tool with the highest possible detection accuracy. The resulting solution should achieve acceptable accuracy for real network deployments.

RECOMMENDED LITERATURE:

[1] CAMBIASO, Enrico, Gianluca PAPALEO, Giovanni CHIOLA a Maurizio AIELLO. Designing and Modeling the Slow Next DoS Attack. International Joint Conference. Cham: Springer International Publishing, 2015, 2015-5-27, 54(4), 249-259. Advances in Intelligent Systems and Computing. DOI:10.1007/978-3-319-19713-5_22

[2] MAZÁNEK, Pavel. Modelování a detekce útoku SlowDrop. Brno, Rok, 75 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Marek Sikora

Date of project specification: 7.2.2022 ***Deadline for submission:*** 24.5.2022

Supervisor: Ing. Marek Sikora

Consultant: Enrico Cambiaso (National Research Council, Italy)

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

ABSTRACT

With the evolving number of interconnected devices, the number of attacks arises. Malicious actors can take advantage of such devices to create (D)DoS ((Distributed) Denial of Service) attacks against victims. These attack are being more and more sophisticated. New category of DoS attacks was discovered that tries to mimic standard user behavior – Slow DoS Attacks. Malicious actor leverages transport protocol behavior to the highest option by randomly dropping packets, not sending or delaying messages, or on the other hand crafting special payloads causing DoS state of application server. This thesis proposes parameters of network flow that should help to identify chosen Slow DoS Attack. These parameters are divided into different categories describing single packets or whole flow. Selected Slow DoS Attack are Slow Read, Slow Drop and Slow Next. For each attack communication process is described on the transport and application layer level. Then important parameters describing given Slow DoS Attack are discussed. Last section sums up methods and tools of generation of these attacks. Next part deals with possibilities and tools to create such an attack connection, discuss basic communication concepts of creating parallel connections (multithreading, multiprocessing) and proposes own Slow DoS Attack generator with endless options of custom defined attacks. Next part describes testing environment for the attack generator and tools and scenarios of data capture with the goal of dataset creation. That dataset is used for subsequent detection using machine learning methods of supervised learning. Decision trees and random forest are used to detect important and drop redundant features of selected Slow DoS Attacks.

KEYWORDS

Anomaly-Based Detection, DoS, Decision Trees, Flow ID, Generator, LDoS, Machine Learning, Python3, Random Forest, Slow DoS Attacks, Slow Read, Slow Drop, Slow Next

ROZŠÍŘENÝ ABSTRAKT

V dnešním propojeném světě ovlivňuje internet každý aspekt lidského života. Zároveň roste počet propojených zařízení, na které nemá člověk přímo vliv, ale které jej naopak mohou ovlivnit. Jak se člověk stává závislejší na moderních technologiích, je pro něj klíčová jejich dostupnost. Vždy se najdou entity, které se snaží o znepřístupnění takových služeb (D)DoS ((Distributed) Denial of Service) pro běžného uživatele.

S rozvojem moderních IPS (Intrusion Prevention System) a IDS (Intrusion Detection System) systémů, prostředků antivirové ochrany či firewallů je pro útočníky čím dál tím obtížnější zamaskovat své chování a vyhnout se tak detekci. Útočníci proto vytvářejí nové typy útoků, které jsou čím dál tím víc sofistikovanější a snaží se vyhnout případné detekci.

Nová kategorie DoS útoků, která napodobuje standardní chování uživatele v síti, byla objevena a pojmenování Slow (pomalé) DoS útoky, někdy nazývané Low-rate nebo Low-bandwidth útoky. Útočník se snaží využít maximálních možností transportního protokolu, pomocí něhož jsou veškerá data přenášena. Mezi pozorované chování útočníka patří náhodné zahazování paketů, neodesílání nebo zdržování odesílání zpráv, oznamování protistraně o nemožnosti příjmu dat, pomalém čtení dat nebo úmyslné vytváření paketů, jejichž payload obsahuje taková data, která způsobí zaneprázdnění aplikačního serveru.

Tato diplomová práce pojednává právě o pomalých DoS útocích. V první kapitole jsou probrány základní koncepty komunikace pomocí protokolu TCP, otevírání spojení pomocí trojcestného podání ruky, standardní zavírání spojení a zavírání spojení způsobené nestandardní událostí. Dále zde je uveden popis fungování aplikačního protokolu HTTP, jenž může být použit jako nadstavbový protokol a je použit v následujících částech. Dále jsou zde popsány parametry spojení podle místa (klient, server, síť) a směru detekce (odesílací a přijímací), které mohou být použity při tvorbě IPS a IDS systémů. Zároveň jsou tyto parametry rozděleny do třech kategorií podle toho, zda-li se podílejí na tvorbě síťového toku, či nikoli. General parametry jsou použity pro popis standardního síťového chování. Intra-flow parametry slouží k popisu jednotlivých paketů a Inter-flow parametry slouži k popisu abstrahovaného toku dat. Nejdůležitější částí Inter-flow parametrů je identifikátor toku dat (Flow ID), který slouží ke seskupení paketů v rámci jednoho spojení. Tvorba tohotu parametru je možna více způsoby. V této práci je aplikován postup, který byl použit při tvorbě datového souboru CIC-IDS2017 z důvodu jednoduššího připojení vlastních dat. Dalšími kategoriemi jsou parametry týkající se objemu a velikosti přenášených dat, časové parametry, které umožňují dělení pomalých DoS útoků a parametry vlastností, mezi které se řadí například TCP příznaky a výsledky TCP analýzy programu Wireshark. Poslední kategorií jsou aplikační parametry.

Druhá kapitola uvádí rozdíl mezi DoS a DDoS útoky a dále rozděluje DoS útoky na kategorii záplavových útoků a útoků využívajících zranitelnost. Kategorie pomalých DoS útoků je uvedena jako podkategorie DoS útoků. V další části je uvedeno rozdělení pomalých DoS útoků do jednotlivých kategorií podle typu chování a podle časových parametrů. Časové parametry jsou dále napojeny na jednotlivé kategorie.

Třetí kapitola se zabývá vybranými pomalými DoS útoky. V této práci byly vybrány útoky Slow Read, Slow Drop a Slow Next, jejichž chování je dále popsáno z pohledu transportní komunikace. Dále jsou uvedeny důležité parametry jednoznačně popisující chování, metody a nástroje umožňující jejich vytvoření.

Ve čtvrté kapitole je navržen a implementován ve skriptovacím jazyce `python3` generátor pomalých DoS útoků. Nejrpve jsou popsány jednotlivé moduly umožňující tvorbu HTTP požadavků s přístupem k transportnímu protokolu, kde byla vybrána knihovna `socket`. Dále je zde popsán mechanismus chování útoků z pohledu uzavírání spojení (rozdíl použití příznaků `FIN` a `RST`). V další části je popsán způsob asynchronní komunikace a paralelní provedení jednotlivých spojení s použitím více vláken nebo více procesů. V další části je navrhnut samotný generátor. Jsou popsány použité moduly potřebné k jeho tvorbě. Dále je uvedené chování a nastavení generátoru a popsány možnosti tvorby vlastního pomalého útoku. Dále je specifikována vlastnost generátoru umožňující logování událostí do standardního výstupu nebo do externího soboru.

Pátá kapitola popisuje testovací prostředí, výchozí nastavení webového serveru Apache ve verzi 2.4.49 a použití výchozích modulů pro správu více spojení a jejich časových limitů sloužících k ukončení spojení na straně serveru. Dále jsou popsány možnosti rozšíření modulů o bezpečnostní moduly s možností ochrany webového serveru proti pomalým DoS útokům. Následuje popis tvorby vlastního datového souboru. Nejprve je probrána možnost zachycování síťových dat v reálném čase a dále jsou popsány existující datové soubory, které je možno využít a rozšířit o vlastní pomalé DoS útoky. Taktéž je zde popsán postup tvorby vlastního datového souboru obsahující označený síťový tok.

V šesté kapitole je uvedeno několik možností detekce pomalých DoS útoků. Mezi dvě hlavní kategorie detekce patří detekce signatur a detekce anomálií. V této práci je kladen důraz na detekci anomálií, avšak jsou zde uvedeny i příklady možných signatur.

Sedmá a poslední kapitola popisuje detekci anomálií s použitím strojového učení s učitelem. Nejrpve jsou popsány jednotlivé metody strojového učení (s učitelem, bez učitele a zpětnovazebné učení) s příklady. Dále jsou probrány dvě metody strojového učení s učitelem, a to rozhodovací stromy a náhodné lesy, které jsou tvořeny právě těmito stromy. Tyto metody jsou použity pro detekci významných parametrů síťového toku determinující pomalý DoS útok. V první části této detekce je pop-

sán způsob výběru, tvorby a sběru dat. Je zde vytvořena převodní tabulka mezi různými datovými soubory a jeden výsledný datový soubor, jenž obsahuje záznam síťového toku po dobu jednoho týdne provozu spolů s označením, zda se jedná o normální provoz nebo o jednotlivý typ útoku. Dále dochází k předzpracování dat, kde jsou chybějící data vynechána, nenumerické hodnoty nahrazeny numerickými pro strojové zpracování a vynechání sloupců dat, které by mohly zkreslovat průběh detekce. V další části jsou vybrány pouze označené pomalé DoS útoky, a k tomu je naškálován normální provoz v poměru (30:70). Dochází tedy k redukci datového souboru. Další část je výběr parametrů, kde je vytvořen rozhodovací strom, který určuje důležité parametry, ty jsou následně křížově validovány. Je vytvořena korelační matice a silně korelované sloupce jsou vynechány. Pro spřesnění lze využí optimalizačních metod pro různe nastavení náhodných stromů, avšak původně zvolené nastavení dosahuje výborných výsledků. Výsledný model je dále serializován a uložen. Takto vytvořený model může být použit pro detkci vybraných pomalých DoS útoků ve firewallech či IDS nebo IPS systémech.

JUREK, Michael. Detection of Modern Slow DoS Attacks. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2022, 127 p. Master's Thesis. Advised by Ing. Marek Sikora

Author's Declaration

Author:	Bc. Michael Jurek
Author's ID:	182503
Paper type:	Master's Thesis
Academic year:	2021/22
Topic:	Detection of Modern Slow DoS Attacks

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to thank the supervisor of my thesis, Ing. Marek Sikora for his academic lead and valuable comments and the advisor, Dr. Enrico Cambiaso for valuable comments, advice and adjustment of direction. I would like to thank my family for the support.

Contents

List of Figures

List of Tables

List of Listings

Introduction

In today's world, the Internet is affecting almost every aspect of human life. In most cases for the better purpose. Provides useful tools for everyday communication which shorten distances between people and allows remote work for professionals. It provides any kind of information in almost every point in time. Internet has no physical boundaries, it can span across the oceans in modern fiber optic cables, through space using satellites. From huge server farms through smart houses, electric cars to tiniest personal devices, such as wearable, IoT (Internet of Things) gadgets and medical equipment.

But there is another side of the coin. Due to overall acceptance and spread, it allows people with bad intentions to do whatever they want. There are malicious users, hackers and attackers among legitimate ones. Their behavior can have destructive impact on the critical infrastructure, hospitals, military facilities, power plants, universities, governments, public movements, freedom to vote or freedom of speech.

In the computer networks and information security we can define model of security policies – the CIA triad (Confidentiality, Integrity and Availability). Confidentiality stands for an effort to keep data private and secret. Basically, it allows only authorized users to access data. It can be achieved by user or data authentication, authorization, access control, strict data classification and encryption. It can be violated by brute force attacks or man-in-the-middle attacks etc. Integrity assures that some data has not been modified by a malicious user. It can be accomplished by cryptography means – encryption, hashing, digital signatures. On the other side auditing, version control and IDS (Intrusion Detection System) with logging. The last Availability ensures that all hardware and software resources keep reliably working. It also assumes that all the resources are updated, redundant and fault tolerant. When the accident happens, it is essential that service can fast recover. [4]

This thesis is focused on the last part of the triad – Availability. Malicious user can behave way that some target resources transit to disabled state and are unavailable or below acceptable threshold for the legitimate users. This behavior is called DoS (Denial of Service).

As the number of interconnected devices is raising, malicious actors try to develop new methods of DoS attacks. On of the newest categories of DoS attacks are SDAs (Slow DoS Attacks) sometimes called (Low-rate or Low-bandwidth DoS Attacks). They try to mimic user behavior to avoid detection and cause DoS state. This thesis is focused on process of generation and detection of such attacks.

1 Network communication

From the transport layer view of ISO/OSI (International Organization for Standardization – Open Systems Interconnection model) model, this thesis is strictly focused on TCP (Transmission Control Protocol) communication. It is reliable, connection-oriented and error-resistent protocol. The connection between the client and server is established in the beginning of the communication, then the application data can be sent. In the end of the communication client or server can close TCP connection. It is possible with two methods. Server must be in listening mode to receive client connection establishment request.

Connection is created using TCP 3-way handshake, in fig. 1.1.

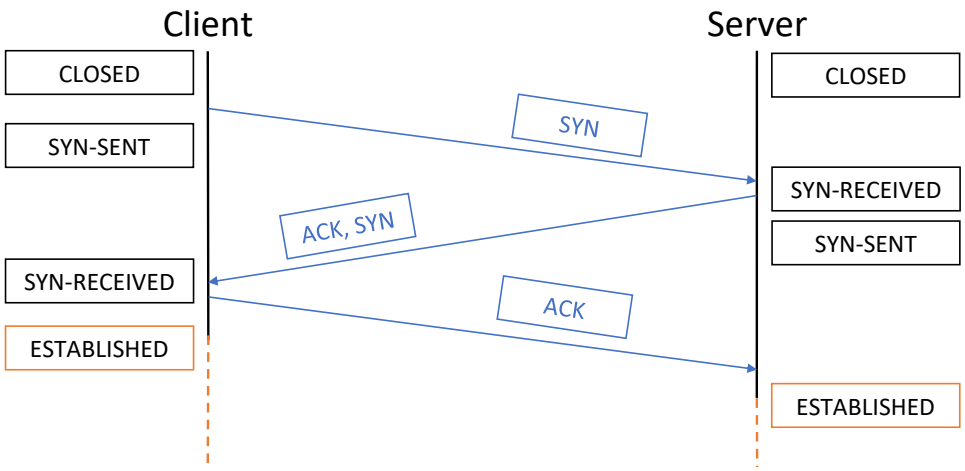


Fig. 1.1: Connection establishment with TCP 3-way handshake

Beginning state of TCP communication is `CLOSED`. The the client who wants to start communicating creates TCP packet with `SYN` flag set to establish new connection. It also means that the TCP socket is on the client side. New state is `SYN-SENT` awaiting acknowledgment and another synchronizing packet. If received the state transits to `SYN-RECEIVED` state. After the acknowledgment sent, client's socket transits to `ESTABLISHED` state. On the server side the socket `ESTABLISHED` state is created after successful acknowledgment receive. [5]

After successful connection establishment application data can be transferred. The sequence number identifies the order of the bytes sent from the client. These data can be received in different order and are reconstructed by the reliable mechanism. The sequence number is randomly chose in the beginning of the 3-way handshake (in `SYN` packet) and then incremented. The application data is sent using TCP `PSH` and TCP `ACK` flags and the raw data is an encoded payload. Application

data can be any kind of higher layer protocol (HTTP – Hypertext Transfer Protocol, HTTPS – Hypertext Transfer Protocol Secure, SMTP – Simple Mail Transfer Protocol, FTP – File Transfer Protocol, SSH – Secure Shell Protocol and others). Every communication exchange should be acknowledged by `ACK` flag. Example of HTTP communication as a TCP payload wtith sending payload data using TCP `PSH` and `ACK` is in fig. 1.2.

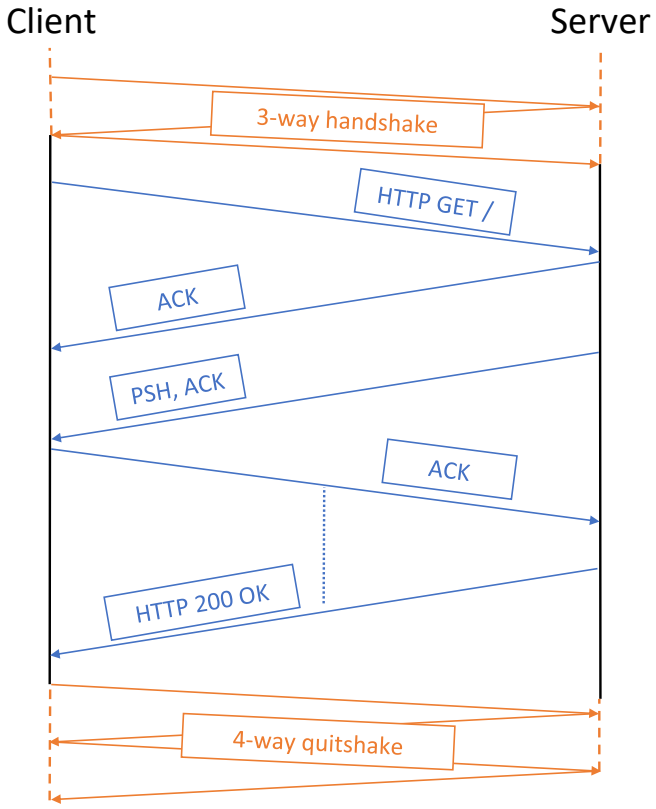


Fig. 1.2: HTTP client-server communication

As a TCP is en error-free protocol, correction mechanisms can be applied. Corrupted or lost packets are dropped and retransmitted signalized with `TCP_RETRANSMISSION` packet. A flow control mechanism otherwise limits the rate a sender can transfer data. Client can signalized how much data is capable to receive (with TCP `WINDOW_SIZE` value set). On the other hand server sends `TCP_WINDOW_FULL`. If duplicate packets are sent TCP packet with `TCP_DUPLICATE` flag is created.

Connection is terminated with TCP 4-way handshake in fig. 1.3, sometimes called 4-way quitshake. When a client wants to terminate the connection, sends TCP packet with `FIN` flag set. In this moment `ESTABLISHED` state transits to `FIN_WAIT_1` awaiting acknowledgment from the server. Then transits to `FIN_WAIT_2`

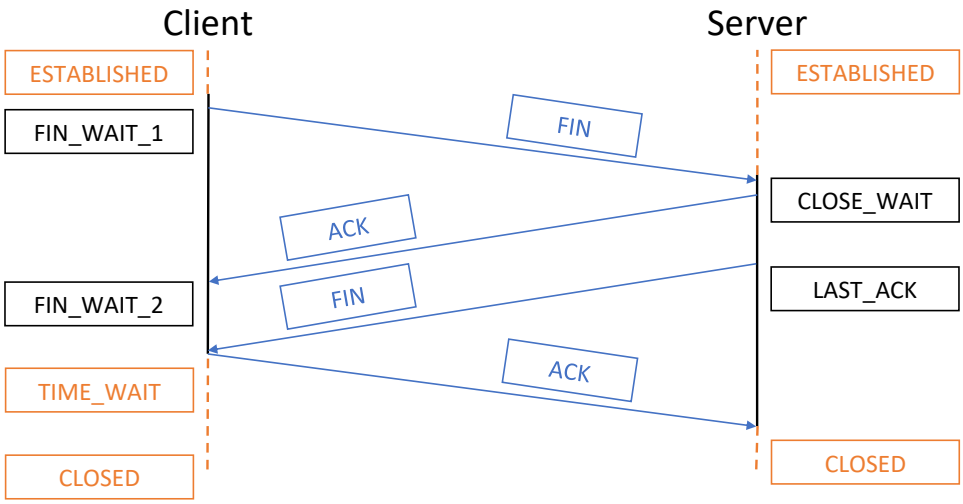


Fig. 1.3: TCP connection legitimate ending

state. Awaits server to send `FIN` flag set. When received, connection transited to `TIME_WAIT` state for certain period of time and then transits to `CLOSED` state.[1] Server transits to `CLOSED` state after the final acknowledgment received. [7]

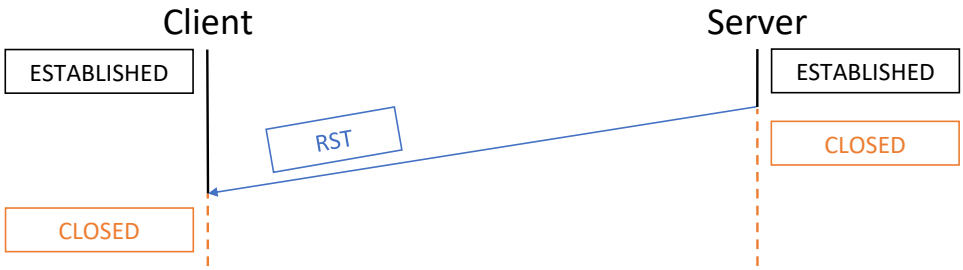


Fig. 1.4: TCP connection server-side ending

Another approach is a server side connection closing. Server can immediately close connection with TCP `RST` packet. There is no waiting state. Mostly it means that an error occurred. Server can send `RST` packets when multiple events happened. When Keep-Alive packet limit exceeds, server sends `RST` packet. Or when max number of connection exceeds, time of initial `SYN` packet expires, logic of upper layer application decides to immediately terminate the connection, Internet Control

[1] The duration of `TIME_WAIT` state is defined by the double of maximum segment lifetime (MSL). It is the maximum amount of time, that any packet can exists in the network before being discarded. Standard implementation values are 30 s, 1 min or 2 min. During the `TIME_WAIT` state it can sent acknowledgment packets if lost `FIN` is received. In this time period socket can't be reused. [6]

Management Protocol (ICMP) `DESTINATION UNREACHABLE` message is received, unexpected socket closure occurs or an error in TCP protocol happens then server decides to close the connection.[8]

1.1 Application layer communication

Application protocols are transmitted inside data fields of TCP protocol that are binary encoded. More information about the data are presented in the protocol of the higher level of ISO/OSI.

One of the examples of application protocols is HTTP protocol. It begun as an text oriented protocol in version HTTP/0.9. It allowed only ASCII (American Standard Code for Information Interchange) text payload. With development of the protocol, in version HTTP/1.0 it started to support transportation of pictures, sounds, video clips and others. [9]

Structure of HTTP protocol is as follow. Client has to open a TCP connection to send a valid HTTP request and waits for a response. This message contains of 3 parts:

- HTTP Request (Response):[2]
  - HTTP Request – contains 3 elements (`GET / HTTP/1.1`). First is HTTP method, that contains method the client wants to perform on the server.[3] Another part of request is a path of the resource the client wants to approach. It is called Request-URI (Uniform Resource Identifier) that is composed of resource that will apply to request. It can compose regular expression, absolute path (with domain or IP address) or relative path. And the last part is the version of the HTTP protocol. [10]
  - HTTP Response – is composed of 3 parts (`HTTP/1.1 200 OK`). First part is the version of HTTP protocol. Second is response status code and last part is reason phrase containing text information. HTTP repsonse does not contain line end.
- HTTP Headers – dictionary type (`Header: Value`) record containing additional information for the server (request headers) of the client (response headers). Each header is enclosed with end of line using `'\r\n'` and before the next part is sent, one new line is awaited using the `'\r\n'` as well.
- HTTP Data – contains the encoded data.

[2]HTTP request or response line is closed with `'\r\n'` meaning end of line.

[3]Example of HTTP methods are: `GET`, `POST`, `OPTIONS`, `PUT`, `DELETE`, `HEAD`, `CONNECT`, `PATCH`, and `TRACE`.

1.2 Communication measures and parameters

The crucial fact in communication measurement depends on approach which is taken in mind. Results of measurement will differ depending on the side where the measurement is realized. Communication measurement side can be divided to:

- Client (Attacker) side – Is the communication, where the attacker establishes the communication. The way from the attacker to server is forwarding way or sending way. The opposite is receiving or backwards way. While capturing network parameters, it can be possible to measure client side parameters mainly.
- Server side – Communication ways are inverted according to client side.
- Communication channel side – This is the measurement implemented during the transmission over the communication channel. It can be used by IDS or IPS systems.

It is crucial to choose one side and stick to that during whole communication process. From this point of view network packets are forming network flows. Network flow can be modeled as a directed graph $G = (V, E)$ with nodes (graph vertices) V and network paths (graph edges) E. Each node is network device that is serving the network traffic with beginning (start) s and ending nodes e. Each path between nodes has some properties p defining quality, capacity or latency of the flow. The network flow can be described as (G, p, s, t). [11]

1.2.1 Flow classification and parameters

This thesis is focused on the end to end communication, so the side of communication can be neglected as the parameters are almost the same. The network flow can be described with Flow ID. This should unambiguously identify the network flow. Basic Flow ID is composed of Source IP address, Destination IP address and Destination TCP port. Extended Flow ID can have extra information about protocol, start of the flow and duration of the flow timestamps. As the flow can contain multiple parallel connections, the information about source port is not important as it changes with new established connections. [4]

[4]Protocol number is inside IPv4 or IPv6 packet in the 'Protocol' field of IP header. These number are assigned by IANA (Internet Assigned Numbers Authority). For TCP protocol it is assigned number 6. [12]

$$
\begin{aligned}
FLOW\_ID &= IP\_SRC + IP\_DST + TCP\_DST \\
EXT\_FLOW\_ID &= IP\_SRC + IP\_DST + TCP\_DST + \\
&\quad + PROTO + CONNS + TIMESTAMP + \\
&\quad + FLOW\_DURATION
\end{aligned}
\tag{1.1}
$$

Flow parameters are listed in tab. A.1. They can be divided into three groups:

- Intra-flow parameters – are parameters describing network communication, single packets inside the flow.
- Inter-flow parameters – are parameters describing aggregated view on the packets into a flow.
- Common parameters – are parameters that are independent on the flow.

Common parameters can be used for both Intra-flow or Inter-flow description. They contain information available from TCP header (source IP address, destination IP address, destination port, protocol number, timestamp). Two important parameters are `FLAG` and `ATTACK_TYPE` describing type of event the packet or flow is part of. `FLAG` is a binary parameter where 0 means normal traffic and 1 stands for malicious or attack traffic. If `FLAG` is set to 1, future attack classification parameter take place. `ATTACK_TYPE` parameter classifies the certain type of the attack. It can contain following values representing futured SDAs:

- 1 – Slow Read DoS attack
- 2 – Slow Drop DoS attack
- 3 – Slow Next DoS attack
- 4 – Custom Slow DoS attack

Intra-flow parameters describe individual packets inside a certain flow. They are formed with data from TCP headers and from simplified parameters from Inter-flow categories. They don't have further knowledge about packet relations.

The network flow identifier can be used as an indexer to reduce the size of the captured packets. These packets can be sorted and classified according to the identifier into a flow. Parameters describing relation between packet are called Inter-flow parameters. They can be classified into following categories:

- General parameters – contain important parameters. They are used for flow description.
- Volume parameters – are volumetric parameters. They testify about some statistical properties and amount expression.
- Time parameters – are parameters describing time periods between important events.
- Feature parameters and TCP Analysis parameters –

- Application layer parameters – are parameters describing behavior of application layer protocol.

General parameters

General parameters are used for flow description. The most important parameter is `FLOW_ID` or `EXT_FLOW_ID`. Another important parameters are timestamps of flow duration and number of connections inside the flow.

Volume parameters

Volume parameters can be created according the following rule:
$Volume\_Parameter =< direction > \_ < type > \_ < stat > \_ < amount >$, where *direction* can be either `FWD` – forwarding way, or `RCV` – receiving way. According the fig. 1.5 direction is determined by the position of the measurement side. (For the measurement at server side directions will be swapped.)

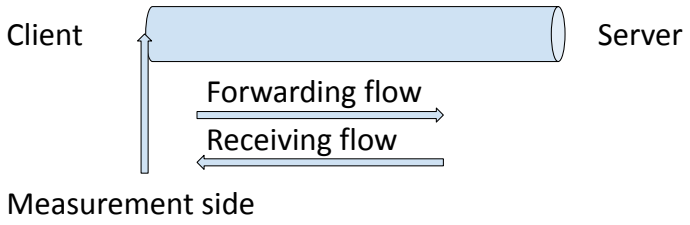


Fig. 1.5: Flow parameters direction

Next part of volume parameters is *type*. Type stands for the values that will be measured. It can be number of packets in the flow, packet throughput, size of packets, length of TCP headers, size of the initial window, value of TCP window size[5], TCP window scale factor[6], TCP maximum segment size[7], bulk rate in bytes and in packets[8], TCP payload size or packets that have at least $1B$ payload.

Next part of volume parameters is *stat* which represents statistical properties on aggregated packets in the flow. They have following values, in tab. 1.1.

[5]It is the size of receiver's buffer. It says how much data receiver is willing to get. [13]

[6]It is used to extend the maximum size of TCP window size. If the value of the TCP window size is larger than 64 kB it can be used.

[7]It specifies the largest amount of data that a device can receive in a single TCP segment.[14]

[8]Bulk rate mean the number of bytes or packets that are sent without explicit acknowledgment.

Name	Description	Calculation (x_i – single parameter)
TOT	Aggregated value of given parameters	$\sum_{i=1}^{n} x_i$
MAX	Maximal value of given parameters	$\max_{i=1}^{n}(x_i)$
MIN	Minimal value of given parameters	$\min_{i=1}^{n}(x_i)$
MEAN	Mean value of given parameters	$\frac{1}{n}\sum_{i=1}^{n} x_i$
STD	Standard deviation value of given parameters	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}$

Tab. 1.1: Statistic values of volume parameters

Last part of volume parameters is *amount* which describes count value of given type. For the amount of bytes (size), mark S is used. For the amount of packets, mark C is used and for the rate R is used.

Time parameters

Another crucial parameter of network communication is time. It has similar structure as volume parameters with exception of *amount*. Time is measured in s or ms. It contains information about IAT (Inter-arrival time) between two or all sent or received packets and information about the state of the flow.

State of the flow means duration the flow was in idle or active state, where active means that communication is happening. On the other side if the flow state is idle, it means that the connection is not closed, but no data is flowing. Flow state does not require direction as it is looked from broader perspective.

Flow contains multiple connections, some of the connections can be idle, closed or active. State parameters describes important bounds (maximum, minimum), total time for all connections inside the flow and mean and standard deviation for one connection.

Feature and TCP Analysis Parameters

TCP header may contain up to nine one-bit flags. [5] Each flag means something different. Each parameter represents number of certain flags inside the flow in appropriate direction. Following flags are measured in this thesis:

- `SYN` – is the first flag beginning the TCP communication.
- `ACK` – indicates that Acknowledgment field of TCP header is important. It is usually used in each TCP segment after the first one with `SYN`
- `FIN` – is the last flag from the host meaning end of communication. It should be acknowledge in 4-way quitshake.
- `PSH` – is used to push data, often from web server to client.

- `URG` – indicates that urgent pointer is set and has some meaning.
- `RST` – is used for connection reset or end.

Wireshark can follow each TCP opened connection (session) and provides additional information about that session. It can create two types of analysis. First type is TCP Flag (SEQ/ACK) analysis and second is TCP Reassembly analysis. TCP Flag analysis follows packets in each session, saves important information from the TCP header (time, sequence numbers, acknowledgment number, window size, flags). Then performs analysis and appropriate flags are added to the TCP packets' section `SEQ/ACK analysis`. [15]

Possible flags of TCP analysis: [16]

- `TCP ACKed unseen segment` – set if the packet acknowledges incoming data, that was not previously captured. Acknowledgment number is lower than current ackownledgement number in the session.
- `TCP Previous segment lost` – happens when a packet arrives with a sequence number greater than expected sequence number should be on give connection. It indicates that some packets did not arrive. It is followed by packet retransmission.
- `TCP Retransmission` – occurs when the sender rentransmits a packet after the expiration of the timeout for the acknowledgment. It is measured by RTO (Retransmission timeout). [17]
- `TCP Dup ACK <frame>#<ack number>` – set if the same ACK number has been seen and it is lower than the last byte of send data. If there is a gap in sequence number, receiver will generate that space with duplicate ACKs for each subsequent packet in that session, until the missing packets are successfully received. It contains the number of the frame where the duplicacy exists and order of duplicate occurrence.
- `TCP Out-Of-Order` – occurs when the packet is with a sequence number lower than the previously received inside one connection.
- `TCP KeepAlive` – is used to force receiver to send ACK packet. The sequence number is equal to the last byte of data in previously received packet.
- `TCP KeepAlive ACK` – ACK response to `TCP KeepAlive` packet.
- `TCP ZeroWindow` – occurs when the receiver is unable to process incoming data (due to receiver's buffer overload). Value of `Window` is close to zero.
- `TCP ZeroWindowProbe` – occurs when the sender is testing if the receiver's `ZeroWindow` condition is in place yet. Sender prolongs the timer for probes.
- `TCP ZeroWindowProbeAck` – is send as an acknowledgment to zero window probe packet.
- `TCP ZeroWindowViolation` – is used when the sender ignores the zero window condition and sends aditional data.

- `TCP WindowFull` – is used by sender with knowledge of the size of receiver buffer. If sender calculates that the payload data in the segment will completely fill the last known receiver buffer size on the host, it will flag data and stop sending. This causes delays in communication often resulting in `TCP ZeroWindow` response by the receiver.
- `TCP WindowUpdate` – is send by receiver when he process all data from his full receiver's buffer. It indicates more free space in receiver buffer. It is often preceded by `TCP ZeroWindow`.

Application layer parameters

Application layer parameters abstract from the TCP flow. It is special category layer above flow parameters. It contains all categories from Inter-flow parameters (volume parameters just as time parameters) that describe application layer behavior.

From the volume parameters it can contains count or size of HTTP requests, number of different HTTP requests, count or size of HTTP methods, number of HTTP headers in each request, etc.

Time parameters can specify certain behavior of SDAs. It is more described in sec. 2.1.1.

2 DoS Attacks – Specification and Classification

With the Internet evolution and more devices being online, new types of attacks have appeared. These attacks strictly depend on the goal and motivation of the attacker. Perpetrator uses attack vectors to exploit target system vulnerability to gain access to a device, personal data, company information or research. With these information he can modify, destroy, steal and sell or disable from using given assets. He can reach his goal with malware, viruses, social engineering methods or specific denial methods (manipulating of network packets, logical errors, programming flaws or resource handling). [18] [19]

The basic division of denial methods is into DoS attacks and DDoS (Distributed Denial-of-Service) attacks. (Sometimes DDoS attacks are born in mind as a subcategory of DoS attacks.) DDoS attacks overloads the victim with tons of requests using compromised devices. Usually attacker infects devices with malware, allowing him to be controlled remotely. We call such devices bots or zombies, that can form group of devices – botnet. During DDoS attack, adversary controls bots and directs all requests to single victim. [20]

DDoS attacks are being actively used to overload the victim's device by seizing all available connections that the service server can handle. Nowadays hacktivist groups (e.g. Anonymous), state actors or even normal people use or are part of botnet that is used for DDoS attack. [21]

Security professionals use DDoS attacks to cover-up subsequent attack vector. Main goal is to hide malicious traffic of initial access to victim's device, vulnerability exploitation, malicious code execution or privilege escalation techniques by volume centric DDoS.

On the other hand DoS attacks are usually managed from single attacker device to a single victim device implementing TCP/IP (Transmission Control Protocol/Internet Protocol) stack, such as personal computer, router, application server (web server, email server, logging server, DNS (Domain Name System) server, DHCP (Dynamic Host Configuration protocol) servers), providing any kind of service. It can also target multiple network nodes, or even whole network segment. These attacks can span from L2 to L7 of ISO/OSI model. The goal of DoS attack is to disable target resources, lower the QoE (Quality of Experience) or do not allow connection at all.

Historically first DoS attack was probably SYN flood attack against one of the oldest ISP (Internet Service Provider) in the world (Panix) in 1996, denying network services for several days. [22] One of the biggest DoS attacks was in February 2014

against cloud provider CloudFlare with maximum throughput of about 400 Gbps. The attacker used NTP (Network Time Protocol) vulnerability to create DDoS by NTP amplification.[1] Another DoS attack was conducted in January 2015 after terrorist attack on Charlie Hebdo where multiple French web sites were taken down by Islamic hacker groups affiliated with terrorist group ISIS. [23] As times progresses, the volume of DDoS attacks increases. In 2017, Google Cloud was targeted with attack with peak volume of 2.54 Tbps. In March 2018 GitHub was attacked by 1.35 Tbps DDoS. In February 2020, Amazon AWS was targeted with DDoS with peak of 2.3 Tbps. In July 2021, CloudFlare was hit by Mirai botnet with 17.2 million requests per second. [24] Lastly in early 2022, during Russian aggression against Ukraine, Russian government, banks and public services was hit by people around the world with HTTP based DDoS. Presented on static HTML website using JavaScript. Where the only need to create multiple asynchronous requests was to open given web page. [21]

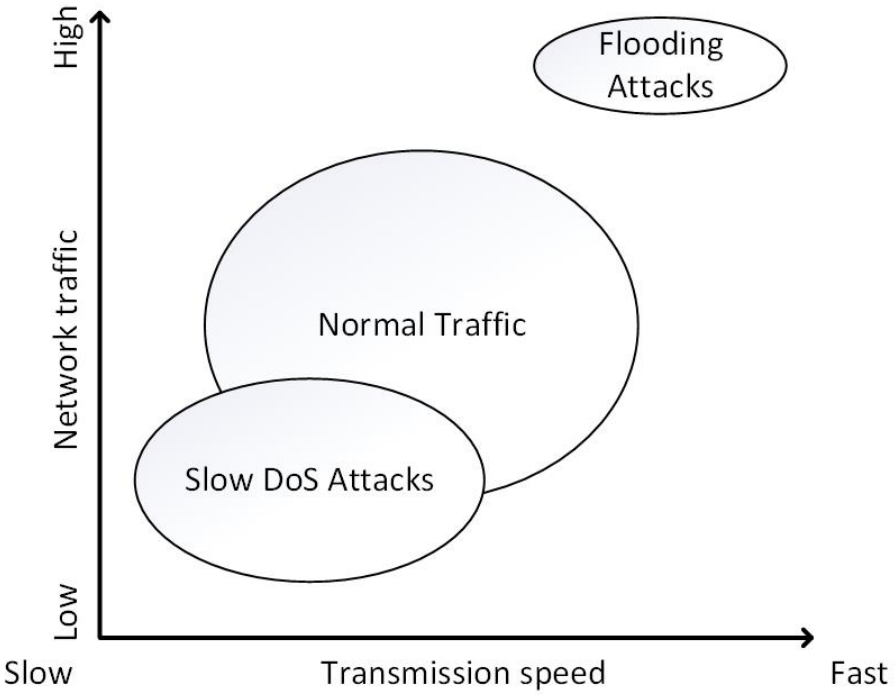


Fig. 2.1: Scheme of SDA traffic, normal traffic and Flooding attack traffic

[1] A few hosts create small malicious NTP requests, where NTP server responds with large NTP responses to single victim, overloads it and causes denial of service.

Basic specification of DoS attacks is at:

- Flood-based attacks – also called the first generation of DoS attacks or network based DoS attacks, aim to overwhelm victim with large amount of simple requests. Adversary usually generates multiple malicious requests which tend to exhaust victim's hardware and software resources and results in service unavailability. [1]

  This attacks can span to almost all layers of ISO/OSI model. Between L2 DoS attacks we can embody MAC (Medium Access Control) attacks, where attacker creates multiple ethernet frames with different MAC addresses forcing network switch MAC table to lose legitimate records.

  On L3 we can distinguish various flood attack types, e. g. ICMP (Internet Control Management Protocol) Flood attacks, where attacker forges Echo requests with size bigger than maximal value.

  Between attacks on L4 we count TCP SYN Attack, where attacker creates large number of incomplete requests, that result in half opened connections with victim and exhausting connection capacities for legitimate users.

  On L7 attacks are not as voluminous as on lower layers. They congest the victim with valid requests of application layer protocols. E. g. DHCP Starvation Attack, DNS Amplification Attack, VOIP flooding ...
- Exploit-based attacks – also called the second generation or new generation of DoS attacks, that use wrong implementation of transport or application protocol. This can be exploited by attacker creating modified, half-opened or changed connection creating uncontrollably opened connection. When this legitimate connections are send slowly with low amount, we can call this behavior as low-rate attack or SDAs. [25] As shown in fig. 2.1, we can see difference between normal traffic, SDA traffic and flooding traffic. Also the difference between normal and flooding (or distributed) attack traffic is enormous. But the distinction between normal traffic and SDA traffic is small. It results in hardness of detection of such a behavior.

  First comparison between Flooded-based and low-rate attacks was provided by Liu in 2012. [26] Then the first taxonomy of SDAs was created by Cambiaso in 2012 [27] and in 2013. [1]

2.1 Slow DoS Attacks

SDAs' name comes from the first and the most famous attack - *Slowloris*, named after asian monkey, known for slow sneaking movement. It was created by Robert "RSnake" Hansen in perl programming language. The behavior of the attack is to keep as many opened connections as possible to web server. It is achieved by creating incomplete legitimate HTTP (Hypertext Transfer Protocol) `GET` request followed by partial legitimate request with header `"X-a"` maintaining the connection opened. Target can't serve other requests from legitimate users resulting in successful DoS. The first use of this SDA was in 2009 during the Iranian presidential election. [28]

Other names for this category of attacks can be Low-rate DoS Attacks, Slow-rate DoS Attacks or Low-bandwidth DoS Attacks.

The purpose of SDAs is to cause unavailability of victim's device by creating small amount of asynchronous connections. Usually transported through TCP protocol. The goal of an attacker is to occupy all possible queues with malicious requests as long as possible. This can be reached by various techniques. Then all new incoming requests from legitimate users are discarded by the server. Once all queues are occupied, the attacker has to send "keep alive" continuing requests, that keep queues occupied.

The first issue that attacker must solve is to find the correct speed of requests sending. When they are send fast enough, it is easier to implement, but on the other side easier to detect by statistical detection of high-rate traffic. [29] Therefore attacker must find the best threshold value, in which he sends malicious requests.

The next issue is to find the correct server timeout settings. Every application server has some timeout settings of handling requests or connections. So for that attacker it is suitable to find timeout values as close as real ones.

Attacker can exhaust queues in two different ways shown in fig. 2.2. It can exhaust internal or external resources. Examples of exhausting the internal resources are: occupying CPU time, exhausting RAM, fill hard disk, network queues on the web server with unwanted data. This can cause *Delayed Responses* type of SDA. In other words, attacker forces victim to do pricey operations which will result in sending responses with delay. Attacks in this category are *Apache Range Headers*[2], *#DoS*[3] and *ReDoS*[4] attacks.

External resources are connected with victim's network. Malicious client application, where attacker can close and reopen connection whenever he wants - *Resource*

[2] *Apache Range Headers* – Attacker asks for specific overlapping byte range of large data from web server.

[3] *#DoS* – Attacker sends single HTTP POST with thousands of variables or keys to be found in hash table.

[4] *ReDoS* – Attacker creates *evil regex* that results in long time matching the expression.

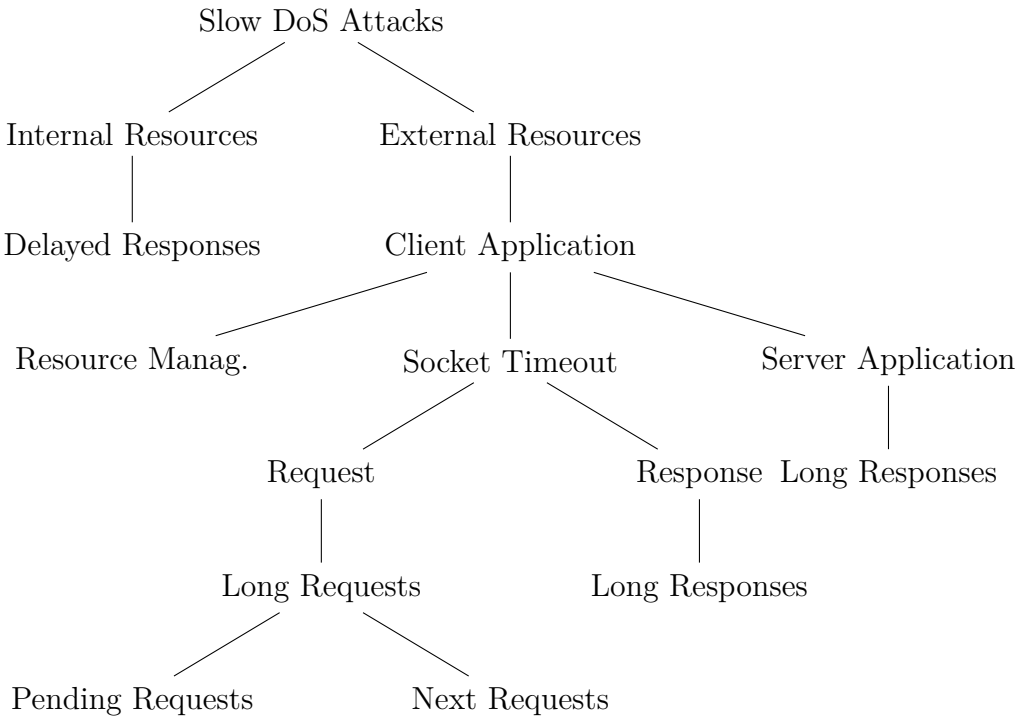


Fig. 2.2: SDAs classification according attacked resource [1]

Management SDA. Examples of the attacks in this category are *LoRDAS*[5] and *Slow Drop*[6] Then *Pending Requests*, *Next Requests* and *Long Responses* are categories challenging victim's application's socket timeout. At *Long Responses* attacker can affect also server application in tab. 2.1.

Pending Requests and *Next Requests* SDAs are types where attacker sends incomplete request to the target. These requests are filling the victim's receiving queues. They are prepared with incomplete protocol requests. And maintaining the attack with repetitively sending requests, exploiting server's timeout settings with limited number of opened connections.

If there are only repetitive initial requests - *Next Requests*. For both phases we have *Pending Requests* SDAs in tab. 2.1. Examples of *Pending Requests* are *Slowloris*[7] or *Slow HTTP Post*[8].

Last category is *Long Responses*, where complete valid requests are being sent

[5] *LoRDAS* – Attacker tries to predict the moment, when the resources will be freed up and reuse them again.

[6] *Slow Drop* – Attacker receives valid responses, but in given or random time period drops certain amount of them.

[7] *Slowloris* – Attacker creates valid HTTP GET accompanied with connection holding request.

[8] *Slow HTTP Post* – Attacker uses HTTP POST request to simulate sending data to page form with big content length parameter set during preparing phase. In the second phase attacker keeps sending random parameters.

Attack category	Attack Examples	TCP/IP	HTTP
Delayed Responses	*Apache Range Headers*	L4	`HEAD`
	#DoS	L4	`POST`
	ReDoS	L4	`POST`
Resource Management	*LoRDAS*	L3, L4	-
	Slow Drop	L3, L4	`GET`
Next Requests	*Slow Next*	L3, L4	`HEAD`
Pending Requests	*Slowloris*	L4	`GET`
	Slow HTTP Post	L4	`POST`
Long Responses	*Slow Read*	L3	-

Tab. 2.1: Categories of SDAs with examples

forcing victim to send responses in slow way. Attack example of this category is *Slow Read*[9].

Sometimes we can add a category of external (intermediary) devices such as routers, switches, firewalls, modems that can be also affected by the SDA. [1] [25] [28] [30]

2.1.1 Timeout Division of SDAs

Crucial parameter of SDAs is time. As all above categories relay on timeout settings we have to introduce time parameters for application layer. We can generalize timeouts as it does not depend on certain TCP/IP layer.

[9] *Slow Read* – Attacker sends legitimate HTTP requests with restricted size of receiving buffer – TCP `WINDOW SIZE` parameter set to small value and reads the responses in slow way.

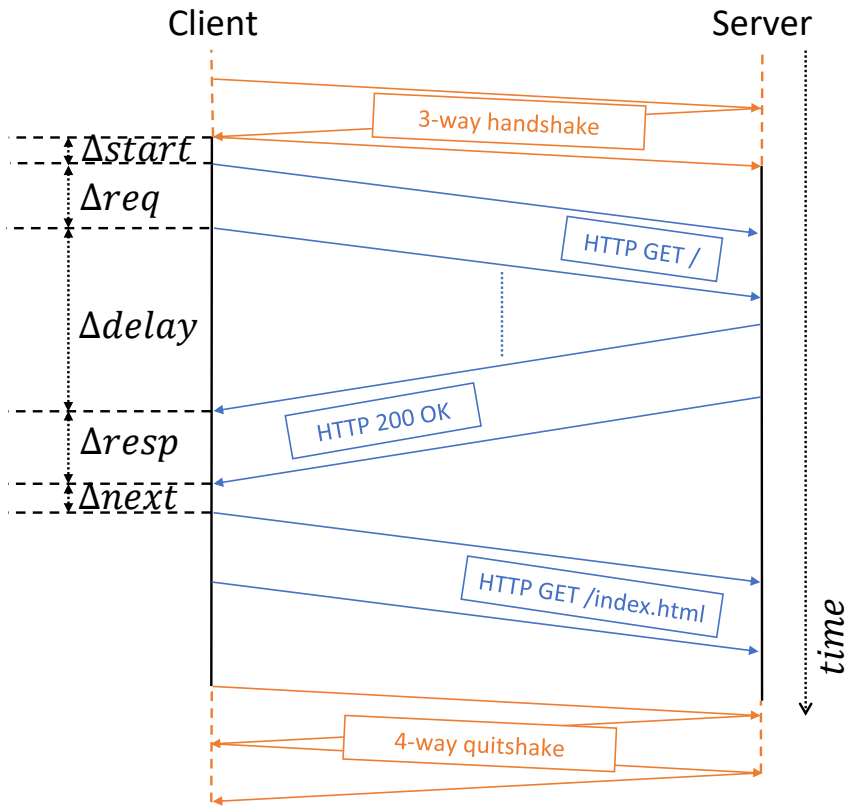


Fig. 2.3: TCP connection time parameters

If a client tries to connect to the server, creates a request. The request is received by the server. After period of time server sends response. This process is repeating till the connection is closed or maintained alive in keep alive state. Client creates two types of request: *Initial request* and *Pending request*. Initial request is send only once in the beginning of the connection. Pending request is send multiple times if set, till the maximal duration value expires or the connection is closed by the server. Following time parameters can be extracted at one endpoint e. g. client (in fig. 2.3), according the flow description in fig. 1.5:

- Δ_{start} – time interval begins in t_0 and ends with the begging of the request.
- Δ_{req} – stands for the time between the beginning and ending of the request.
- Δ_{delay} – identifies the time between receiving the requests and the response sending.
- Δ_{resp} – stands for the time between the beginning and ending of the response.
- Δ_{next} – identifies the time between the end of the response and the beginning of the next request.

These parameters are combined with *stat* parts and are listed in tab. A.1.

Some of the above mentioned attacks relay on the timeout settings. They can

be called *Timeout Exploiting SDAs* and mapped to previous categories in tab. 2.2. Pending Requests can be matched with Δ_{req} parameter. They exploits creation process of requests. Incomplete requests are being sent. Delayed Responses category can be matched with Δ_{delay} parameter. They exhausts internal resources of victim, delaying the response sending for the time specified period of time. The last important category is Long Responses that can be matched with Δ_{resp} parameter exploiting the response creation time interval. We have to extend previous categories for the Δ_{start} and Δ_{next} parameter. Δ_{start} parameter introduces Lazy Requests category and Δ_{next} parameter introduces Next Requests category. This to categories are interconnected. Only in the beginning of the attack is Δ_{start} valid parameter. Throughout the attack proceeds it changes into Δ_{next} category. Example SDA of the Next Requests is *Slow Next*[10]. [25] [31]

Timeout parameter	Attack types
Δ_{start}	**Lazy Requests**
Δ_{req}	Pending Requests
Δ_{delay}	Delayed Responses
Δ_{resp}	Long requests
Δ_{next}	**Next Requests**

Tab. 2.2: Timeout Exploiting Slow DoS Attacks

Another important structure is the network queue. Whenever server has to send a large file, it needs to convert it and then transport it through transport medium using L1 of TCP/IP model. First it is buffered and then prepared for sending in the sending queue. Sending queues has limited capacity. Whenever all request workers are busy and another requests stay in the buffered memory, all functional workers keeps sending the queued connection data on the TCP socket. If the size of the queue is the big enough, web server in some point stops following set timeouts resulting in connection being closed due to kernel limit of the space.[11] [33]

[10] *Slow Next* – Attacker keep sending HTTP Head request with connection keep alive set.

[11] *TCP Listen Backlog* is the value that determines the number of fully acknowledged connections to be accepted by the process. In other words it sets the size of the sending queue. For Apache 2.4 the default value is 511. It is possible to display backlogged connection with *ss*:
$ $ss - lti'(sport =: http)'$. [32]

3 Selected Slow DoS Attacks

This thesis is focused on attacks that are uneasy to detect, simulating legitimate users with slow internet connection. So for that, 3 attacks from categories in sec. 2.1 were chosen. First attack – *Slow Read* is from Long Responses category focused on filling victim's sending queues. Second attack – *Slow Drop* is from Resource Management category, where attacker is randomly dropping receiving responses causing server to send them again. The last attack – *Slow Next* is a member of Next Requests category, where attacker sends valid request with parameter to be alive as long as possible.

3.1 Slow Read Attack

This is the one of a few attacks directly targeting transport layer of TCP/IP model. Attacker creates legitimate TCP connection with victim. Attacker sets parameter of transport protocol size of window – `WINDOW_SIZE` to as low value as possible, e. g. 4, 8, 16, 32, 64 B. [1] It means that the attacker proposes the number of bytes of receiving queue that he can accept and process without dropping them. Usually it is done by setting this value in the initial 3-way handshake request in TCP `SYN`. It results in the victim's slow responses sending. If there is a large image on the site, it can take long period of time to be send. This basic attack can be followed by any type of application layer protocol.

The behavior of Slow Read attack is specified in fig. 3.1. After the server receives valid HTTP request, $\Delta resp$ describes the time period where the server tries to send response. Time period $\Delta buffer$ stands for the duration needed to exhaust resource on the server side. TCP Analysis flag in this time period are `TCP Window Full`, `TCP Zero Window` and `TCP KeepAlive`.

Server manifests that the receiver's capabilities to receive data are low. TCP Analysis flags that traffic with `TCP Window Full` and tries to send application data

[1] TCP `WINDOW_SIZE` is a field inside TCP header which can be set between 0 and 65535 B. It is an advertisement value of how much data receiver is willing to get. For the connections, where *bandwidth * delay* exceeds this value, receiver limits the amount of received data dropping them. To mitigate the problem new `WINDOW_SCALE` was introduced allowing larger window sizes to be set. It multiplies `WINDOW_SIZE` value to get real value. Maximum that can be set is limited to 14 bits. So maximum real `WINDOW_SIZE` value can be $2^{14} * 2^{16} = 2^{30} = 1073741824B = 1GB$.

On the other hand low values close to 0 B can preheat overwhelmed receiver, potential connection downsize or malicious behavior of SDA.

Server sends TCP segment confirming last successfully client-acknowledged data (TCP $WindowFull$) and responses with TCP $ZeroWindow$. Server waits in `FIN_WAIT1` state. If the client is available, sends TCP $WindowUpdate$ to server.[34]

with `PSH` set. Attacker responds with `TCP Zero Window` which means that he has not capabilities to receive another data.

Then the DoS state happens. Attacker sends TCP segments that using TCP Analysis are flagged as `TCP Retransmission` meaning that the server does not respond to the requests.

This attack was firstly discovered by Sergey Sheykan from Qualsys Labs and published in `slowhttptest` program available in Kali Linux distribution. [2]

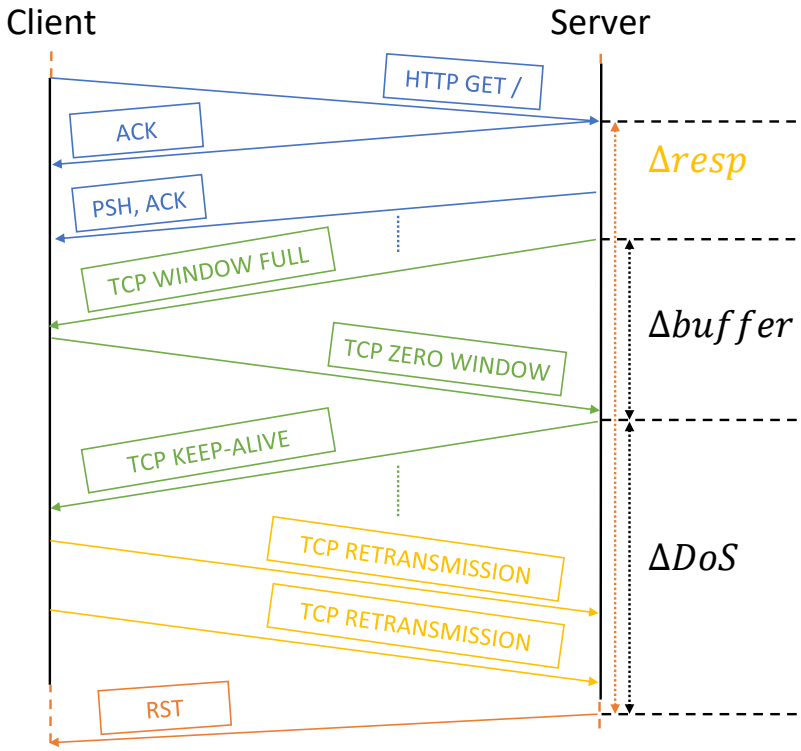


Fig. 3.1: Slow Read Attack behavior

3.1.1 Slow Read SDA parameters

It is needed to introduce some important parameters that represents the behavior of the attack and that can be useful for further detection:

- *connections* – number of opened TCP connections (connected sockets). It can be described using general parameter `FLOW_CONNS` from tab. A.1.
- *connection rate* – speed of opening new TCP connections $[cns/s]$. It can be described using Inter-flow parameter `FLOW_CONNS_R`.
- *duration* – time of receiving valid responses. It can be described using `FLOW_DUR` from tab. A.1.

- *window size* – the value of TCP `WINDOW_SIZE` parameter, read from initial handshake or from TCP Analysis. It is described by `FWD_TW_xxx_S` and `BWD_TW_xxx_S` parameters from tab. A.1.
- *data sent* – size and content of the data being sent. It is described using `FWD_PD_xxx_S` and `BWD_PD_xxx_S` parameters.

3.1.2 Slow Read testing

For the testing purposes the script in python3 was created. It establishes TCP connection with parameter of window size set to low value alongside with IP address of victim and port to connect the created socket.

```
def establish_connection(ip, port, window_size):
    """Creates 1 synchronous TCP conenction."""
    _get = f"GET /index.html HTTP/1.1\r\nHost: \
    {ip}:{port}\r\n\r\n".encode('utf-8')
    t_delay = 0.1
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, \
        window_size) # sets receiver buffer size
        s.connect((ip, port))
        s.send(_get)
        while True:
            data = s.recv(window_size)
            if not data:
                break
            time.sleep(t_delay) # t_delay parameter
    except socket.error as soerr:
        print("Socket closed!")
        sys.exit(-1)
    except KeyboardInterrupt:
        s.close()
        print("\nExiting...")
```

Listing 3.1: Slow Read attack connection setup

Other possible way to generate Slow Read Attack is using `slowhttptest` program. [2]

```
slowhttptest -X -c 2000 -r 200 -g -o slow_read -u \
http://<IP_web_server> -w 512 -y 1024 -z 32 -n 5 -k 3 -p 3
```

Listing 3.2: Example of the successful Slow Read Attack using `slowhttptest` [2]

Another way to create Slow Read attack is using `pyslodos.py` program proposed in this thesis using: [3]

```
(venv) python3 pyslowdos.py <IP_web_server> slow_read -r 24
```

Listing 3.3: Example of Slow Read Attack using `pyslowdos.py`

In attachment there is shell script `tcp_window_size_test.sh` that stands for TCP window size testing of the server's receiver buffer size. It generates `index.html` file of random data and given size using `generate_index.py` script. Then it generates Slow Read Attack for 1 connection using `tcp_window_size.py` script and tests it against web server for different sizes of receiver buffer. Then it generates `capture_$RCV_BUFF_SIZE.pcap` and `out.csv` file with following data: receiver buffer size, last time (connection duration), capture size (size of the `.pcap` file) and index file size. The last step is that it creates graph from `.csv` data available in fig. D.1.

[2] Parameters of the `slowhttptest` settings are following:

- *-X* – Slow Read mode of the attack,
- *-c <int>* – maximal number of created TCP connections to the web server,
- *-r <int>* – specifies the connection rate,
- *-g* – specifies the output of the program to file,
- *-o <string>* – output file path,
- *-l <int>* – attack duration,
- *-u <url>* – victim URL (Uniform Resource Locator) address,
- *-w <int>* – lower bound of the TCP window size value,
- *-y <int>* – upper bound of the TCP window size value,
- *-z <int>* – specifies number of bytes being read from the receiver buffer,
- *-n <int>* – specifies interval between reading operations from the receiver buffer,
- *-k <int>* – number of requests per socket,
- *-p <int>* – time interval at which the server is declared unavailable.

[3] Optional parameter of `pyslowdos.py` in Slow Read mode is:

- *-r <int>* – value of TCP Window Size.

3.2 Slow Drop Attack

This is the type of SDA, where attacker simulates unreliable connection. It can mask old wireless connection or dial up connection. Attacker sends legitimate requests to the server. Server processes the requests and sends the response. Incoming responses can be dropped during the transmission process or the attacker drops them periodically or randomly. Also the ratio between acknowledge received responses and dropped ones can be set, which hardens subsequent detection. Attacker can make the process of detection even harder setting different requests with different user agents specified in HTTP headers. [30]

The way response is resend relays on TCP implementation. Particularly on the mechanism of the congestion and avoidance control. Different implementation of TCP protocol results in different behavior.[4] [35]

Packet dropping approach should simulate unreliable client environment. Attacker should be able to selectively drop TCP segments with `PSH` and `ACK` flags set before the client interprets them. They can be dropped in transmission (using network tap) or on the receiving host (before interpretation of client software).

One of the methods of dropping packets on the receiving host is using `iptables`. Iptables is used as a stateless packet filter IPS that runs inside linux kernel. It contains several chains of operation. Each chain specific certain rules that specify what to do with captured packet. Iptables contains following tables: [36]

- `filter` – stands for default table that contains pre-built chains: `INPUT` (packets destined for local socket), `OUTPUT` (locally generated packets) and `FORWARD` (packets routed through device).
- `nat` – used for network address translation. It contains following built-ins `PREROUTING`, `OUTPUT` and `POSTROUTING`.
- `mangle` – used for specific packet alteration.
- `raw` – used for creating exemptions from connection tracking.

[4] Two possible TCP implementations are:

- *TCP Tahoe* – is a implementation which suggests slow start mechanism. After dropped packets, server sets the congestion window to 1 and the for each ACK received it increases the CWD (Current Window) by 1. It increases the CWD value exponentially till we loose the packet that is the sign of the congestion. This mechanism is repeated. For congestion avoidance it saves the half of the current CWD as a threshold value. After reaching that value it stops increasing exponentially and start increasing slowly.
- *TCP Reno* – it stands on previous implementation some intelligence and the basic pipeline is not emptied every time the congestion occurs. *TCP Vegas* – emphasizes the packet delay, rather than packet loss. It detects the congestion in the early stage as RTT (Round Trip Time) starts increasing. It heavily depends on the accurate calculation of the RTT.

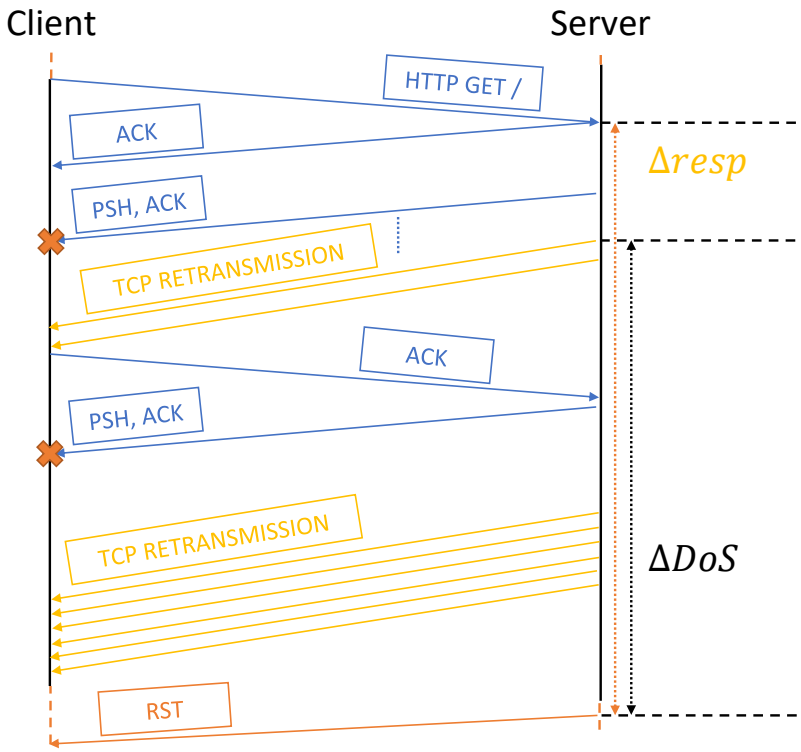


Fig. 3.2: Slow Drop Attack behavior

Slow Drop Attack setup with iptables is composed of two steps:

1. Accept incoming traffic creating TCP socket. Following example allows TCP packet with specific flags set to match an `ACCEPT` rule. The first parameter sets the flags to be examined inside the packet. The second parameter stands for the flags that must be set to match.

```
iptables -A INPUT -p tcp --tcp-flags SYN,ACK SYN,ACK -j ACCEPT
```

Listing 3.4: Iptables accept incoming connection initialization traffic

2. Randomly drop incoming traffic with set rate. It uses `statistic` module that provides to match random with probability (drop rate) or n-th packets. Matched packets are subsequently dropped.

```
iptables -A INPUT -p tcp --sport 80 -m statistic --mode random \
--probability 0.8 -j DROP
```

Listing 3.5: Iptables drop incoming packets with drop rate

If error or end of the attack occurs, Slow Drop attack needs to handle iptables cleaning and connection and thread closures. Iptables can be cleaned using: [37]

1. Set accept policies for all available chains.

```
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
```

Listing 3.6: Iptables accept all policies

2. Delete all rules.

```
iptables -F
```

Listing 3.7: Iptables delete all rules

Slow Drop Attack was published by Dr. Enrico Cambiaso and others in 2019.

3.2.1 Slow Drop SDA parameters and testing

Slow Drop SDA proposes some distinctive parameters of the attack that can be useful in further detection:

- *drop rate* – number of dropped packets. It can be measured with bigger occurrence of TCP segments with `RCV_RTRSMN_F` analysis flag.
- *connections* – number of opened TCP connections (connected sockets). It can be described using general parameter `FLOW_CONNS` from tab. A.1.
- *connection rate* – speed of opening new TCP connections $[cns/s]$. It can be described using Inter-flow parameter `FLOW_CONNS_R`.
- *duration* – time of receiving valid responses. It can be described using `FLOW_DUR` from tab. A.1.
- *data sent* – size and content of the data being sent. It is described using `FWD_PD_xxx_S` and `BWD_PD_xxx_S` parameters.

For testing purposes we can use python script `slowdrop_mazanek.py` in lst. 3.8, that generates the Slow Drop Attack. [5] [38]

```
(venv) python3 slowdrop_mazanek.py -url http://<IP_web_server> -d 0.85
```

Listing 3.8: Example of the Slow Drop Attack

Another way to create Slow Drop Attack is using `pyslowdos.py` program proposed in this thesis using: [6]

```
(venv) python3 pyslowdos.py <IP_web_server> slow_drop -D 0.65
```

Listing 3.9: Example of Slow Drop Attack using `pyslowdos.py`

3.3 Slow Next Attack

Slow Next is the type of SDA independent on application layer. Attacker establishes legitimate TCP connection with victim and wait for the response. Server process the requests and sends the response. After the response being sent, the connection on the server side is in the `FIN_WAIT1` state, waiting for certain period of time for the other requests in fig. 3.3. This attack exploits Δ_{next} parameter. After the time interval (defined in *KeepAliveTimeout* timer) expires, the web server closes the opened socket. If the attacker knows the setting of the web server, it can exploit the behavior sending the valid maintaining requests before the timeout expires. [25]

Slow Next Attack was published by Dr. Enrico Cambiaso and other in 2015.

Any suggested implementation of the Slow Next SDA is not working with Apache 2.4.49 neither with Apache 2.4.29. This thesis propose own mode of `pyslowdos.py` attack script generator to create Slow Next DoS Attack. [7]

[5] Arguments of `slowdrop_mazanek.py` settings are following:
- *-url <url>* – victim URL address,
- *-t <int>* – number of threads generating HTTP GET requests,
- *-tgs <int>* – random delay between new thread is being created with upper bound set,
- *-gn <int>* – set the waiting interval if the previous responses were successfully delivered,
- *-d <float>* – drop rate.

[6] Optional parameter of `pyslowdos.py` in Slow Drop mode is:
- *-D <float>* – sets the drop rate.

[7] Optional parameter of `pyslowdos.py` in Slow Next mode is:
- *-k <float>* – sets the time interval between two consecutive requests.

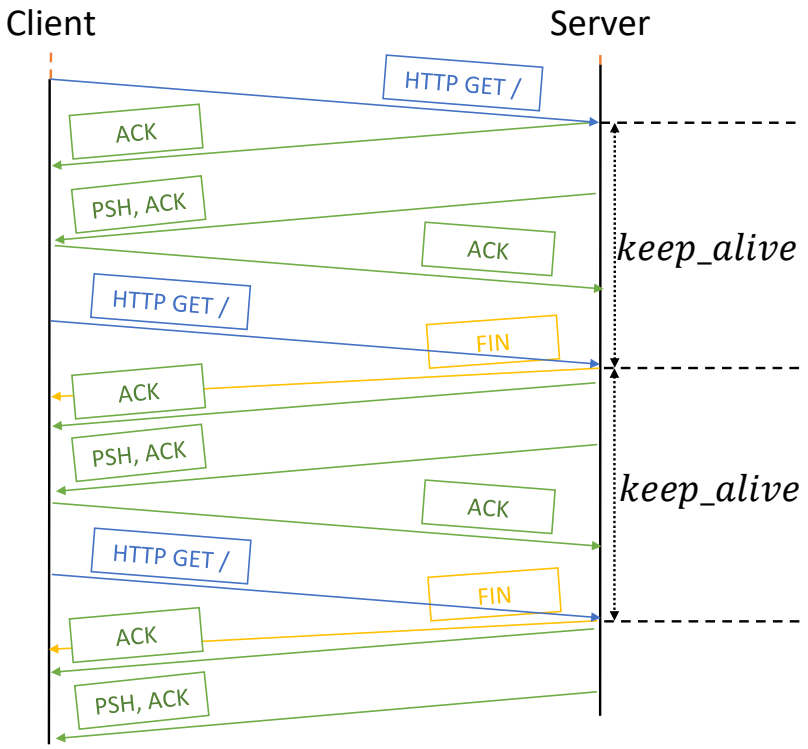


Fig. 3.3: Slow Next Attack behavior

To reach a DoS attacker has to open a lot of TCP connections as the proposed attack does not have implicit vulnerability. Every HTTP request occupies server receiving buffer resources as it has to process the request. If client does not close the connection, it waits for server closure. Client wants to send new connection 3-way handshake before or right after the server closes the current connection.

For the opening connection before the previous one was terminated requires more difficult mechanism. One possibility is to open new thread for a single connection before *KeepAliveTimeout* expires. Other one can be implemented as is in fig. 3.4 and way that the client sets socket to `TIME_WAIT` state using `shutdown(socket.SHUT_RD)` function. It says that client doesn't want to read the response and one-way closes socket for incoming data. Before that attacker waits for the `keep_alive` period similar or slightly smaller than *KeepAliveTimeout* or Δ_{next} parameter on the server side. Server keeps sending data till client hardly closes connection with `RST` flag. [8]

Another implementation timeout is $\Delta_{next_{wait}}$ timeout which stands for the period between `RST` flag was set and new connection being established. It should be set to as low values as possible to simulate successful attack.

[8]Only issue is that if the `keep_alive` timeout is larger it will cause the receiving buffer full. Server will send `TCP Window Full`.

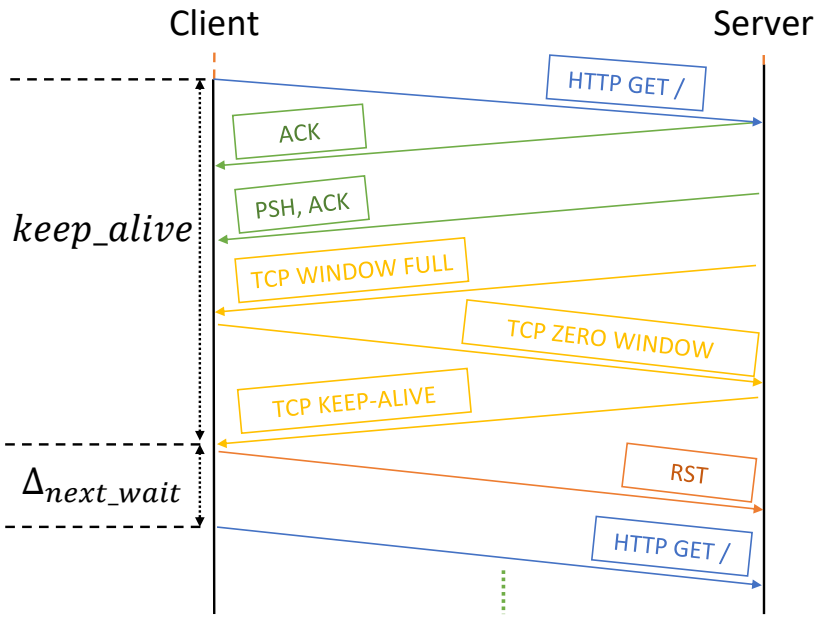


Fig. 3.4: Slow Next Attack implementation behavior

3.3.1 Slow Next SDA parameters and testing

Parameters of the Slow Next SDA:

- *connections* – number of opened TCP connections (connected sockets). It can be described using general parameter `FLOW_CONNS` from tab. A.1.
- *connection rate* – speed of opening new TCP connections $[cns/s]$. It can be described using Inter-flow parameter `FLOW_CONNS_R`.
- *duration* – time of receiving valid responses. It can be described using `FLOW_DUR` from tab. A.1.
- *data sent* – size and content of the data being sent. It is described using `FWD_PD_xxx_S` and `BWD_PD_xxx_S` parameters.
- *keepalivetimeout* – upper bound of the time interval between two consecutive requests. It can be described using `D_NEXT_xxx` parameters.

Example of Slow Next Attack can be tested using `pyslowdos.py` script.

```
(venv) python3 pyslowdos.py <IP_web_server> slow_next -k 4.5
```

Listing 3.10: Example of Slow Next Attack using `pyslowdos.py`

4 Slow DoS Attack Generator

This part introduces `python3` script providing arguments to create presented SDAs. It provides subcommand to create custom SDA. As proposed attacks use HTTP as an application protocol, the implementation uses HTTP as well. Application protocol is a text payload inserted into TCP payload. So it is easy to replace HTTP protocol with any other application layer protocol with complying syntax.

4.1 TCP connection with HTTP payload modeling using available `python3` modules

As an payload of TCP protocol can be any application protocol, represented as an encoded string. In `python3` there are several options how to create HTTP request-response application. For example using `requests` module, that allows very simple HTTP/1.1 request and response creation. Example in fig. 4.1.[39]

```
import requests

r = requests.get("http://<Target-IP>:<port>")
```

Listing 4.1: HTTP request-response with requests

Packet capture of such a communication is in fig. B.1, where there is 3-way handshake, `PSH` packet with HTTP request and fragmented responses that are subsequently reassembled to response and connection is closed with 4-way quitshake.

Another approach is with using `urllib3` module, that is simple HTTP client providing thread safeness, provides function for encoding, decoding, encryption and decryption. Also provides proxy support. Example of HTTP request and response in fig. 4.2. [40]

```
import urllib3

http = urllib3.PoolManager()
r = http.request('GET', 'http://<Target-IP>:<port>')
```

Listing 4.2: HTTP request-response with urllib3

The last approach is with using `socket` module, that is low-level networking interface providing access to the BSD (Berkeley Software Distribution) socket interface. It uses standard system call and library interface of C socket implementa-

tion with object oriented approach. Example of HTTP request and response is in fig. 4.3.[41]

```
import socket

host = '<Target-IP>'
port = 80
request = f"GET / HTTP/1.1\r\nHost:{host}\r\n\r\n""
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.send(request.encode())
#s.shutdown(socket.SHUT_WR)
while True:
    response = s.recv(4096)
    if not response:
        break
s.close()
```

Listing 4.3: HTTP request-response with socket

TCP socket object is created using `socket()` function. It requires following arguments: socket address family and socket type.

Socket address family depends on the operating system and and required build options. It determines structure of the address. Examples of socket address families are following:

- `AF_INET` – provides inter-process communication between different processes using tuple $(host, port)$, where *host* is IPv4 address or hostname of the target.
- `AF_INET6` – provides inter-process communication using IPv6 address.
- `AF_UNIX` – provides inter-process communication, where the address is represented by path string that is bounded to a file system entry.

Socket type are used to describe inter-process communication behavior. The most commonly used are stream and datagram types. Examples of socket types:

- `SOCK_STREAM` – uses connection oriented protocol, where data are reliably delivered. This socket type uses TCP protocol for data transmission.
- `SOCK_DGRAM` – uses best-effort, connectionless data delivery. This socket type uses UDP protocol for data transmission.
- `SOCK_RAW` – provides access to network layer protocol. It support socket abstraction. It can be used for custom transport protocol definition or for routing protocol implementation.

Subsequently TCP socket is connected using `connect()` function to the target and binary or encoded string data are send using `send()` function. [41]

4.1.1 TCP connection and attack closures

TCP socket can be closed in two ways that are summarized in tab. 4.1 and in fig. 4.1. From the sender side or from the client side. Client side closure is done with `close()` sokcet function. It is shown by TCP segment with `RST` flag immediately send. [41] Another way of client-side closure is using `kill()` thread function that will kill the socket handling thread. It will close opened connection as well.

Another socket closing function is `shutdown()` that can close one or both halves of the connection. It is specified by `how` argument which can contain following values:

- `SHUT_RD` – no data are received.
- `SHUT_WR` – no data are send.
- `SHUT_RDWR` – no more data are send or received.

If client decides to stop sending data, he creates TCP segment with `FIN` flag for one way connection closure. Once that happens, all further operations will fail.

Server can close the connection in two ways as well. First way is that server decides to close the connection due to internal reason. It can be connection timeout interval reached, server side issue, IPS prevents the system from intrusion or anti-malware protection.

Another way is that server closes the connection standard way after all data has been sent and no other requests to respond.

Side of closure	TCP flag	Reason of closure
Client	`RST`	connection
Client	`FIN`	standard
Server	`RST`	timeout
Server	`FIN`	standard

Tab. 4.1: Types of connection closure

Server side timeout closure

Next challenge to be solved is when server closes TCP connection in hard way. HTTP server most often closes connection hard way when connection timeout has been reached. If the socket exception handler does not catch server side closure, it needs to be handled. One possibility is to kill the program manually. It will also close one side opened connection. Another approach is to set the maximum attack duration for each thread.

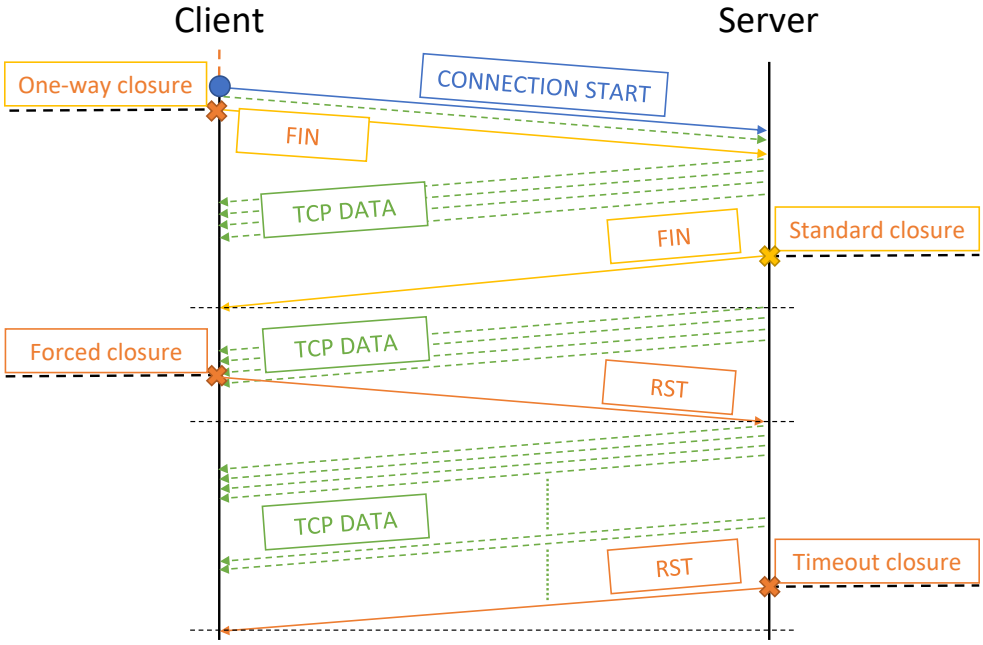


Fig. 4.1: TCP connection closures according the communicating sides

4.2 Multiple connections handling

If it is required to use more than one connection created and active, it needs to think about multiple connection handling methods. For attack execution with one connection the synchronous approach is sufficient, as in fig. 4.2.

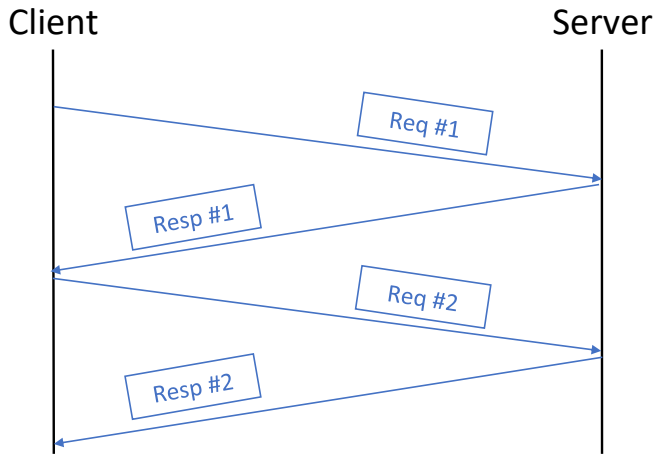


Fig. 4.2: Synchronous communication

But if it is required to connect multiple connections, the execution can wait for each and every connection to be successfully terminated. In other words one connection blocks the application execution. Another approach must be chosen.

As the application is meant to work as a single process on a single processor core, there is no need for parallel execution. Also multiprocessing approach, where the application is divided into multiple separated processes, can be eliminated.

4.2.1 Concurrent execution

The next possibility of reaching concurrency is using specific framework with single thread (sometimes called worker thread) or multiple threads called multithreading. [42] Principle of concurrency is that two or more tasks are executed on single or multiple threads taking advantage of CPU time-slicing feature. In the words of communication, client can create multiple sessions, create and connect multiple sockets independent on each other. Example of concurrent communication is in fig. 4.3.

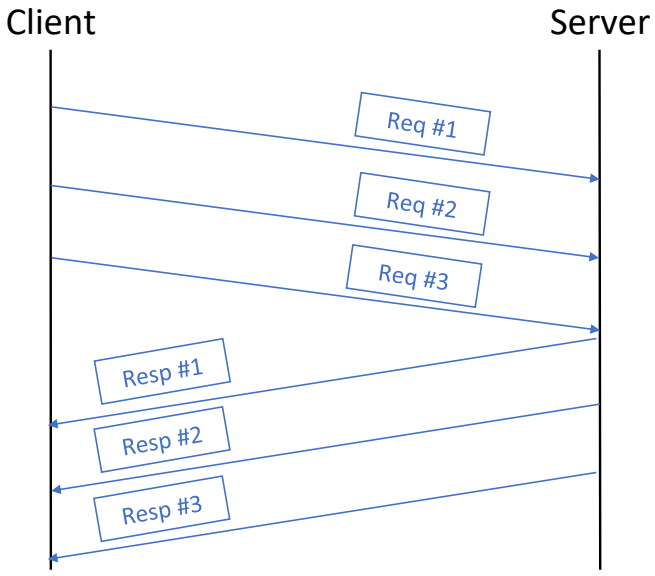


Fig. 4.3: Concurrent communication

Asynchronous modules

On of the methods of concurrent execution is `asyncio` python framework. It works in seemingly concurrent manner. It allows to run operations that are normally blocked by execution on single thread. Crucial concept is `async` and `await`. Code written wiht `async` before definition of function is called by `asyncio` in asynchronous way.

Async code can run inside event loop and this code (also called coroutine functions) has to use `await` that is waiting to corouting object to be executed. More in fig. 4.4. Advantage of asynchronous approach is that it saves the memory space in contrary with threads. [43]

```
import asyncio

async def func(args):
    print("Starting task")
    await asyncio.sleep(1)
    print("Finishing task")

asyncio.run(func(args))
```

Listing 4.4: Python asynchronous example

Multithreading

Thread is a sequence of instructions running in the context of one process. To execute multiple threads only one python process is enough. Disadvantage of such execution is that the process memory is shared between the threads. So CPU-bound tasks can't be effectively solved in python using multiple threads.[1] But for I/O bound operations, such as network communication using sockets, is very useful.[45]

Python's default module used for work with threads is `threading`. It is a high-level interface on top of `_thread` low level interface. It provides additional functions of the thread governance. [46] Thread can be created by making and instance of `Thread` class and started, in fig. 4.5. Another important setting of the thread is to decide whether it is a `daemon` thread or not. Daemon thread will shut down after the main program finishes. While the thread is not a daemon thread, it waits to be closed by execution or manually killed. Waiting for thread closing is with function `join()`. Possible outcomes: [47]

- `daemon=True` – If there is no thread ending, with main program finishing, daemon threads finishes (even before execution). Elsewhere main program waits for children's thread finish.

[1] The limitation is based on python's GIL (Global Interpreter Lock). GIL secures that threads of one process will not be executed at same time. It creates mutex that prevents multiple threads from executing bytecodes at once. At any time only one thread can lock execution for specific object. Solution is use multiple cores and one of multiprocessing modules.[44]

- `daemon=False` – If there is not thread ending, main program is separated with daemon thread, that can finish after the main program ends. On the other side it is the same as first example.

Running thread can be killed. If many threads are needed, it is possible to put them to shared memory structure and run them from that. Another approach is to use `ThreadPoolExecutor` that can create multiple threads and starts and finishes the threads autonomously.

```
import threading
import time

thread_count = 5
threads = []

def task():
    print("Starting task")
    time.sleep(1)
    print("Finishing task")

for _ in range(thread_count):
    # create threads
    thread = threading.Thread(target=task, daemon=True, args=())
    threads.append(thread)
    # start threads
    thread.start()

# execute threads - wait to complete
for x in threads:
    x.join()
```

Listing 4.5: Python threading example

4.3 Python SDA generator `pyslowdos.py`

Slow DoS Attack generator is created with `python3` programming language using object-oriented approach. It contains help page with required and optional arguments and with available operational modes. Alongside the generator script there is a `requirements.txt` file with required modules, `README.md` with installation hints, generator usage and attack examples.

This thesis is using following modules:

- `socket` – module for low-level networking interface. Provides access to BSD socket interface.[41]
- `threading` – module that allows code concurrency. It is appropriate module for running multiple I/O (Input-output)-bound tasks simultaneously. [48]
- `time` – module that provides various time-related functions. [49]
- `logging` – module that provides classes and function for event or application logging. It provides functions to create custom syslog-like logging. [50]
- `argparse` – module that provides to create user-friendly command-interface, defines required arguments and parses them. It automatically generates help and usage messages. [51]
- `random` – module that provides way to generate pseudo-random numbers for various distributions. [52]
- `os` – module that provides access to operating system functions. [53]

4.3.1 Generator arguments and structure

Slow DoS Attack generator contains positional and optional arguments enclosed in sec. C.2. There is only one required argument IP or URL address of the target. Generator script requires operational mode of the attack to be set. It is used for the SDA type selection. Other arguments are optional. They contain port number (default is 80), number of opened connections during the attack[2], maximum attack duration[3], maximum single connection duration, delay between threads creation and logging level with logging file output possibility.

Closing option arguments are `close` and `rec`. If `close` is set, it stands for client side closure if no valid data are being send. If `rec` is set, it means closed connection (server or client side closure) inside valid duration period is reconnected (default value is False).[4]

Generator modes are following:

- `slow_read` – stands for Slow Read Attack. It has only one optional argument that is TCP window size value.
- `slow_drop` – stands for Slow Drop Attack. It has optional argument setting drop rate.

[2]One connection is equal to one thread.

[3]Duration argument stands for a lower limit of the real attack duration if the connection reconnect is set. If not reconnect set attack can end before the duration period. If the attacker receiving buffer is blocked with incoming requests, it waits for the server side closure, that can happen after the attack duration period.

[4]If both closing arguments are set, generator will close the connection immediately after no more data are received. But second argument will cause reopening of such closed connection.

- `slow_next` – stands for Slow Next Attack. It has optional argument setting keep alive timeout.
- `custom` – stands for Custom Slow DoS Attack. It can contain optional arguments from previous modes plus other optional arguments. They can set sender and receiver buffer size with chunk sending or receiving interval. Then it can has set delta time arguments. For application layer, custom mode provides initial HTTP request with headers and data. It can contain repeting (pending) HTTP request with headers and data as well.

In Custom Slow DoS Attack mode client can send pending requests till the server closes the connection (if not connection reconnect set). If client keeps sending pending requests and has no incoming data, server keeps connection opened in `FIN_WAIT2` state. If no Δ_{next} argument set client will keep pending request in quite overwhelming way.

Custom mode requests and responses can be send or received in following manner:

- Request
  - Initial
    - Slow Sending – Is chosen if `-s <int>` parameter is set. It stands for the size of sender buffer. It can be accompanied by `-st <float>` parameter standing for interval between sending individual chunks.
    - Normal Sending – Standard initial request sending.
  - Pending
    - Slow Sending – Is chosen if `-s <int>` parameter is set. It stands for the size of sender buffer. It can be accompanied by `-st <float>` parameter standing for interval between sending individual chunks.
    - Normal Sending – Standard pending request sending. If pending request is closed and `-rec` is set, it will keep sending pending requests.
- Response
  - Initial
    - Slow Reading – Is chosen if `-r <int>` parameter is set. It stands for the size of receiver buffer. It can be accompanied by `-rt <float>` parameter standing for interval between receiving individual chunks.
    - Normal Reading – Standard response for initial request.
  - Pending
    - Slow Reading – Is chosen if `-r <int>` parameter is set. It stands for the size of receiver buffer. It can be accompanied by `-rt <float>` parameter standing for interval between receiving individual chunks.
    - Normal Reading – Standard response for pending request. If pending request is closed and `-rec` is set, it will keep sending pending requests.

If keep alive `-k <float>` is set. It means that the connection waits for the given period and then client one-side closes connection for incoming data and resends request.

Logging

Slow DoS Attack generator provides two types of logging events using messages. One is standard output logging and another is file logging specified with the name using parameter `-log <string>`. This application provides simplified version of `syslog` protocol. Each log message contains following structure:

```
[timestamp] [thread] [module] [severity] [message]
```

Listing 4.6: Logging structure

This application does not provide all logging levels. It contains only following severity levels set with the occurrence of the program parameter `-l`: [54]

- *no occurrence of -l* – `Level 0` (`ERROR`) – Error message about program or component malfunction. Normal functionality of the program is affected. It is equal to Level 3 syslog severity.
- `-l` – `Level 1` (`WARNING`) – Warning message about possible program or component issue or incorrect user input that can cause application failure. Equivalent to level 4 syslog severity.
- `-ll` – `Level 2` (`INFO`) – Informational message about standard program functioning. It is used for important events. It is equivalent of syslog level 5 and 6 severity level.
- `-lll` – `Level 3` (`DEBUG`) – Debugging level shows extended information about program execution, inputs, outputs. It is equivalent to syslog severity level 7.

5 Testing environment and data capture

Testing environment in fig. 5.1 composes of the attacker device, victim's server and interconnecting router. For the attacker purpose, Ubuntu 20.04.3 LTS operating system was chosen running on the PC with Intel i7 4720 HQ CPU and 16 GB of RAM. Attacker has IP address 10.0.0.200/24.

Server Dell PowerEdge T30 with 16 GB RAM and Intel Xeon E3-1225 CPU with Ubuntu 20.04.3 LTS installed with default installation of Apache web server using apt packaging software in version 2.4.49. Victim has the IP address 10.0.0.100/24 connected to default router with IP 10.0.0.1/24.

Router with armv7 CPU and operating system OpenWRT interconnects above devices. As it is router with Linux OS, basic commands can be used to capture traffic using `tcpdump` program that generates `.pcapng` files. These files are used for further detection of the SDAs.

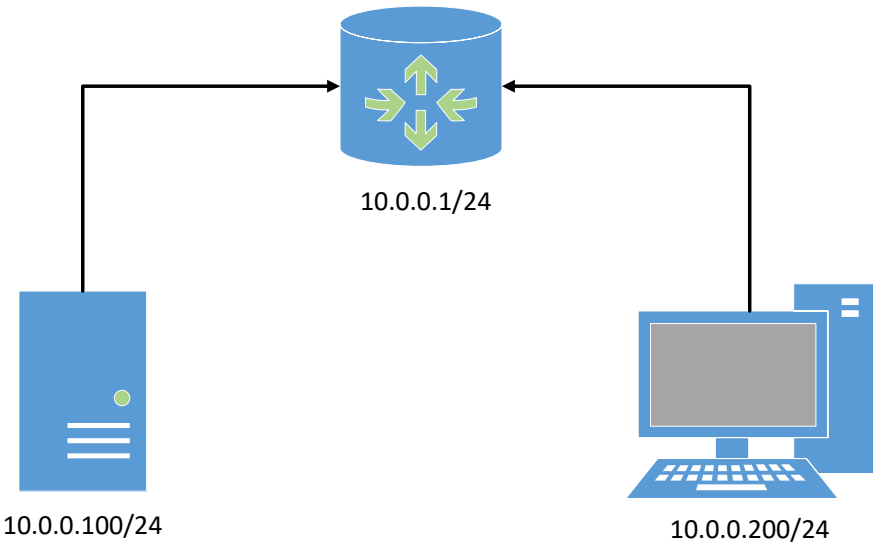


Fig. 5.1: Testing network

In the attachment of this thesis there is `generate_index.py` `python3` script that generates `index.html` file in given location (with optional argument `-p <string>`, default: `'/var/www/html/'`) with random data of given size (with required argument `size <int>`).

5.1 Web server

Default web server available in Ubuntu 20.04.3 LTS is Apache 2.4.49. It can be installed by `apt` packaging software or directly built from source files. Generally, Apache web server is based on modularity, which enables to configure core features modules add different settings modules, security modules, working modules or network modules to enhance web server functionality, performance and security.

Modules can be found inside the web server configuration folder:
`/etc/apache2/mods-enabled/` and can be loaded with
`Load Module <module_name> <path/to/module.so>` using `a2enmod` program. The Apache main configuration file `apache2.conf` is inside `/etc/apache2/` enabling time settings, requests settings, logging settings, includes module configuration, includes list of ports to listen on, web server default directory and includes virtual host configuration in lst. 5.1.

```
# ...
Timeout 300
# ...
MaxKeepAliveRequests 100
# ...
KeepAliveTimeout 5
# ...
```

Listing 5.1: Default timeout settings available from *apache2.conf*.

`Timeout` stands for the number of seconds Apache will wait to close connection. `MaxKeepAliveRequests` means the maximum capacity to be handled during persistent connection. If set to 0 allow unlimited amount. `KeepAliveTimeout` is one of the crucial parameters. It proposes the number of seconds to wait for the next request from the same opened connection.

One of the core networking modules is MPM (Multi-Processing Module), which configures the settings of the network ports binding or use of child processes and threads to handle requests. Apache offers 3 different MPMs:

- *prefork* – uses multiple non-threading child processes, where each process servers single connection,
- *worker* – uses multiple threading child processes, where each thread handles single connection,
- *event* – similar to *worker* but connections remaining in idle or keep alive state handles differently. They are grouped and handled by a single thread allowing to free up memory and server other threads. This is the default setting at Apache web server from version 2.4. [55]

As being said above the default configuration to handle multiple connection is MPM *event* available in `/usr/lib/apache2/modules/mod_mpm_event.so`.

```
<IfModule mpm_event_module>
StartServers             2
MinSpareThreads          25
MaxSpareThreads          75
ThreadLimit              64
ThreadsPerChild          25
MaxRequestWorkers        150
MaxConnectionsPerChild   0
</IfModule>
```

Listing 5.2: MPM event module parameters available from *mod_mpm_event.so*

`StartServers` is one of the parameters from lst. 5.2 determining the number of starting processes, when the Apache web server is started. `MinSpareThreads` sets the minimal number of working threads in spare or idle state during web server operation. `MaxSpareThreads` set maximal value of working threads in idle state. `ThreadLimit` sets the maximum value of server threads serving user requests. The memory for these threads is always allocated. Otherwise `ThreadsPerChild` sets the maximum number of threads created by each child process. It has to have lower value than `ThreadLimit`. `MaxRequestWorkers` is the most important parameter. For simplicity it sets the maximal value of established connections that the Apache web server can handle instantly. In the end `MaxConnectionsPerChild` limits the number of connections that child server process will handle during its life. After reaching this value of connection, the child process will die. If set to 0, then the process will never expire.[56]

Other way how to find Apache settings is with `apachectl`. It is open source program designed to provide control of the Apache httpd daemon. It can start, restart or stop web server from functioning or can be used for inserting environment variables in lst. 5.3. [57]

```
#!/bin/bash
apachectl -V
```

Listing 5.3: Finding basic Apache server settings

One of the most common Apache modules is *mod_security* which deals with security settings of the web server. This being set can protect web server from various attacks especially flooding, DDoS attacks or against XSS and SQL injection. Nowadays it stands as fully-fledged web application firewall. It can be configured by OWASP (Open Web Application Security Project) ModSecurity Core Rule Set that provides a set of generic rules to defend the web server against OWASP Top 10 attacks. This can be additionally installed. [58]

```
<IfModule reqtimeout_module>
# ...
RequestReadTimeout header=20-40,minrate=500
# ...
RequestReadTimeout body=10,minrate=500
</IfModule>
```

Listing 5.4: Request timeouts settings

One of the important default modules from Apache 2.4 is *mod_reqtimeout* available in `/etc/apache2/mods-enabled/reqtimeout.conf` and listed in lst. 5.4 provides different timeouts and minimum data rates for receiving requests. If the configured timeout occurs or data rate is belong the set value, connection will be closed by the server. It is basic protection against type of DoS attacks, where attacker opens new TCP connections and no or low data flows (TCP *SYN FLOOD*). `RequestReadTimeout` sets the maximum value of 20 seconds for the first byte of the request. From that moment it requires a minimum data rate of 500 $bytes/s$ and if that is not satisfied it will wait no longer than 40 s in total.
`RequestReadTimeout` sets the maximum waiting interval for the first byte of the request body. [59]

Another Apache security module is *mod_ratelimit* available in `/etc/apache2/mods-enabled/reqtimeout.conf` and listed in lst. 5.1 that provides `RATE_LIMIT` filter limiting client bandwidth (or rate limiting). It is available from Apache 2.4.24 and works correctly from 2.4.33. This mechanism is applied to every HTTP response to the client. It can be applied to certain location (e. g. data or upload path). It uses `rate-limit` variable to set connection speed in KiB/s or `rate-initial-burst` optional variable to set maximum amount of initial data before connection is slowing down in KiB. [60]

```
<IfModule ratelimit_module>
    <Location "/data">
        SetOutputFilter RATE_LIMIT
        SetEnv rate-limit 512
        SetEnv rate-initial-burst 756
    </Location>
</IfModule>
```

Listing 5.5: Ratelimit module example with output filter

Another possibility is to use custom created filters or specific modules. For example *mod_evasive* module that monitors incoming server requests. If more requests than set max limit income or more than 50 simultaneous connections are being created or requests done from blacklisted IP addresses *mod_evasive* module detects such behavior and responds with 403 error. [61]

In lst. 5.1 is default configuration from `httpd.conf` with following parameters:

- `DOSHashTableSize` – stands for the space that web server allocates.
- `DOSPageCount` – is the number of requests for individual page that trigger blacklisting.
- `DOSSiteCount` – stands for the total number of requests for individual page by one host (certain IP).
- `DOSPageInterval` – number of seconds for the requests can trigger blacklisting.
- `DOSSiteInterval` – number of seconds for the total number of requests for individual page by one host.
- `DOSBlockingPeriod` – time that IP address is blacklisted.

```
<IfModule mod_evasive20.c>
    DOSHashTableSize    3097
    DOSPageCount        2
    DOSSiteCount        50
    DOSPageInterval     1
    DOSSiteInterval     1
    DOSBlockingPeriod   10
</IfModule>
```

Listing 5.6: Evasive module default configuration [3]

5.2 Data capture and dataset

For further detection or prevention it is possible to create own datasets by capturing individual packets. They can be captured already filtered with `tcpdump` using destination (host) and port number filter. This approach has some downsides as privacy issues, large space for captured data or filtration mechanism.

```
timeout 60 tcpdump -i eth0 -nn host <IP-address>\
-nn port <port> -w output_file.pcapng
```

Listing 5.7: Traffic capture script on all interfaces restricted for 60 s duration

Other way is to use publicly available datasets. These data setsare usually thematically focused, providing any kind of network traffic with various anomalies or cyber attacks. One of the first was the 1998 DARPA dataset. [62] It contains traffic with 1000 hosts and more than 300 instances of 38 different attacks against UNIX hosts.

Another important dataset was NSL-KDD. It is an improvement of the KDD99 dataset containing two training, one testing set and small verifying dataset. [63]

In the last few years Canadian Institute of Cybersecurity at the University of New Brunswick started to creating very important datasets that are used around the world for the training attack detection models, IDSs, IPSs, firewall rules etc. [64]

The first important dataset is ISCX IDS dataset from 2012. It is a seven day Internet stream capture containing various application protocols (FTP, HTTP, IMAP, SMPTP or SSH). It contains normal and anomalous traffic that is labeled (e. g. using `BENIGN` or `NORMAL` and `ATTACK`). Before a lot of datasets were heavily anonymized deleting payload. That would for example lead for impossibility of application layer detection. This dataset contains 7 days of data capture resulting in the size of 84.42 GB in total. [65]

Another important dataset is CIC DoS dataset from 2017. This is the first dataset completely focused on application layer DoS attacks. It contains high-volume attacks (HTTP GET flooding, DNS flooding, SIP INVITE flooding ...) and low-volume, low-rate or Slow DoS attacks. It is focused on universal type of application Slow DoS attack in variation of slow sending and slow reading. It mixes generated Slow DoS attacks with attack-free traffic from ISCX IDS dataset (2012). It consists of 4 different types of Slow DoS attacks created with different tools, obtaining 8 different captures towards 10 web server with higher amount of connection in parent dataset. Resulting datset contains of 24 h of traffic with amount of 4.6 GB.[66]

Next dataset is Intrusion Detection Evaluation Dataset (CIC-IDS2017) from 2017. It consists of benign and malicious traffic. This dataset includes various types of network devices (modems, firewalls, switches or routers). It contains various application layer protocols (HTTP, HTTPS, FTP, SSH) and attacks (Brute force, DoS, DDoS, Heart-bleed). It also includes `.csv` files that are created from network captured files (.pcaps) and after network traffic analysis with `CICFlowMeter` and labeled flows containing 80 flow features. [1]

It contains 5 days of complete data capture on a testing network where each day contains some implemented attacks or normal traffic. Monday capture has 11.0 GB and includes only benign traffic. Tuesday capture has 11.0 GB and includes normal traffic with attacks. Wednesday has 13.0 GB and includes normal traffic with attacks. Thursday has 7.8 GB and includes normal traffic with attacks and Friday has 8.3 GB and includes normal traffic with attacks as well. In total 51.1 GB. Traffic feature and flow csv files are boldly smaller. Traffic feature csv files has 843 MB and flow csv files has 1.12 GB. [68]

The last dataset is DDoS Evaluation Dataset (CIC-DDoS2019) from 2019. It consists of benign and modern DDoS attacks traffic and csv files with labeled flow data. It was created during 2 days. For each day it contains capture of raw network traffic and event logs per each machine plus flow csv files per machine. [69]

Last approach is to combine a certain dataset with custom data. There are two possibilities. One to merge raw captured data with original dataset or create flow labeling of the custom data and subsequently merge it with original flowed dataset. In this thesis the approach of merging csv data into original dataset was chosen.

5.2.1 Flow and dataset creation

To generate flow according the parameters described in sec. 1.2.1 it is needed to create custom flow generator from captured network traffic. With CIC-IDS2017 dataset there was custom flow generator proposed – CICFlowMeter. [70] It creates traffic flow with lower number of features. Also `python3` implementation of such generator exists – Python CICFlowMeter. [71] It creates flow with parameters included in lst. D.3. It is used to create `.csv` files from original dataset and from captured attack datasets.

Dataset CIC DoS (2017) with custom attack can be used further with Unsupervised learning methods. CIC-IDS2017 contains labeled flows with custom attack labeled flows can be used with Supervised learning methods.

[1] `CICFlowMeter` is an Ethernet traffic bi-flow generator and analyzer for anomaly detection. It creates labeled flows based on time stamp, source and destination IPs, source and destination ports and protocols. It is written in Java.[67]

Following steps has been taken:

1. Create labeled flow `.csv` files from network traffic captures using:
   - Install Python CICFlowMeter:

```
git clone https://github.com/datthinh1801/cicflowmeter.git
cd cicflowmeter/
python3 -m venv venv
source venv/bin/activate
python3 setup.py install
```

Listing 5.8: Python CICFlowMeter download, instalation and activation

   - Convert pcap files to flow csv files (for larger files it takes a few hours):

```
cicflowmeter -f slow_read.pcap -c slow_read.csv
```

Listing 5.9: Example of converting `slow_read.pcap` file into `slow_read.csv` file

2. Change timestamps of custom SDAs according to tab. D.1.

```
import pandas as pd
slow_read_time = 13800 # 230 minutes
# ...
original_file = 'original.csv'
slow_read = 'slow_read.csv'
# ...
df = pd.read_csv(slow_read)
rows = df.shape[0]
for i in range(rows):
    df.loc[i,'timestamp'] += time
df.to_csv(file,index=False)
```

Listing 5.10: Example of incrementing flow time in csv file

3. Merge csv to single dataset csv file.

```
import pandas as pd
source = pd.read_csv(original_file)
to_merge = pd.read_csv(file)
result = pd.concat([source, to_merge])

result.to_csv(destination, index=False)
```

Listing 5.11: Example of two csv files merging

6 Slow DoS Attacks Detection

Various approaches to the detection can be applied. In this part the structure of the attacks based on various protocols is strictly defined as follow:

- Application layer – HTTP protocol in version 1.1,
- Transport layer – TCP protocol creating TCP sockets.

Basic division of detection techniques is to real-time detection and subsequent (offline) detection. Real time detection can be done with IDS (Intrusion Detection System). This can be any hardware or software device placed in network or run as a daemon on intermediary devices sniffing the network traffic. If the detection is positive, any unusual activity detected is reported to SIEM (Security Information and Event Management).

Detection can be focused on detection of incoming traffic inside the network. IDS doing such job is called NIDS (Network Intrusion Detection Systems). On the other side detection on the host side is called HIDS (Host-based Intrusion Detection System). This monitors logs, files and folders, data and memory structures and running processes inside the host's operating system.

The further division that can be used for both above approaches is on:

- Signature-based detection – Detection mechanisms is looking for specific structure of the malicious traffic. It can be some specific repeating patterns, time periods or known malicious data. Signature detection can be then used in automated IDS and IPS (Intrusion Prevention System). Two best known IDS/IPS are Suricata and Snort. Signature of the attack can be created by the defined rules. With this approach we can detect only known attacks listed in common vulnerability databases or CVEs (Common Vulnerabilities and Exposures systems). [72]
- Anomaly-based detection – Detection mechanisms is monitoring the resource activity and classifying it to normal or anomalous. To distinguish the legitimate traffic from malicious we need the mechanisms of classification. This usually contains two phases – training phase and testing phase. The purpose of training phase it to create a model of legitimate traffic. On the other hand in testing phase the malicious traffic with legitimate traffic is tested against the training model. The deviations or anomalies are detected using machine learning algorithms, artificial intelligence techniques, neural networks or statistical strict anomaly detection. The advantage of this approach is that it can detect even unknown attacks. But the downside is that it can suffer from the false positive detection results. (Legitimate traffic can be marked as malicious and vice versa.) [73]

6.1 SDAs Signature Detection

Signatures can have various forms. It can be pattern in the protocol payload containing malicious code, chain of commands that attacker tries to execute to take profit, network behavior of threat actor or unauthorized network access. [74]

Signature detection can focus on single-packet detection (atomic signatures) that is more resource intensive and time consuming or multiple-packet detection (composite signatures) that requires less resources. [75]

Downside of signature detection is that it is used for predefined or static signatures that represent given threat. If attacker changes something in the look, code or approach it should evade the detection. So for that, combination of signature detection with behavior or anomaly-based detection is required.

```
{
    "name": string,
    "initial_request": {
        "content": "{METHOD} {PATH} HTTP/1.1",
        "window_size": integer
    },
    "pending_request": {
        "content": "{METHOD} {PATH} {DATA}",
        "repeat": boolean,
        "repeat_count": integer
    },
    "response": {
        "status_code": integer,
        "content": string,
        "keep_alive": integer
    },
    "connection":{
        "connections": integer,
        "connection_rate": integer,
        "threads": integer,
        "sending_buffer": integer,
        "retransmissions": integer
    },
    "timeouts": [{req}, {delay}, {resp}, {next}],
}
```

Listing 6.1: Host-based SDAs specification

```
{
    "name": string,
    "target": ip_address,
    "port": integer,
    "user_agent": string,
    "initial_request": {
        "content": "{METHOD} {PATH} HTTP/1.1",
        "window_size": integer
    },
    "pending_request": {
        "content": "{METHOD} {PATH} {DATA}",
        "repeat": boolean,
        "repeat_count": integer
    },
    "timeouts": [{req}, {delay}, {resp}, {next}],
}
```

Listing 6.2: Network-based SDAs specification

In lst. 6.1 and lst. 6.2, host-based and network-based signature specification of SDAs is proposed for detection. The scheme of the attack is based on `json` structure describing key parameters of the attack regardless the source of the attack.

Parameter `name` stands for the name of SDA. Parameter `initial_request` determines the first request attacker has to send. It can consist of subparameters or the only one string of HTTP request. Between subparameters are `content` parameter, optional parameter of the TCP window size – `window_size`. Then `pending_request` is one of the most important parameters determining mechanism of the attack. Contains various subparameters as `content`, information of repetition `repeat`. If `repeat` set `repeat_count` is required. Next parameter is `response` containing `status_code` informaiton, `content` parameter of the response and next optional parameters (`keep_alive`). Then the basic volumetric parameters are set: `connection` with `connections` specifying the number of opened connection at total. Then `connection_rate` specifies the rate of the new connections opening. `Threads` stands for the number of parallel threads operating the opened connections. Resending parameter `retransmissions` means the number of not acknowledged responses. And `sending_buffer` determining the size of the response queue. The last field is `timeouts` parameters containing timeout settings from sec. 2.1.1.

If we want to count the source information of the attacker alongside the detailed information of the victim, we have to extend the specification in lst. 6.2.

Where extra parameters are `target` specifying the IP address of the victim alongside the `port` and `user_agent` giving some information about attacker's device.

Specification examples of the related SDA are lst. D.1 for Slow Read attack, lst. D.2 for Slow Drop attacks and lst. D.3 for Slow Next attack.

6.2 SDAs Anomaly or Behavior-Based Detection

Anomaly detection creates a model of legitimate traffic and afterwards the malicious traffic model is created. This model deviates from valid one. Between anomaly-based methods we count statistical anomaly detection methods, wavelet analysis, entropy-based methods, machine learning techniques and neural networks.

The goal of the statistical anomaly methods is to identify traffic parameters that deviates from normal traffic. For each of the parameters we can compute statistical properties as mean value, variance, distribution function etc. Then the statistical tests are applied to determine the deviation from the normal profile. The traffic that deviates from the normal traffic is evaluated with score, if that score exceeds the threshold value we can consider such as a malicious traffic. [76]

The wavelet analysis can be used to analyze the edges (similar to images). The edge can represent threshold value as in statistical anomaly methods. [77]

The entropy-based methods come from information theory introduced by Claud Shannon, where entropy of a random variable is the average level of information. The principle is that the more redundant data being sent, the lower the entropy value so the compression is easier. Two types of compression algorithms can be used: Dictionary based (Lempel-Ziv-Welch algorithm) and Model based (Huffman Coding). If the distribution of the traffic belongs to certain class the entropy is lower. If the entropy value is larger, the traffic belongs to many classes. [76]

The machine learning methods are used for legitimate and malicious attack differentiation. Machine learning methods are divided to Supervised learning methods, Unsupervised learning methods and Semi-supervised learning methods. The supervised methods are used for classification or categorization and prediction (regression). It is based on two datasets – training dataset and testing dataset. The training dataset consists of inputs and correct outputs to learn the model, sometimes called learning with teacher. The accuracy of trained model is measured by error function. The classification problem assigns testing data to limiting categories. On the other side regression tries to find connection between dependent and independent variable and predict future values. [78]

Whereas the unsupervised methods are used for clustering. They try to discover hidden unknown patterns among unlabeled data. Process of learning is without external teacher. Semi-supervised methods are combination of the previous. Examples

of the supervised learning methods are Naïve Bayes, Decision Trees, Support Vector Machines, K-nearest Neighbors, Linear Regression or Neural Networks (Multilayer networks). Examples of the unsupervised learning methods are K-means Clustering, certain types of Neural Networks (Hopfield networks, Helmholtz network or Autoencoder networks). [79]

7 Machine Learning SDA Detection

Machine learning (ML) is part of artificial intelligence that is used to create analytical models based on sample data. The idea of ML is that the system can learn from data or identify important patterns or parameters without being explicitly told. It can solve various problems where no algorithms are presented or only inefficient. ML contains various approaches divided into following categories: [80]

- Supervised learning – Data that contains valid inputs and desired outputs (classified and labelled) are called training data. They are used to create mathematical model that is through iterative approach created and can be used to predict the output of testing data (separated from training data). Usually performance is measured by performance methods. Basic types of supervised learning methods are classification and regression.

  Classification is used to create or distinct categories. Classification tries to create boundary between marked data, e. g. `BENIGN` vs `ATTACK` traffic. It can not be strictly binary classification. `ATTACK` category may contain multiple attack types.

  Regression is used when outputs have numerical values in some interval. It can be used for predicting future values or estimate output according the input (depending) values.

  Examples of supervised learning algorithms are Logistic regression, Support-vector machine, Decision Trees and Random Forests.
- Unsupervised learning (**Clustering**) – Take a set of data that does not contain classification. It contains only input data and the result of a process is to find structure in that data. Algorithms create clusters according various features and observers their associations. It is often used in Anomaly detection and

  Examples of unsupervised learning algorithms are Neural networks or K-means.
- Reinforcement learning – Is based on multiple simulation and rewarding function where the goal is to maximize a reward. Also for wrong or faulty behavior can be assigned a penalty. It does not need labelled dataset.

7.1 ML methods

ML methods as described above are algorithms solving some learning challenges. With supervised learning it is needed to have training dataset and testing dataset to correct learning process. Then third dataset can be proposed to make prediction for validation dataset. It should be completely new data. On the other hand

unsupervised learning requires only training dataset. It is faster than supervised learning. Reinforcement learning is not taken in mind in this thesis.

7.1.1 Decision Trees

It is an algorithm of supervised learning. It is one of the most popular classifiers. It uses trees where nodes contains decision statement and leaves contain result of the decision. It can be value, label, category or class. It gives the unstructured big data some meaning and structure. But this method is prone to overfit for complex trees.[81]

7.1.2 Random Forest

It is an algorithm of supervised learning. It can be used for various tasks that requires using multiple decision trees at training phase. It can be used for classification where final class is determined by the majority of single decision trees. Or it can be used for regression prediction of mean or average values in trees. It prevents training from overfitting. [82]

7.2 ML performance methods

The most important method of ML classification is confusion matrix also known as error matrix. It provides metric for evaluation performance of ML model. In this thesis confusion matrix is used to evaluate the performance of classification algorithm of supervised learning. Normal benign flow is marked as N and attack flow is marked as A.

Type	Network Activity	Detection Classification Result
True Negative (TN)	Normal network traffic	`BENIGN (0)`
True Positive (TP)	Attack traffic	`ATTACK (1)`
False Negative (FN)	Normal network traffic	`ATTACK (1)`
False Positive (FP)	Attack traffic	`BENIGN (0)`

Tab. 7.1: Results of detection – confusion matrix

Following classification metrics are used to determine success of classification:

- Accuracy (ACC) – refers to the number of correct detection over all detection made by classification or ML model. [83]

$$ACC = \frac{TP + TN}{A + N} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

- Precision (PR) – refers to true classification of an attack. It says about the detection method how precise is attack prediction in comparison with all classified data as an attack. [84]

$$PR = \frac{TP}{TP + FP} \tag{7.2}$$

- Sensitivity (SN) – refers to true classification of an attack. It says about the detection method how well it can detect an attack. [85]

$$SN = \frac{TP}{A} = \frac{TP}{TP + FN} \tag{7.3}$$

- F1 Score ($F1$) – represents a harmonic mean of precision and sensitivity. It can be used to measure detection method accuracy. [83]

$$F1 = \frac{PR + SN}{2} = \frac{2TP}{2TP + FP + FN} \tag{7.4}$$

7.3 ML Implementation

Implementation of ML consists of following steps. First step is to **collect data** that we want to process. This step is crucial. Without quality data there is no machine learning reflecting the given problem. More quality data, more accurate is machine learning model. [86] Next step is **preprocessing** that prepares the data for training. In this step data are cleaned, not important columns are dropped and all non-numerical data are converted to standardized numerical format. Next part is **feature selection** that is process of selecting important features or columns for further use in machine learning model. Unimportant feature are dropped. In this part it is decided which ML strategy will be used. Next part is **machine learning model training** where training data are used to train ML model. Next part is **evaluating ML model, tuning and selection** which uses ML performance methods or metrics to evaluate performance of ML model and tries different model parameters to select best ML model.

Two approaches can be taken according to fig. 7.1. Different SDAs and benign traffic can be classified into multiple categories. Or all Slow DoS Attacks and benign traffic can be classified using binary classification. In first example feature extraction can be done on each file separately or on one merged file. Next step of implementing of ML algorithm is common for both paths. In this thesis second approch has been chosen.

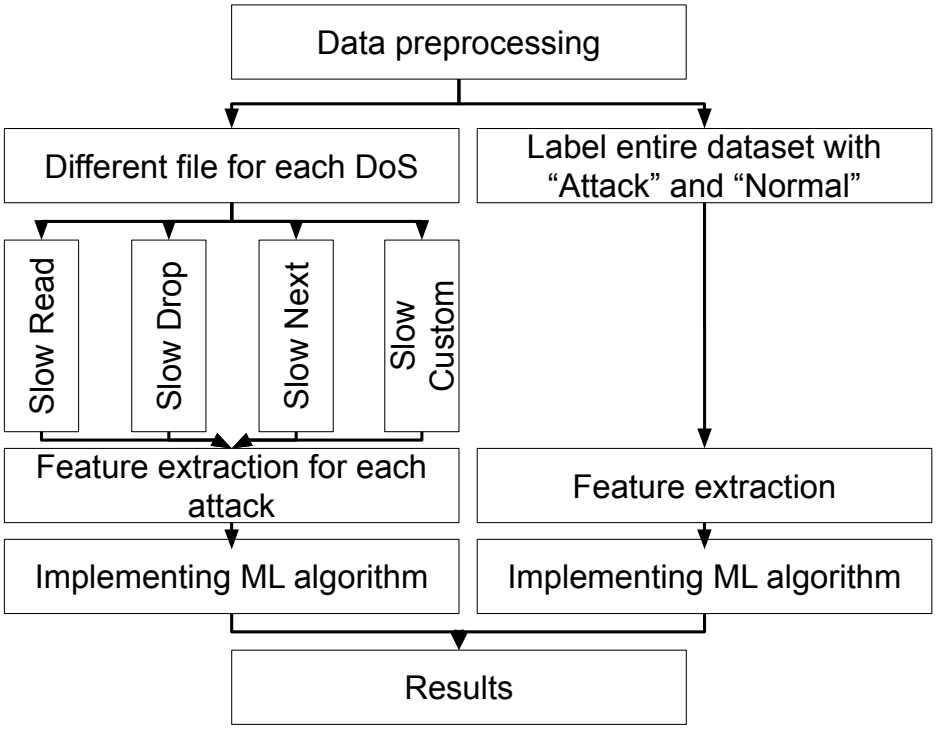


Fig. 7.1: Machine learning detection implementation

Modules

ML section is implemented in Jupyter Notebook web-based interactive environment using `python3`. Other used libraries are:

- `numpy` – is a library that provides tools and support for large vectors, arrays and matrices with mathematical functions to operate them.
- `matplotlib` – is a library that provides an API to plot graphs.
- `scikit-learn` – is a open-source machine learning library for python. It provides implementation of various machine learning algorithms and tools.
- `pandas` – is a library that provides tools for data analysis and manipulation.
- `seaborn` – is a statistical data visualization library based on `matplotlib`.
- `pickle` – is a python module that implements tools for serializing and de-serializing python objects. Python objects are converted to byte stream for saving and reverse operation for loading. It can be used for ML model saving.

7.3.1 Data creating and collecting

In sec. 5.2.1 `.csv` file of captured traffic was created using Python CICFlowMeter. CICIDS2017 has been chosen as an original dataset. From that dataset generated and labelled flow `.csv` files has been used. Custom generated `.csv` files and from dataset has different feature columns. As the goal of this section is to create single

file with original and custom data, different structure is an issue. It is done using transfer map listed in lst. 7.3.1.

```
column_map = {
    'Flow ID':'Flow ID',
    ' Source IP':'src_ip',
    ' Source Port':'src_port',
    ' Destination IP':'dst_ip',
    ' Destination Port':'dst_port',
    ' Protocol':'protocol',
    ' Timestamp':'timestamp',
    ' Flow Duration':'flow_duration',
    ' Total Fwd Packets':'tot_fwd_pkts',
    ' Total Backward Packets':'tot_bwd_pkts',
    'Total Length of Fwd Packets':'totlen_fwd_pkts',
    ' Total Length of Bwd Packets':'totlen_bwd_pkts'
    # ...
}
```

Listing 7.1: Transfer map between original and custom files

Custom dataset has missing `'Flow ID'` column. Custom function that creates flow identifier is created.

```
def flow_id(df,index,row):
    return df.loc[index]['dst_ip']+'-'+df.loc[index]['src_ip']\
    +'-'+str(df.loc[index]['dst_port'])+'-'\
    +str(df.loc[index]['protocol'])
```

Listing 7.2: Flow ID generator function

For each custom file redundant columns `'src_mac'` and `'dst_mac'` are deleted. Then columns of custom files are renamed and redistributed according to original files. Then original files contain redundant column `' Fwd Header Length.1'` that is dropped subsequently.

In next part single `supervised.csv` file is created. It contains all original files and 20 copies of each of custom attack files.

7.3.2 Preprocessing

In this section data are prepared and cleaned for next processing. `supervised.csv` file contains 3178805 rows of data and 85 columns. In this part data that has no Flow ID or Label are dropped. It is 288602 rows of data. Then data that has not numerical values are processed:

- `inf, -inf, nan` records are replaced with -1.
- `type('4') == str` numerical records in other types are converted to integers using `pd.to_numeric` function.
- String values that represents Flow ID, Source and Destination IP and Timestamp are replaced using `LabelEncoder()`. [87] It provides functionality to replace data features that contains other values than numerical data with integers.

In the next part there is a need to check the ration between normal traffic records and attack traffic records as well focus on SDAs has to be taken in mind. Ratio between normal (benign) traffic to normal traffic is 0.7865. Otherwise ration between complete attack traffic to normal traffic is 0.2135. Ratio between selected SDAs and normal traffic is 0.0205 and attack traffic is 0.0963.

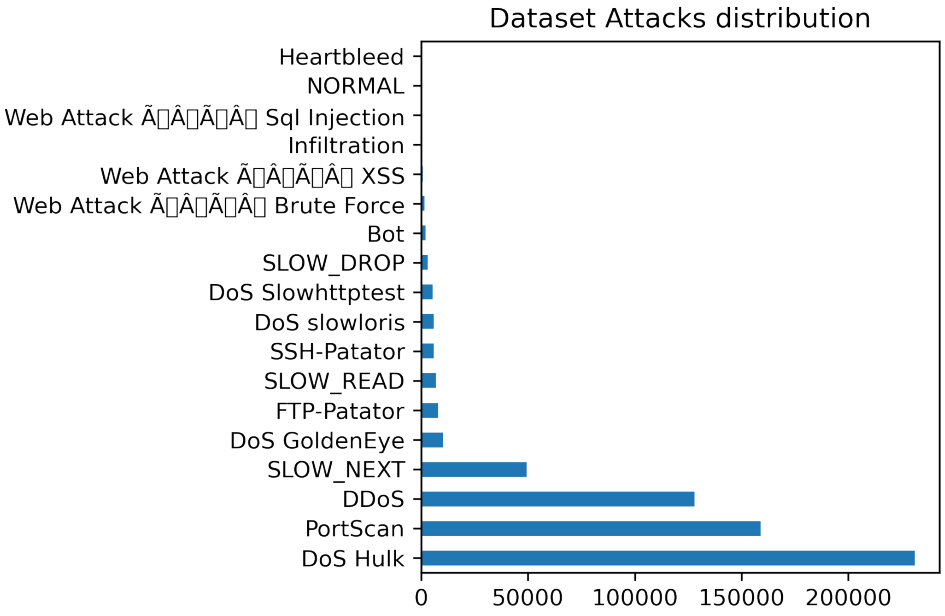


Fig. 7.2: Distribution of individual attacks

As the ratio should be balanced (aprox. 30% of attack traffic and aprox. 70% normal traffic) and SDAs does not require that condition, it is needed to crate balanced dataset `supervised_sda_balanced.csv`. Maximum number records of normal traffic satisfying above distribution is 138693 and number of Slow DoS records is 59440.

Probability to choose normal records labelled as `BENIGN` is 0.05278.

Final balanced dataset contains following data:

Flow label	Number of records	Numerical labels
BENIGN	119845	0
SLOW_READ	49420	1 (or 1)
SLOW_DROP	7020	1 (or 2)
SLOW_NEXT	3000	1 (or 3)

Tab. 7.2: `supervised_sda_balanced.csv` dataset record distribution with numerical labeles

Next step is to prepare data for training. It is needed to replace string labels of records with numerical values. For that there is needed a map between label and numbers. According tab. 7.2 it depends what type of labeling is chosen. If binary labeling is presented (`BENIGN == 0` and `SDA == 1`) binary classifier will be used. Otherwise multiple classes are presented. Another step is to drop columns that should not influence result. Flow ID with columns that are forming flow identifier can be dropped as the shape of the flow traffic is more important. Following column are dropped `Flow ID`, `Source IP`, `Source Port`, `Destination IP`, `Destination Port`, `Protocol` and `Timestamp`.

7.3.3 Feature selection

Feature selection is a process of selecting important features or columns from dataset that are used for machine learning model construction. It provides several benefits for machine learning. It simplifies models, makes model training times shorter or reduce the dimension of the dataset. It tries to find redundant and not important data between features. [88]

Simple algorithm is to test each possible subset of features against training dataset and tries to minimize error rate. So for that it is crucial to select correct evaluation metric that is divided into following categories:

- Wrappers – create models for every combination of subset from features and select features that result is best according to performance metric. Use specific machine learning algorithm to find optimal features. It takes a the most of computational time to find optimal features. It has downside of overfitting. Examples methods are **Forward Selection**, **Backward Selection**, **Stepwise selection** or **Recursive Feature Elimination (RFE)**.
- Filters – use statistical methods to evaluate relationship between features and target variable (e. g. `Label`). These methods are faster in time complexity.

They are not prone to overfit. Examples methods are **Correlation**, **Chi-Square test**, **ANOVA**.

- Embedded methods – are methods that performs feature selection during machine model construction process. Feature selection is done by observing each iteration of model training phase. These methods are faster that wrappers. These methods are used to solver overfitting using coefficient penalization. Example methods are **LASSO**, **Elastic Net** or **Ridge Regression**.

First step is to create matrix X which contains all data without labels and vector y that contains only label values. Matrix is divided using `train_test_split()` function into two submatrices in ratio (70:30) for training and testing. Vector is divided as well. Next step is to create Decision tree using `DecisionTreeClassifier`. Simple decision tree is created from X_train and y_train sets with maximum of 5 leaf nodes. To avoid overfitting it is crucial to validate model against testing data that are separated from training data. It done using `cross_val_score()` evaluation function with `KFold` cross-validator. [89]

The result is trained decision tree on given features. It is visualized in fig. 7.3. It provides most important features and values as a decision statement which determines final class of flow classification. Depending the index of the leaf value it sorts classified flow to two classes. Index 0 classifies to normal flow, otherwise index 1 classifies to SDA flow.

Using confusion matrix the result of classification can be visualized. It correlates with tab. 7.1 that determines if the classification was successful or not. Each row determines real data in actual class, while data in rows represents instances in a predicted class.

Following columns (features) has been choosen according the feature importance from decision tree.

Feature name	Importance value
`FIN Flag Count`	0.94592
`Fwd Packet Length Mean`	0.05404
`Total Fwd Packets`	0.00003353
`Bwd IAT Min`	0.00000239415

Tab. 7.3: Important features selected using decision tree

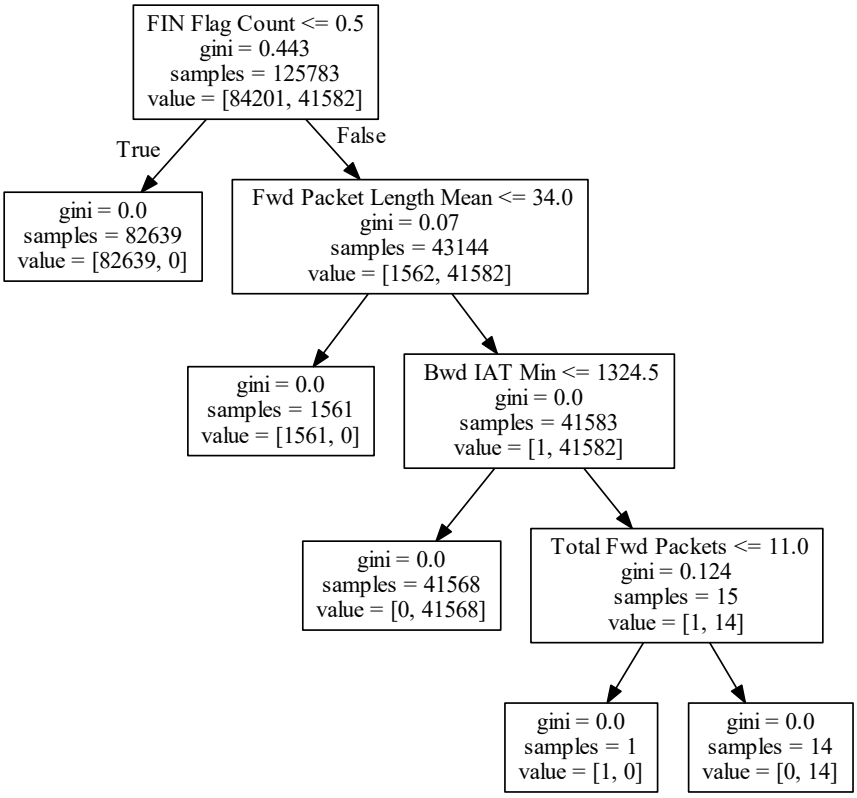


Fig. 7.3: Decision tree on selected features for Slow DoS attacks

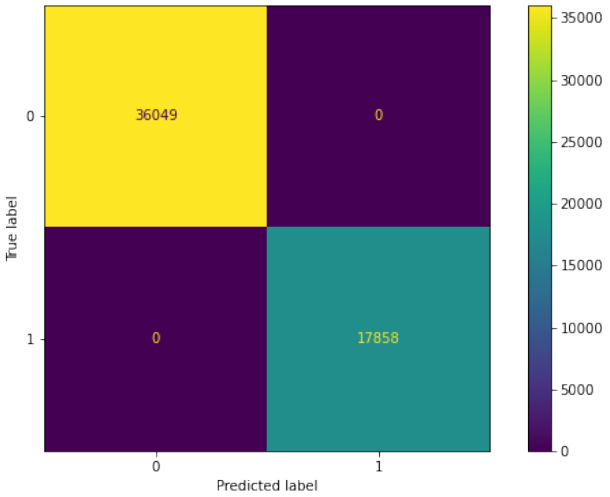


Fig. 7.4: Confusion matrix of decision tree classification

7.3.4 Machine learning model training

Next part is to combine multiple decision trees into random forest classifier machine learning model using sub-samples of training set. The sub-sample size is defined by `max_samples` parameter that stands for the number of samples that are drawn from X_train set to train each decision tree. Number of tress in the forest is determined by `n_estimators` paramter. Default value is 100 decision trees.

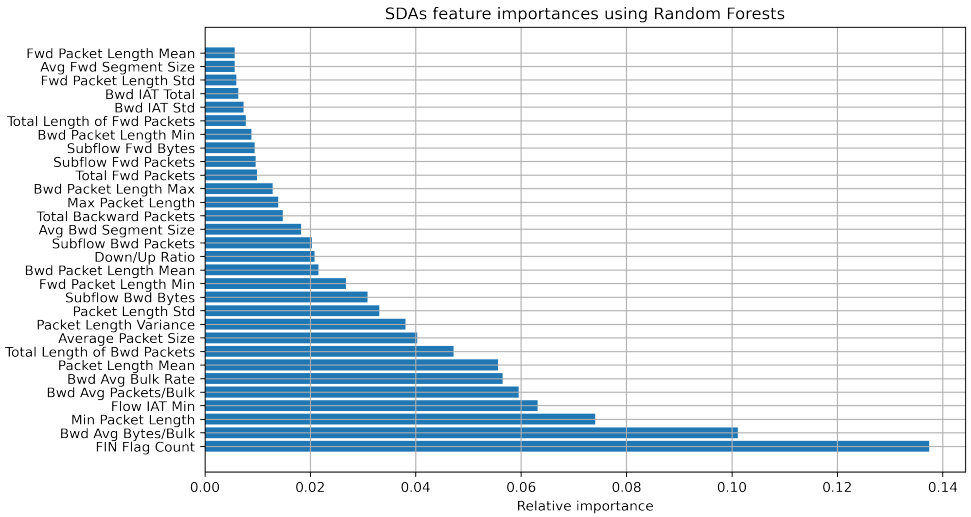


Fig. 7.5: Important features for SDA selected using Random Forest

The most important feature is `FIN Flag Count` that correlates to decision tree classification as the most important feature. Second parameter `Fwd Packet Length Mean` is on the 13. place in random forest. Third parameter `Total Fwd Packets` is on 21. place. Selected features for SDAs are in fig. 7.5. Confusion matrix is the same as for decision tree. To find redundant feature, correlation between columns in the dataset function `corr()` is used. Intersected values are not selected for correlation. Correlation of columns show how dependent columns are on each other. If the value of correlation coefficient is equal or close to 1 that column are correlated and can be dropped. [90] Correlation matrix for SDAs is in fig. 7.6 as a heat map.

Process of selecting correlated columns is following:

1. Create absolute value of correlated coefficients using `abs()` function.
2. Create upper triangular matrix of correlated coefficients using lst. 7.3.

```
upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),\
k=1).astype(np.bool))
```

Listing 7.3: Process of creating upper trinagular matrix

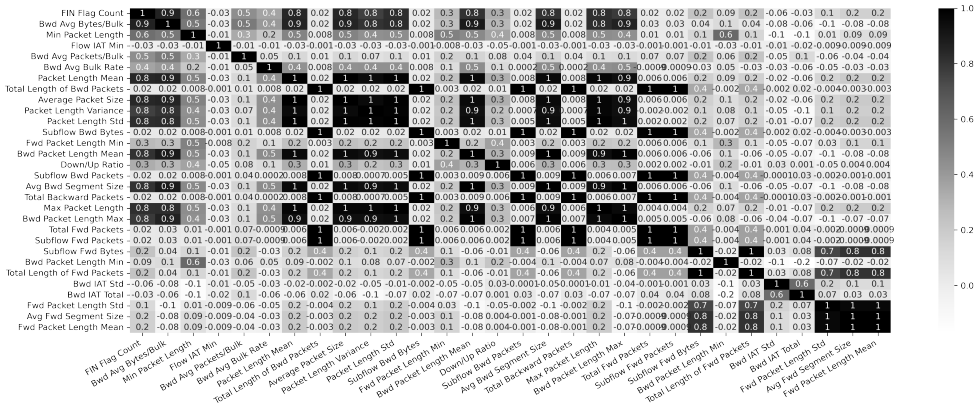


Fig. 7.6: SDA feature correlation heatmap

3. Select column that is in upper trinagular matrix and has correlation coefficient bigger than 0.95.

Following columns are correlated and therefore dropped. Whenever building IDS it can provide lead to what columns should be focused on. Also highly correlated columns provides redundant information that is not needed for SDA detection.

```
[' Average Packet Size',' Packet Length Variance',' Packet Length Std',
' Subflow Bwd Bytes',' Bwd Packet Length Mean',' Subflow Bwd Packets',
' Avg Bwd Segment Size',' Total Backward Packets',' Max Packet Length',
'Bwd Packet Length Max',' Total Fwd Packets','Subflow Fwd Packets',
'Total Length of Fwd Packets',' Avg Fwd Segment Size',
' Fwd Packet Length Mean']
```

Listing 7.4: Dropped correlated columns of SDA features

The final correlation heatmap is in fig. 7.7.

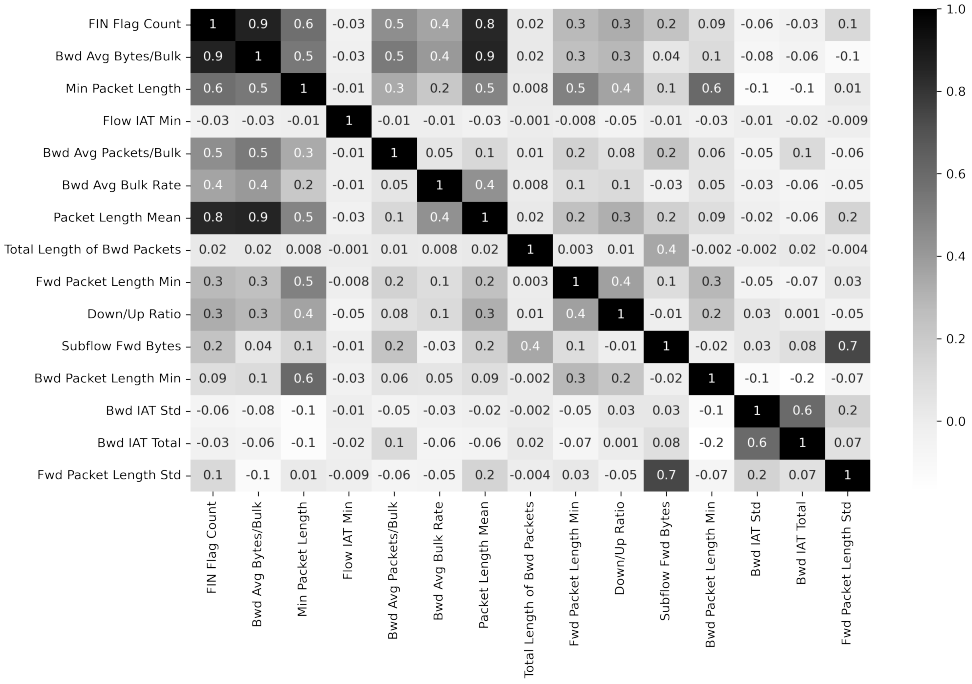


Fig. 7.7: SDA feature correlation heatmap after optimization

7.3.5 ML evaluating, selecting parametets, tunning and saving

To make the random forest classifier more precise it is needed to find correct parameters. It can be done using the hyper-parameter tuning methods. These search all parameters combinations (`GridSearchCV`) or random samples from parameter spaces (`RandomizedSearchCV`) to find best cross validation score. [91] As random forest classifier selected most important features and finds correlated column it is not needed to find better setting of random forest classifier.

Metric	Results
Accuracy	1.0
Precision	1.0
Sensitivity	1.0
F1 Score	1.0

Tab. 7.4: ML model metrics

Machine learning model can be serialized using `pickle` library and saved as `.pkl` file. Size of machine learning model `sda_ml_detection_model.pkl` is $453\,KB$.

Conclusion

This master thesis deals with the problematic of the Slow DoS Attacks. It proposes some network communication parameters. Practical result of this thesis is `python3` generator implementation of selected SDAs (Slow Read, Slow Drop and Slow Next) and with possibility to create custom defined SDA. The next result is classification of SDA using machine learning methods.

First chapter sums up basic TCP communication concepts. How to create TCP socket, how to end TCP socket peacefully and violently. It compares closure methods depending the side (server and client). It contains description of application layer communication and practical structure of HTTP protocol to be used later in the generator. Following section describes communication measures and parameters according the side of detection (client, server, network) that can be useful in IPS or IDS creation process. Then there is focus on how the network flow is created. Two types of flow identifiers are proposed. In following subsections there are flow parameters discussed. General parameters describes standard network behavior, Inter-flow parameters describe flow information and Intra-flow parameters show information about certain packets. Volume parameters size or rate information, time parameters reflect time division of SDAs introduced in the next chapter. Feature and TCP Analysis parameters describe information about the count of the TCP and TCP Analysis flags inside the flow. Last category of parameters are application layer parameters that abstract from the TCP flow.

In the second chapter there is DoS specification explained with division to DoS and DDoS. DoS attacks are categorized to Flood-based DoS attacks and Exploit-based attacks. Then the SDAs are proposed as part or subcategory of the DoS Attacks. This thesis summarizes attack categorizes of SDAs and examples of such attacks. Then there is a time division according the phases of TCP communication with HTTP protocol as an application layer. In the last part of the second chapter there is a matching table between SDAs attack types and timeout parameters.

Third chapter describes selected SDAs (Slow Read, Slow Drop and Slow Next). For each attack communication schema is shown with important parameters defining given attack. Next part sums up methods of generation of given attack with custom generator settings.

In the fourth chapter generator of SDAs is proposed written in `python3` script language. At first python modules modeling HTTP payload are described where `socket` library has been chosen and described. Then TCP connection and attack closure possibilities were described with appropriate TCP flags. In the next part multiple connection handling and methods of concurrent execution (asynchronous vs multithreading) is described. In the next part very generator with modules,

arguments and structure is proposed. The behavior and modes of custom generator are described. In the last part logging option is discussed.

Fifth chapter describes testing environment and ways to create dataset. In the first part web server settings is describes with default Apache web server implementation and modules. Then another possible modules that can be activated or imported to protect web server against various attacks are enclosed. Next part deals with possibilities of real time packet capture against available dataset that should be used. In the next section algorithm of creating traffic flow is described.

In the sixth chapter various approaches of detection are described. Two main categories are signature-based detection and anomaly-based detection. Parameters of signature-based detection are described for SDAs. Then anomaly-based and behavior-based detection is discussed.

Seventh chapter deals with machine learning detection of SDAs. Three main categories of ML detection are described: supervised learning, unsupervised learning and reinforcement learning. In the next section decision trees and random forest supervised methods are described with performance methods for evaluation of success of these methods. Next part deals with implementation of machine learning. First part of ml is data crating and collecting. Then preprocessing of the data is described. In the next part feature selection is proposed for a single decision tree. Next part describes machine learning model training and the last part of this thesis deals with ml evaluation, parameter selection, tuning and ml model saving.

One of the two main goals of this thesis was to implement SDA generator that can create custom defined attacks as well as three specified SDAs (Examples are in sec. C.4).

Second main goal was detection of modern SDAs. At first division of detection methods to signature and anomaly-based detection was introduced. Theoretical signature for selected SDAs was proposed as a json schema. The result of anomaly-based detection is the supervised machine learning model (random forest) based on decision trees suitable for offline detection of selected SDAs with 100% success according measured parameters (Accuracy, Precision, Sensitivity and F1 Score). With combination of network flow creating script it can be used in IDS.

For the future work, generator could be extended by application layer fuzzer as well as module and socket function tests could be created. Flow meter script could implement features introduced in this thesis (esp. TCP Analysis features). Machine learning model can be extended by methods of unsupervised learning for important feature selection using K-means algorithm or Autoencoder networks.

Bibliography

[1] Enrico Cambiaso, Gianluca Papaleo, Giovanni Chiola, and Maurizio Aiello. Slow dos attacks: definition and categorisation. *International Journal of Trust Management in Computing and Communications*, 1:300–319, 10 2013. doi: 10.1504/IJTMCC.2013.056440.

[2] Sergey Shekyan. Slowhttptest (1) - linux man pages. [online]. [cit. 5. 12. 2021]. URL: https://bit.ly/3DKyFsF.

[3] Apache evasive maneuvers module. [online]. [cit. 13. 5. 2022]. URL: https://github.com/jzdziarski/mod_evasive.

[4] Debbie Walkowski. What is the cia triad ?, 2019. URL: https://bit.ly/3DPNjyE.

[5] Transport control protocol. [online], 2001-. [cit. 21. 3. 2022]. URL: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[6] W. Richard Stevens. *TCP/IP illustrated*. Addison-Wesley Publishing Company, Reading, 1994.

[7] Rashmi Bhardwaj. What is tcp fin packet? [online]. [cit. 21. 3. 2022]. URL: https://ipwithease.com/what-is-tcp-fin-packet/.

[8] Rashmi Bhardwaj. Tcp rst flag. [online]. [cit. 21. 3. 2022]. URL: https://ipwithease.com/tcp-rst-flag/.

[9] Charles Kozierok. The tcp/ip guide. [online]. [cit. 3. 4. 2022]. URL: http://www.tcpipguide.com/free/t_HTTPEntitiesandInternetMediaTypes.htm.

[10] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. URL: https://www.rfc-editor.org/info/rfc2616, doi:10.17487/RFC2616.

[11] Flow network. [online], 2001-. [cit. 4. 4. 2022]. URL: https://en.wikipedia.org/wiki/Flow_network.

[12] Protocol numbers. [online]. [cit. 17. 4. 2022]. URL: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml.

[13] Tony Fortunato. Network analysis: Tcp window size. [online]. [cit. 5. 4. 2022]. URL: https://www.networkcomputing.com/data-centers/network-analysis-tcp-window-size.

[14] Maximum segment size. [online], 2001-. [cit. 5. 4. 2022]. URL: https://en.wikipedia.org/wiki/Maximum_segment_size.

[15] Tcp analysis. [online]. [cit. 2. 5. 2022]. URL: shorturl.at/ixyX3.

[16] Tcp analyze sequence numbers. [online]. [cit. 3. 5. 2022]. URL: shorturl.at/svGUW.

[17] Justin Baker. Tcp rtos: Retransmission timeouts & application performance degradation. [online]. [cit. 3. 5. 2022]. URL: shorturl.at/vyE27.

[18] Abi Tyas Tunggal. What is an attack vector? 16 common attack vectors in 2022. [online]. [cit. 9. 3. 2022]. URL: https://www.upguard.com/blog/attack-vector.

[19] Att&ck matrix for enterprise. [online]. [cit. 9. 3. 2022]. URL: https://attack.mitre.org/.

[20] What is a ddos attack ? [online]. [cit. 25. 11. 2021]. URL: https://bit.ly/3ELORuV.

[21] Russia must be stopped! help ukraine win! [online]. [cit. 8. 3. 2022]. URL: https://stop-russian-desinformation.near.page/.

[22] Denial-of-service attack. [online]. [cit. 22. 11. 2021]. URL: https://bit.ly/3m1yfI7.

[23] Classification of dos attacks. [online]. [cit. 9. 3. 2022]. URL: https://www.incibe-cert.es/en/blog/classification-dos-attacks.

[24] Denial-of-service attack. [online]. [cit. 9. 3. 2022]. URL: https://en.wikipedia.org/wiki/Denial-of-service_attack.

[25] Enrico Cambiaso, Gianluca Papaleo, Maurizio Aiello, and Giovanni Chiola. Designing and modeling the slow next dos attack. 06 2015. doi:10.1007/978-3-319-19713-5_22.

[26] Xiao ming LIU, Gong CHENG, Qi LI, and Miao ZHANG. A comparative study on flood dos and low-rate dos attacks. *The Journal of China Universities of Posts and Telecommunications*, 19:116–121, 2012. URL: https://bit.ly/3IJuvEY, doi:10.1016/S1005-8885(11)60458-5.

[27] Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. Taxonomy of slow dos attacks to web applications. volume 335, 10 2012. doi:10.1007/978-3-642-34135-9_20.

[28] Slowloris. [online]. [cit. 1. 12. 2021]. URL: https://bit.ly/3DSfay6.

[29] Vasilios A. Siris and Fotini Papagalou. Application of anomaly detection algorithms for detecting syn flooding attacks. In *GLOBECOM*, 2004.

[30] Enrico Cambiaso, Giovanni Chiola, and Maurizio Aiello. Introducing the slowdrop attack. *Computer Networks*, 150:234–249, 2019. URL: https://bit.ly/31QsuGk, doi:10.1016/j.comnet.2019.01.007.

[31] Enrico Cambiaso, Maurizio Aiello, Maurizio Mongelli, and Ivan Vaccari. Detection and classification of slow dos attacks targeting network servers. pages 1–7, 08 2020. doi:10.1145/3407023.3409198.

[32] Apache mpm common directives. [online]. [cit. 7. 12. 2021]. URL: https://bit.ly/3m2cuYJ.

[33] Ryan Frants. Apache tcp backlog. [online]. [cit. 7. 12. 2021]. URL: https://bit.ly/31Ltzzu.

[34] David Borman, Robert T. Braden, Van Jacobson, and Richard Scheffenegger. TCP Extensions for High Performance. RFC 7323, September 2014. URL: https://rfc-editor.org/rfc/rfc7323.txt, doi:10.17487/RFC7323.

[35] A comparative analysis of tcp tahoe, reno, new-reno, sack and vegas. [online]. [cit. 7. 12. 2021]. URL: https://bit.ly/3dJEMTm.

[36] Iptables(8) - linux man page. [online]. [cit. 9. 5. 2022]. URL: https://linux.die.net/man/8/iptables.

[37] How to reset iptables to the default settings. [online]. [cit. 8. 5. 2022]. URL: https://kerneltalks.com/virtualization/how-to-reset-iptables-to-default-settings/.

[38] Pavel Mazánek. Modelování a detekce útoku slowdrop. [online], 2020. [cit. 7. 12. 2021]. URL: http://hdl.handle.net/11012/189192.

[39] Requests: Http for humans. [online]. [cit. 21. 3. 2022]. URL: https://docs.python-requests.org/en/latest/.

[40] Urllib3 1.26.9. [online]. [cit. 23. 4. 2022]. URL: https://pypi.org/project/urllib3/.

[41] Socket – low-level networking interface. [online]. [cit. 6. 5. 2022]. URL: https://docs.python.org/3/library/socket.html.

[42] Vipin Jain. What is the difference between concurrency, parallelism and asynchronous methods? [online]. [cit. 29. 3. 2022]. URL: shorturl.at/jwNPU.

[43] Asyncio (superseded by async page). [online]. [cit. 29. 3. 2022]. URL: https://cheat.readthedocs.io/en/latest/python/asyncio.html.

[44] Globalinterpreterlock. [online]. [cit. 30. 3. 2022]. URL: https://wiki.python.org/moin/GlobalInterpreterLock.

[45] Giorgos Myrianthous. Multi-threading and multi-processing in python. [online]. [cit. 30. 3. 2022]. URL: https://towardsdatascience.com/multithreading-multiprocessing-python-180d0975ab29.

[46] Threading — thread-based parallelism. [online]. [cit. 30. 3. 2022]. URL: https://docs.python.org/3/library/threading.html.

[47] Jim Anderson. An intro to threading in python. [online]. [cit. 31. 3. 2022]. URL: https://realpython.com/intro-to-python-threading/.

[48] Threading – thread-based parallelism. [online]. [cit. 8. 5. 2022]. URL: https://docs.python.org/3/library/threading.html.

[49] Time – time access and conversions. [online]. [cit. 8. 5. 2022]. URL: https://docs.python.org/3/library/time.html.

[50] Logging – logging facility for python. [online]. [cit. 8. 5. 2022]. URL: https://docs.python.org/3/library/logging.html.

[51] Argparse – parser for command-line options, arguments and sub-commands. [online]. [cit. 8. 5. 2022]. URL: https://docs.python.org/3/library/argparse.html.

[52] Random – generate pseudo-random number. [online]. [cit. 9. 5. 2022]. URL: https://docs.python.org/3/library/random.html.

[53] Os – miscellaneous operating system interfaces. [online]. [cit. 8. 5. 2022]. URL: https://docs.python.org/3/library/os.html.

[54] Syslog. [online], 2001-. [cit. 25. 4. 2022]. URL: https://en.wikipedia.org/wiki/Syslog.

[55] Apache http server version 2.4. [online]. [cit. 5. 12. 2021]. URL: https://bit.ly/3dNzaYk.

[56] Jason Potter. Apache performance tuning: Mpm directives. [online]. [cit. 6. 12. 2021]. URL: https://bit.ly/3pTPTik.

[57] Apachectl - apache http server control interface. [online]. [cit. 6. 12. 2021]. URL: https://bit.ly/3rZcyMP.

[58] Securing apache 2 with modsecurity. [online]. [cit. 6. 12. 2021]. URL: https://bit.ly/3yodStW.

[59] Apache module mod_reqtimeout. [online]. [cit. 6. 12. 2021]. URL: https://bit.ly/3m2WB4t.

[60] Apache module mod_ratelimit. [online]. [cit. 13. 5. 2022]. URL: shorturl.at/bqBX1.

[61] Defend against dos & ddos on apache with mod_evasive. [online]. [cit. 13. 5. 2022]. URL: https://phoenixnap.com/kb/apache-mod-evasive.

[62] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems. *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, pages 12–26, 1999. [cit. 4. 4. 2022]. doi:10.1109/DISCEX.2000.821506.

[63] Nsl-kdd dataset. [online]. [cit. 4. 4. 2022]. URL: https://www.unb.ca/cic/datasets/nsl.html.

[64] Datasets. [online]. [cit. 19. 5. 2022]. URL: https://www.unb.ca/cic/datasets/index.html.

[65] Intrusion detection evaluation dataset (icxids2012). [online]. [cit. 4. 4. 2022]. URL: https://www.unb.ca/cic/datasets/ids.html.

[66] Cic dos dataset (2017). [online]. [cit. 13. 5. 2022]. URL: https://www.unb.ca/cic/datasets/dos-dataset.html.

[67] Cicflowmeter. [online]. [cit. 14. 5. 2022]. URL: https://github.com/ahlashkari/CICFlowMeter.

[68] Intrusion detection evaluation dataset (cic-ids2017). [online]. [cit. 13. 5. 2022]. URL: https://www.unb.ca/cic/datasets/ids-2017.html.

[69] Ddos evaluation dataset (cic-ddos2019). [online]. [cit. 13. 5. 2022]. URL: https://www.unb.ca/cic/datasets/ddos-2019.html.

[70] Cicflowmeter (formerly iscxflowmeter). [online]. [cit. 15. 5. 2022]. URL: https://www.unb.ca/cic/research/applications.html#CICFlowMeter.

[71] Python cicflowmeter. [online]. [cit. 15. 5. 2022]. URL: https://github.com/datthinh1801/cicflowmeter.

[72] Bricata. Layers of cybersecurity: Signature detection vs. network behavioral analysis. [online]. [cit. 12. 12. 2021]. URL: https://bit.ly/31WUuYC.

[73] What is a intrusion detection system? [online]. [cit. 12. 12. 2021]. URL: https://bit.ly/3dNz96I.

[74] What are signatures and how does signature-based detection work? [online]. [cit. 16. 5. 2022]. URL: https://home.sophos.com/en-us/security-news/2020/what-is-a-signature.

[75] Ids and ips characteristics. [online]. [cit. 19. 4. 2022]. URL: https://contenthub.netacad.com/netsec/11.1.4.

[76] Christian Callegari. Statistical approaches for network anomaly detection. [online]. [cit. 13. 12. 2021]. URL: https://bit.ly/3oY2y4F.

[77] Alberto Dainotti, Antonio Pescapè, and Giorgio Ventre. Wavelet-based detection of dos attacks. 11 2006. doi:10.1109/GLOCOM.2006.279.

[78] Supervised learning. [online]. [cit. 13. 12. 2021]. URL: https://ibm.co/3m0Y1w5.

[79] Unsupervised learning. [online]. [cit. 13. 12. 2021]. URL: https://ibm.co/3GUY1pN.

[80] Machine learning. [online]. [cit. 16. 5. 2022]. URL: https://en.wikipedia.org/wiki/Machine_learning.

[81] Decision tree learning. [online]. [cit. 17. 5. 2022]. URL: https://en.wikipedia.org/wiki/Decision_tree_learning.

[82] Random forest. [online]. [cit. 17. 5. 2022]. URL: https://en.wikipedia.org/wiki/Random_forest.

[83] Performance metrics for classification problems in machine learning. URL: shorturl.at/fjBJ8.

[84] Precision and recall. [online], 2001-. [cit. 20. 4. 2022]. URL: https://en.wikipedia.org/wiki/Precision_and_recall.

[85] Sensitivity and specificity. [online], 2001-. [cit. 20. 4. 2022]. URL: https://en.wikipedia.org/wiki/Sensitivity_and_specificity.

[86] Matthew Mayo. Frameworks for approaching the machine learning process. [online]. [cit. 16. 5. 2022]. URL: https://www.kdnuggets.com/2018/05/general-approaches-machine-learning-process.html.

[87] How to convert string categorical variables into numerical variables using label encoder in python. [online]. [cit. 17. 5. 2022]. URL: shorturl.at/czIY7.

[88] Feature selection. [online]. [cit. 17. 5. 2022]. URL: https://en.wikipedia.org/wiki/Feature_selection.

[89] 3.1. cross-validation: evaluating estimator performance. [online]. [cit. 18. 5. 2022]. URL: https://scikit-learn.org/stable/modules/cross_validation.html.

[90] Pandas.dataframe.corr. [online]. [cit. 18. 5. 2022]. URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html.

[91] 3.2. tuning the hyper-parameters of an estimator. [online]. [cit. 18. 5. 2022]. URL: https://scikit-learn.org/stable/modules/grid_search.html.

Symbols and abbreviations

BSD Berkeley Software Distribution
IoT Internet of Things
CIA Confidentiality, Integrity and Availability
IDS Intrusion Detection System
DoS Denial of Service
DNS Domain Name System
DHCP Dynamic Host Configuration protocol
ISO/OSI International Organization for Standardization – Open Systems Interconnection model
ISP Internet Service Provider
DDoS Distributed Denial-of-Service
MAC Medium Access Control
GIL Global Interpreter Lock
ICMP Internet Control Management Protocol
SDAs Slow DoS Attacks
TCP Transmission Control Protocol
TCP/IP Transmission Control Protocol/Internet Protocol
QoE Quality of Experience
IAT Inter-arrival time
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
ASCII American Standard Code for Information Interchange
SMTP Simple Mail Transfer Protocol
FTP File Transfer Protocol
I/O Input-output
SSH Secure Shell Protocol
MPM Multi-Processing Module
OWASP Open Web Application Security Project
API Application Programming Interface
URL Uniform Resource Locator
URI Uniform Resource Identifier
CWD Current Window
RTT Round Trip Time
IDS Intrusion Detection System
SIEM Security Information and Event Management
NIDS Network Intrusion Detection Systems
HIDS Host-based Intrusion Detection System

CVEs Common Vulnerabilities and Exposures systems
IPS Intrusion Prevention System
NTP Network Time Protocol
IANA Internet Assigned Numbers Authority
SVM Support Vector Machines
CNN Convolutional Neural Network
RTO Retransmission timeout
G**,** V**,** E Graph $G = (V, E)$, where V are vertices and E edges.
N Number of valid packets
A Number of attack packets

List of appendices

A Communication and flow parameters

Parameter	Type	Unit	Description
Common communication parameters			
IP_SRC	ip	–	Source IP address
IP_DST	ip	–	Destination IP address
TCP_DST	int	–	Destination TCP port
PROTO	int	–	Packet protocol number
TIMESTAMP	time	s	Packet or flow beginning timestamp
FLAG	int	–	Type of event (flow)
ATTACK_TYPE	string	–	Attack type if flagged as an attack
Inter-flow parameters			
General parameters			
FLOW_ID	flow id	–	Flow identifier
EXT_FLOW_ID	flow id	–	Extended Flow identifier
FLOW_DUR	time	s	Flow duration
FLOW_CONNS	int	–	Number of connections inside the flow
FLOW_CONNS_R	float	cns/s	Connection flow rate inside the flow
FWD_RCV_R	float	–	Ratio between sent and received packets in the flow
Volume parameters			
FWD_TOT_C	int	$pkts$	Total number of sent packets per flow
RCV_TOT_C	int	$pkts$	Total number of received packets per flow
FWD_FTH_TOT_R	int	$pkts/s$	Total packet throughput in the forwarding flow
RCV_FTH_TOT_R	int	$pkts/s$	Total packet throughput in the receiving flow
FWD_FTH_MAX_R	float	$pkts/s$	Maximal packet throughput in the forwarding flow
RCV_FTH_MAX_R	float	$pkts/s$	Maximal packet throughput in the receiving flow
FWD_FTH_MIN_R	float	$pkts/s$	Minimal packet throughput in the forwarding flow
RCV_FTH_MIN_R	float	$pkts/s$	Minimal packet throughput in the receiving flow
FWD_FTH_MEANR	float	$pkts/s$	Mean packet throughput in the forwarding flow
RCV_FTH_MEANR	float	$pkts/s$	Mean packet throughput in the receiving flow
FWD_FTH_STD_R	float	$pkts/s$	Standard deviation of the packet throughput in the forwarding flow
RCV_FTH_STD_R	float	$pkts/s$	Standard deviation of the packet throughput in the receiving flow
FWD_TOT_S	int	B	Total flow size of sent packets

RCV_TOT_S	int	*B*	Total flow size of received packets
FWD_MAX_S	int	*B*	Flow size of the biggest sent packet
FWD_MIN_S	int	*B*	Flow size of the smallest sent packet
FWD_MEAN_S	float	*B*	Flow mean size of the sent packets
FWD_STD_S	float	*B*	Flow standard deviation size of the sent packets
RCV_MAX_S	int	*B*	Flow size of the biggest received packet
RCV_MIN_S	int	*B*	Flow size of the smallest received packet
RCV_MEAN_S	float	*B*	Flow mean size of the received packets
RCV_STD_S	float	*B*	Flow standard deviation of the received packets
FWD_HDR_TOT_S	int	*B*	Flow total size of the sent headers length
RCV_HDR_TOT_S	int	*B*	Flow total size of the received headers length
FWD_HDR_MAX_S	int	*B*	Flow maximal size of the sent headers length
RCV_HDR_MAX_S	int	*B*	Flow maximal size of the received headers length
FWD_HDR_MIN_S	int	*B*	Flow minimal size of the sent headers length
RCV_HDR_MIN_S	int	*B*	Flow minimal size of the received headers length
FWD_HDR_MEANS	float	*B*	Flow mean size of the sent headers length
RCV_HDR_MEANS	float	*B*	Flow mean size of the received headers length
FWD_HDR_STD_S	float	*B*	Flow standard deviation size of the sent headers lengths
RCV_HDR_STD_S	float	*B*	Flow standard deviation size of the received headers lengths
FWD_INIT_S	int	*pkts*	Flow total number of sent packets in the initial window
FWD_INIT_C	int	*B*	Flow size of sent packets in the initial window
RCV_INIT_S	int	*pkts*	Flow total number of received packets in the initial window
RCV_INIT_C	int	*B*	Flow size of received packets in the initial window
FWD_TW_MAX_S	int	*B*	Forwarding flow maximal TCP window size value
RCV_TW_MAX_S	int	*B*	Receiving flow maximal TCP window size value
FWD_TW_MIN_S	int	*B*	Forwarding flow minimal TCP window size value
RCV_TW_MIN_S	int	*B*	Receiving flow minimal TCP window size value
FWD_TW_MEAN_S	float	*B*	Forwarding flow mean TCP window size value
RCV_TW_MEAN_S	float	*B*	Receiving flow mean TCP window size value
FWD_TW_STD_S	float	*B*	Forwarding flow standard deviation of TCP window size value
RCV_TW_STD_S	float	*B*	Receiving flow standard deviation of TCP window size value
FWD_WSF_MAX_S	int	*B*	Maximal value of the TCP window size scale

			factor in forwarding flow
RCV_WSF_MAX_S	int	B	Maximal value of the TCP window size scale factor in receiving flow
FWD_WSF_MIN_S	int	B	Minimal value of the TCP window size scale factor in forwarding flow
RCV_WSF_MIN_S	int	B	Minimal value of the TCP window size scale factor in receiving flow
FWD_WSF_MEANS	float	B	Mean value of the TCP window size scale factor in forwarding flow
RCV_WSF_MEANS	float	B	Mean value of the TCP window size scale factor in receiving flow
FWD_WSF_STD_S	float	B	Standard deviation value of the TCP window size scale factor in forwarding flow
RCV_WSF_STD_S	float	B	Standard deviation value of the TCP window size scale factor in receiving flow
FWD_MSS_MAX_S	int	B	Maximal value of maximum segment size in the forwarding flow
RCV_MSS_MAX_S	int	B	Maximal value of maximum segment size in the receiving flow
FWD_MSS_MIN_S	int	B	Minimal value of maximum segment size in the forwarding flow
RCV_MSS_MIN_S	int	B	Minimal value of maximum segment size in the receiving flow
FWD_MSS_MEANS	float	B	Mean value of maximum segment size in the forwarding flow
RCV_MSS_MEANS	float	B	Mean value of maximum segment size in the receiving flow
FWD_MSS_STD_S	float	B	Standard deviation value of maximum segment size in the forwarding flow
RCV_MSS_STD_S	float	B	Standard deviation value of maximum segment size in the receiving flow
FWD_BB_MAX_R	int	B/blk	Maximal bytes bulk rate in the forwarding flow
RCV_BB_MAX_R	int	B/blk	Maximal bytes bulk rate in the receiving flow
FWD_BB_MIN_R	int	B/blk	Minimal bytes bulk rate in the forwarding flow
RCV_BB_MIN_R	int	B/blk	Minimal bytes bulk rate in the receiving flow
FWD_BB_MEAN_R	float	B/blk	Mean bytes bulk rate in the forwarding flow
RCV_BB_MEAN_R	float	B/blk	Mean bytes bulk rate in the receiving flow
FWD_BB_STD_R	float	B/blk	Standard deviation of bytes bulk rate in the forwarding flow
RCV_BB_STD_R	float	B/blk	Standard deviation of bytes bulk rate

			in the receiving flow
FWD_PB_MAX_R	int	*ps/blk*	Maximal packets bulk rate in the forwarding flow
RCV_PB_MAX_R	int	*ps/blk*	Maximal packets bulk rate in the receiving flow
FWD_PB_MIN_R	int	*ps/blk*	Minimal packets bulk rate in the forwarding flow
RCV_PB_MIN_R	int	*ps/blk*	Minimal packets bulk rate in the receiving flow
FWD_PB_MEAN_R	float	*ps/blk*	Mean packets bulk rate in the forwarding flow
RCV_PB_MEAN_R	float	*ps/blk*	Mean packets bulk rate in the receiving flow
FWD_PB_STD_R	float	*ps/blk*	Standard deviation of packets bulk rate in the forwarding flow
RCV_PB_STD_R	float	*ps/blk*	Standard deviation of packets bulk rate in the receiving flow
FWD_PD_TOT_S	int	*B*	Total TCP payload size in the forwarding flow
RCV_PD_TOT_S	int	*B*	Total TCP payload size in the receiving flow
FWD_PD_MAX_S	int	*B*	Maximal TCP payload size in the forwarding flow
RCV_PD_MAX_S	int	*B*	Maximal TCP payload size in the receiving flow
FWD_PD_MIN_S	int	*B*	Minimal TCP payload size in the forwarding flow
RCV_PD_MIN_S	int	*B*	Minimal TCP payload size in the receiving flow
FWD_PD_MEAN_S	int	*B*	Mean TCP payload size in the forwarding flow
RCV_PD_MEAN_S	int	*B*	Mean TCP payload size in the receiving flow
FWD_PD_STD_S	int	*B*	Standard deviation of the TCP payload size in the forwarding flow
RCV_PD_STD_S	int	*B*	Standard deviation of the TCP payload size in the receiving flow
FWD_PLD_C	int	–	Number of sent packets with at least $1B$ payload in the flow
RCV_PLD_C	int	–	Number of received packets with at least $1B$ payload in the flow
Time parameters			
FWD_IAT_TOTAL	int	*s*	Flow total time between all sent packets
FWD_IAT_MAX	int	*s*	Flow maximal time between two sent packets
FWD_IAT_MIN	int	*s*	Flow minimal time between two sent packets
FWD_IAT_MEAN	float	*s*	Flow mean time between two sent packets
FWD_IAT_STD	float	*s*	Flow standard deviation of the time between two sent packets
RCV_IAT_TOTAL	int	*s*	Flow total time between all received packets
RCV_IAT_MAX	int	*s*	Flow maximal time between two received packets
RCV_IAT_MIN	int	*s*	Flow minimum time between two received packets
RCV_IAT_MEAN	float	*s*	Flow mean time between two received packets
RCV_IAT_STD	float	*s*	Flow standard deviation of the time between two received packets

ACT_MAX	time	*s*	Maximum time the flow was active before becoming idle
ACT_MIN	time	*s*	Minimum time the flow was active before becoming idle
ACT_MEAN	time	*s*	Mean time the flow was active before becoming idle
ACT_STD	time	*s*	Standard deviation time the flow was active before becoming idle
IDLE_MAX	time	*s*	Maximum time the flow was idle before becoming active
IDLE_MIN	time	*s*	Minimum time the flow was idle before becoming active
IDLE_MEAN	time	*s*	Mean time the flow was idle before becoming active
IDLE_STD	time	*s*	Standard deviation time the flow was idle before becoming active
Feature and TCP Analysis Parameters			
FWD_PSH_C	int	–	Number of PSH flags set in flow forwarding way
FWD_URG_C	int	–	Number of URG flags set in flow forwarding way
FWD_ACK_C	int	–	Number of ACK flags set in flow forwarding way
FWD_RST_C	int	–	Number of RST flags set in flow forwarding way
FWD_SYN_C	int	–	Number of SYN flags set in flow forwarding way
FWD_FIN_C	int	–	Number of FIN flags set in flow forwarding way
RCV_PSH_C	int	–	Number of PSH flags set in flow receiving way
RCV_URG_C	int	–	Number of URG flags set in flow receiving way
RCV_ACK_C	int	–	Number of ACK flags set in flow receiving way
RCV_RST_C	int	–	Number of RST flags set in flow receiving way
RCV_SYN_C	int	–	Number of SYN flags set in flow receiving way
RCV_FIN_C	int	–	Number of FIN flags set in flow receiving way
FWD_ACKUN_F	int	–	Number of packets that was not previously captured in the forwarding flow
RCV_ACKUN_F	int	–	Number of packets that was not previously captured in the receiving flow
FWD_PRVLST_F	int	–	Number of packets with bigger sequence number than expected in the forwarding flow
RCV_PRVLST_F	int	–	Number of packets with bigger sequence number than expected in the receiving flow
FWD_RTRSMN_F	int	–	Number of retransmitted packets in the forwarding flow
RCV_RTRSMN_F	int	–	Number of retranmitted packets

			in the receiving flow
FWD_DUPACK_F	int	–	Number of duplicate packets in the forwarding flow
RCV_DUPACK_F	int	–	Number of duplicate packets in the receiving flow
FWD_OOO_F	int	–	Number of packets with sequence number out of order in the forwarding flow
RCV_OOO_F	int	–	Number of packets with sequence number out of order in the receiving flow
FWD_KEEPAL_F	int	–	Number of packets forcing other part to send acknowledgement in the forwarding flow
RCV_KEEPAL_F	int	–	Number of packets forcing other part to send acknowledgement in the receiving flow
FWD_KALACK_F	int	–	Number of acknowledgement responses to `KeepAlive` packets in the forwarding flow
RCV_KALACK_F	int	–	Number of acknowledgement responses to `KeepAlive` packets in the receiving flow
FWD_ZEROW_F	int	–	Number of packets indicating impossibility of receiver getting other data in the forwarding flow
RCV_ZEROW_F	int	–	Number of packets indicating impossibility of receiver getting other data in the receiving flow
FWD_WINFLL_F	int	–	Number of packets indicating receiver's full buffer in the forwarding flow
RCV_WINFLL_F	int	–	Number of packets indicating receiver's full buffer in the receiving flow
FWD_WINUP_F	int	–	Number of packets where receiver indicates the free size of his buffer in the forwarding flow
RCV_WINUP_F	int	–	Number of packets where receiver indicates the free size of his buffer in the receiving flow
FWD_ZWINP_F	int	–	Number of testing packet if the receiver's zero window is in place in the forwarding flow
RCV_ZWINP_F	int	–	Number of testing packet if the receiver's zero window is in place in the receiving flow
FWD_ZWINPA_F	int	–	Number of packet acknowledging zero window probe in the forwarding flow
RCV_ZWINPA_F	int	–	Number of packet acknowledging zero window probe in the receiving flow
FWD_ZWINVIL_F	int	–	Number of packet where the sender ignores zero window condition in the forwarding flow
RCV_ZWINVIL_F	int	–	Number of packet where the sender ignores zero

			window condition in the receiving flow
Application layer parameters			
D_START_MAX	time	ms	Flow maximal time of the application layer data start forwarding
D_START_MIN	time	ms	Flow minimal time of the application layer data start forwarding
D_START_MEAN	time	ms	Flow mean time of the application layer data start forwarding
D_START_STD	time	ms	Flow standard deviation time of the application layer data start forwarding
D_REQ_MAX	time	ms	Flow maximal time for application request
D_REQ_MIN	time	ms	Flow minimal time for application request
D_REQ_MEAN	time	ms	Flow mean time for application request
D_REQ_STD	time	ms	Flow standard deviation time for application request
D_DELAY_MAX	time	ms	Flow maximal time to start receiving response
D_DELAY_MIN	time	ms	Flow minimal time to start receiving response
D_DELAY_MEAN	time	ms	Flow mean time to start receiving response
D_DELAY_STD	time	ms	Flow standard deviation time to start receiving the application response
D_RESP_MAX	time	ms	Flow maximal time to receive response
D_RESP_MIN	time	ms	Flow minimal time to receive response
D_RESP_MEAN	time	ms	Flow mean time to receive response
D_RESP_STD	time	ms	Flow standard deviation time to receive the application response
D_NEXT_MAX	time	ms	Flow maximal time from the end of the response to the beginning of the new one
D_NEXT_MIN	time	ms	Flow minimal time from the end of the response to the beginning of the new one
D_NEXT_MEAN	time	ms	Flow mean time from the end of the response to the beginning of the new one
D_NEXT_STD	time	ms	Flow standard deviation time from the end of the response to the beginning of the new one
Intra-flow parameters			
TCP_SRC	int	–	Source TCP port
TCP_WND_S	int	B	Window size value
TCP_WND_SSF	int	B	Window size scale factor value
TCP_HDRS	int	B	Size of TCP headers
PACKET_L	int	B	Packet size
TCP_PAYLOAD	int	B	TCP payload size

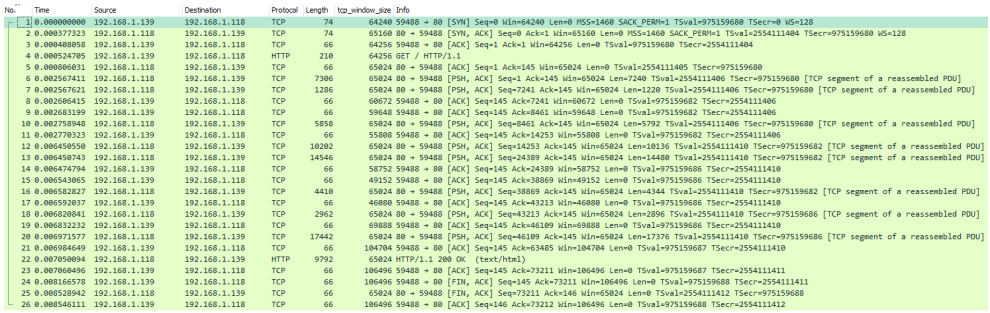
TCP_PSH_F	bool	–	PSH flag set
TCP_URG_F	bool	–	URG flag set
TCP_ACK_F	bool	–	ACK flag set
TCP_RST_F	bool	–	RST flag set
TCP_SYN_F	bool	–	SYN flag set
TCP_FIN_F	bool	–	FIN flag set
TCP_CWR_F	bool	–	CWR flag set
TCP_ECE_F	bool	–	ECE flag set

Tab. A.1: Flow parameters

B Examples of generated TCP traffic

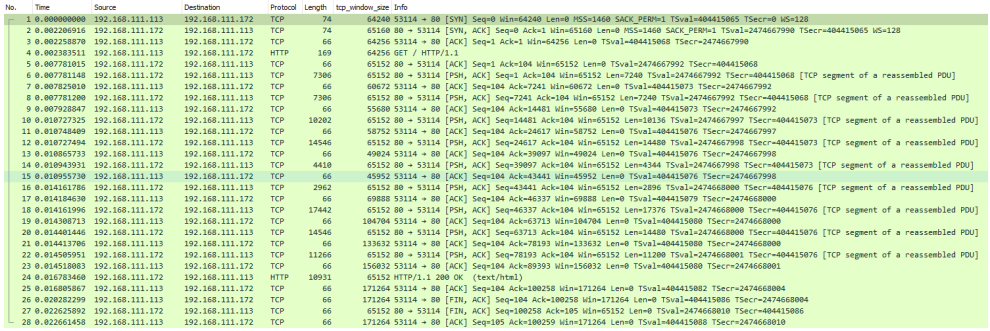
B.1 HTTP traffic using different modules

B.1.1 HTTP GET



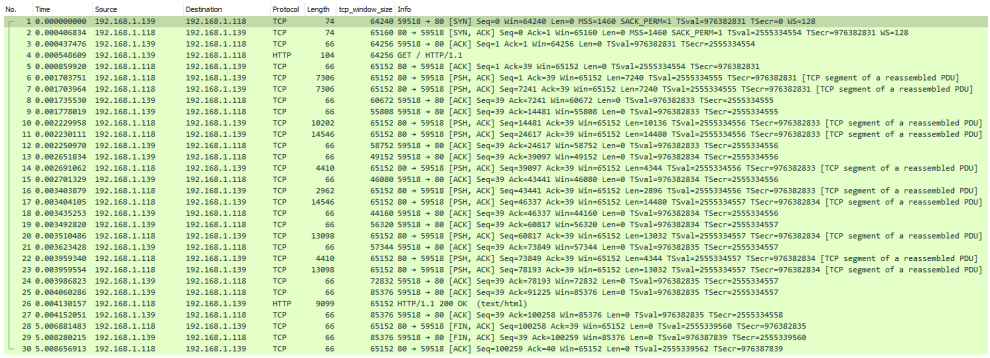
No.	Time	Source	Destination	Protocol	Length	tcp_window_size	Info
1	0.000000000	192.168.1.139	192.168.1.118	TCP	74	64240	59488 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=975159680 TSecr=0 WS=128
2	0.000377323	192.168.1.118	192.168.1.139	TCP	74	65160	80 → 59488 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2554111404 TSecr=975159680 WS=128
3	0.000408058	192.168.1.139	192.168.1.118	TCP	66	64256	59488 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=975159680 TSecr=2554111404
4	0.000524705	192.168.1.139	192.168.1.118	HTTP	210	64256	GET / HTTP/1.1
5	0.000806031	192.168.1.118	192.168.1.139	TCP	66	65024	80 → 59488 [ACK] Seq=1 Ack=145 Win=65024 Len=0 TSval=2554111405 TSecr=975159680
6	0.002567411	192.168.1.118	192.168.1.139	TCP	7306	65024	80 → 59488 [PSH, ACK] Seq=1 Ack=145 Win=65024 Len=7240 TSval=2554111406 TSecr=975159680 [TCP segment of a reassembled PDU]
7	0.002567621	192.168.1.118	192.168.1.139	TCP	1286	65024	80 → 59488 [PSH, ACK] Seq=7241 Ack=145 Win=65024 Len=1220 TSval=2554111406 TSecr=975159680 [TCP segment of a reassembled PDU]
8	0.002606415	192.168.1.139	192.168.1.118	TCP	66	60672	59488 → 80 [ACK] Seq=145 Ack=7241 Win=60672 Len=0 TSval=975159682 TSecr=2554111406
9	0.002683199	192.168.1.139	192.168.1.118	TCP	66	59648	59488 → 80 [ACK] Seq=145 Ack=8461 Win=59648 Len=0 TSval=975159682 TSecr=2554111406
10	0.002758948	192.168.1.118	192.168.1.139	TCP	5858	65024	80 → 59488 [PSH, ACK] Seq=8461 Ack=145 Win=65024 Len=5792 TSval=2554111406 TSecr=975159680 [TCP segment of a reassembled PDU]
11	0.002770323	192.168.1.139	192.168.1.118	TCP	66	55808	59488 → 80 [ACK] Seq=145 Ack=14253 Win=55808 Len=0 TSval=975159682 TSecr=2554111406
12	0.006450550	192.168.1.118	192.168.1.139	TCP	10202	65024	80 → 59488 [PSH, ACK] Seq=14253 Ack=145 Win=65024 Len=10136 TSval=2554111410 TSecr=975159682 [TCP segment of a reassembled PDU]
13	0.006450743	192.168.1.118	192.168.1.139	TCP	14546	65024	80 → 59488 [PSH, ACK] Seq=24389 Ack=145 Win=65024 Len=14480 TSval=2554111410 TSecr=975159682 [TCP segment of a reassembled PDU]
14	0.006474794	192.168.1.139	192.168.1.118	TCP	66	58752	59488 → 80 [ACK] Seq=145 Ack=24389 Win=58752 Len=0 TSval=975159686 TSecr=2554111410
15	0.006543065	192.168.1.139	192.168.1.118	TCP	66	49152	59488 → 80 [ACK] Seq=145 Ack=38869 Win=49152 Len=0 TSval=975159686 TSecr=2554111410
16	0.006582827	192.168.1.118	192.168.1.139	TCP	4410	65024	80 → 59488 [PSH, ACK] Seq=38869 Ack=145 Win=65024 Len=4344 TSval=2554111410 TSecr=975159682 [TCP segment of a reassembled PDU]
17	0.006592037	192.168.1.139	192.168.1.118	TCP	66	46080	59488 → 80 [ACK] Seq=145 Ack=43213 Win=46080 Len=0 TSval=975159686 TSecr=2554111410
18	0.006820841	192.168.1.118	192.168.1.139	TCP	2962	65024	80 → 59488 [PSH, ACK] Seq=43213 Ack=145 Win=65024 Len=2896 TSval=2554111410 TSecr=975159686 [TCP segment of a reassembled PDU]
19	0.006832232	192.168.1.139	192.168.1.118	TCP	66	69888	59488 → 80 [ACK] Seq=145 Ack=46109 Win=69888 Len=0 TSval=975159686 TSecr=2554111410
20	0.006971577	192.168.1.118	192.168.1.139	TCP	17442	65024	80 → 59488 [PSH, ACK] Seq=46109 Ack=145 Win=65024 Len=17376 TSval=2554111410 TSecr=975159686 [TCP segment of a reassembled PDU]
21	0.006984649	192.168.1.139	192.168.1.118	TCP	66	104704	59488 → 80 [ACK] Seq=145 Ack=63485 Win=104704 Len=0 TSval=975159687 TSecr=2554111410
22	0.007050094	192.168.1.118	192.168.1.139	HTTP	9792	65024	HTTP/1.1 200 OK (text/html)
23	0.007060496	192.168.1.139	192.168.1.118	TCP	66	106496	59488 → 80 [ACK] Seq=145 Ack=73211 Win=106496 Len=0 TSval=975159687 TSecr=2554111411
24	0.008166578	192.168.1.139	192.168.1.118	TCP	66	106496	59488 → 80 [FIN, ACK] Seq=145 Ack=73211 Win=106496 Len=0 TSval=975159688 TSecr=2554111411
25	0.008528942	192.168.1.118	192.168.1.139	TCP	66	65024	80 → 59488 [FIN, ACK] Seq=73211 Ack=146 Win=65024 Len=0 TSval=2554111412 TSecr=975159688
26	0.008546111	192.168.1.139	192.168.1.118	TCP	66	106496	59488 → 80 [ACK] Seq=146 Ack=73212 Win=106496 Len=0 TSval=975159688 TSecr=2554111412

Fig. B.1: Example of HTTP GET traffic using module `requests`



No.	Time	Source	Destination	Protocol	Length	tcp_window_size	Info
1	0.000000000	192.168.111.113	192.168.111.172	TCP	74	64240	53114 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=404415065 TSecr=0 WS=128
2	0.002206916	192.168.111.172	192.168.111.113	TCP	74	65160	80 → 53114 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2474667990 TSecr=404415065 WS=128
3	0.002258870	192.168.111.113	192.168.111.172	TCP	66	64256	53114 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=404415068 TSecr=2474667990
4	0.002383511	192.168.111.113	192.168.111.172	HTTP	169	64256	GET / HTTP/1.1
5	0.007781015	192.168.111.172	192.168.111.113	TCP	66	65152	80 → 53114 [ACK] Seq=1 Ack=104 Win=65152 Len=0 TSval=2474667992 TSecr=404415068
6	0.007781148	192.168.111.172	192.168.111.113	TCP	7306	65152	80 → 53114 [PSH, ACK] Seq=1 Ack=104 Win=65152 Len=7240 TSval=2474667992 TSecr=404415068 [TCP segment of a reassembled PDU]
7	0.007825010	192.168.111.113	192.168.111.172	TCP	66	60672	53114 → 80 [ACK] Seq=104 Ack=7241 Win=60672 Len=0 TSval=404415073 TSecr=2474667992
8	0.007781200	192.168.111.172	192.168.111.113	TCP	7306	65152	80 → 53114 [PSH, ACK] Seq=7241 Ack=104 Win=65152 Len=7240 TSval=2474667992 TSecr=404415068 [TCP segment of a reassembled PDU]
9	0.007928847	192.168.111.113	192.168.111.172	TCP	66	55680	53114 → 80 [ACK] Seq=104 Ack=14481 Win=55680 Len=0 TSval=404415073 TSecr=2474667992
10	0.010727325	192.168.111.172	192.168.111.113	TCP	10202	65152	80 → 53114 [PSH, ACK] Seq=14481 Ack=104 Win=65152 Len=10136 TSval=2474667997 TSecr=404415073 [TCP segment of a reassembled PDU]
11	0.010748409	192.168.111.113	192.168.111.172	TCP	66	58752	53114 → 80 [ACK] Seq=104 Ack=24617 Win=58752 Len=0 TSval=404415076 TSecr=2474667997
12	0.010727494	192.168.111.172	192.168.111.113	TCP	14546	65152	80 → 53114 [PSH, ACK] Seq=24617 Ack=104 Win=65152 Len=14480 TSval=2474667998 TSecr=404415073 [TCP segment of a reassembled PDU]
13	0.010865733	192.168.111.113	192.168.111.172	TCP	66	49024	53114 → 80 [ACK] Seq=104 Ack=39097 Win=49024 Len=0 TSval=404415076 TSecr=2474667998
14	0.010943931	192.168.111.172	192.168.111.113	TCP	4410	65152	80 → 53114 [PSH, ACK] Seq=39097 Ack=104 Win=65152 Len=4344 TSval=2474667998 TSecr=404415073 [TCP segment of a reassembled PDU]
15	0.010955730	192.168.111.113	192.168.111.172	TCP	66	45952	53114 → 80 [ACK] Seq=104 Ack=43441 Win=45952 Len=0 TSval=404415076 TSecr=2474667998
16	0.014161786	192.168.111.172	192.168.111.113	TCP	2962	65152	80 → 53114 [PSH, ACK] Seq=43441 Ack=104 Win=65152 Len=2896 TSval=2474668000 TSecr=404415076 [TCP segment of a reassembled PDU]
17	0.014184630	192.168.111.113	192.168.111.172	TCP	66	69888	53114 → 80 [ACK] Seq=104 Ack=46337 Win=69888 Len=0 TSval=404415079 TSecr=2474668000
18	0.014161996	192.168.111.172	192.168.111.113	TCP	17442	65152	80 → 53114 [PSH, ACK] Seq=46337 Ack=104 Win=65152 Len=17376 TSval=2474668000 TSecr=404415076 [TCP segment of a reassembled PDU]
19	0.014308713	192.168.111.113	192.168.111.172	TCP	66	104704	53114 → 80 [ACK] Seq=104 Ack=63713 Win=104704 Len=0 TSval=404415080 TSecr=2474668000
20	0.014401446	192.168.111.172	192.168.111.113	TCP	14546	65152	80 → 53114 [PSH, ACK] Seq=63713 Ack=104 Win=65152 Len=14480 TSval=2474668000 TSecr=404415076 [TCP segment of a reassembled PDU]
21	0.014413706	192.168.111.113	192.168.111.172	TCP	66	133632	53114 → 80 [ACK] Seq=104 Ack=78193 Win=133632 Len=0 TSval=404415080 TSecr=2474668000
22	0.014505951	192.168.111.172	192.168.111.113	TCP	11266	65152	80 → 53114 [PSH, ACK] Seq=78193 Ack=104 Win=65152 Len=11200 TSval=2474668001 TSecr=404415076 [TCP segment of a reassembled PDU]
23	0.014518083	192.168.111.113	192.168.111.172	TCP	66	156032	53114 → 80 [ACK] Seq=104 Ack=89393 Win=156032 Len=0 TSval=404415080 TSecr=2474668001
24	0.016783460	192.168.111.172	192.168.111.113	HTTP	10931	65152	HTTP/1.1 200 OK (text/html)
25	0.016805867	192.168.111.113	192.168.111.172	TCP	66	171264	53114 → 80 [ACK] Seq=104 Ack=100258 Win=171264 Len=0 TSval=404415082 TSecr=2474668004
26	0.020282299	192.168.111.113	192.168.111.172	TCP	66	171264	53114 → 80 [FIN, ACK] Seq=104 Ack=100258 Win=171264 Len=0 TSval=404415086 TSecr=2474668004
27	0.022625892	192.168.111.172	192.168.111.113	TCP	66	65152	80 → 53114 [FIN, ACK] Seq=100258 Ack=105 Win=65152 Len=0 TSval=2474668010 TSecr=404415086
28	0.022661458	192.168.111.113	192.168.111.172	TCP	66	171264	53114 → 80 [ACK] Seq=105 Ack=100259 Win=171264 Len=0 TSval=404415088 TSecr=2474668010

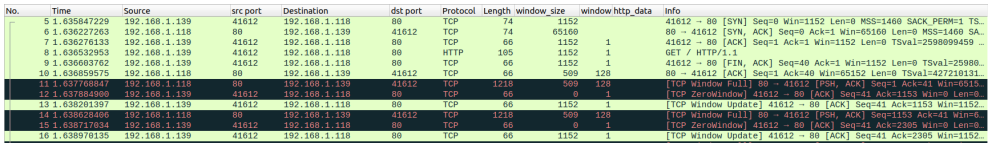
Fig. B.2: Example of HTTP GET traffic using module `urllib3`



No.	Time	Source	Destination	Protocol	Length	tcp_window_size	Info
1	0.000000000	192.168.1.139	192.168.1.118	TCP	74	64240	59518 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=976382831 TSecr=0 WS=128
2	0.000406834	192.168.1.118	192.168.1.139	TCP	74	65160	80 → 59518 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2555334554 TSecr=976382831 WS=128
3	0.000437476	192.168.1.139	192.168.1.118	TCP	66	64256	59518 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=976382831 TSecr=2555334554
4	0.000548609	192.168.1.139	192.168.1.118	HTTP	104	64256	GET / HTTP/1.1
5	0.000859920	192.168.1.118	192.168.1.139	TCP	66	65152	80 → 59518 [ACK] Seq=1 Ack=39 Win=65152 Len=0 TSval=2555334554 TSecr=976382831
6	0.001703751	192.168.1.118	192.168.1.139	TCP	7306	65152	80 → 59518 [PSH, ACK] Seq=1 Ack=39 Win=65152 Len=7240 TSval=2555334555 TSecr=976382831 [TCP segment of a reassembled PDU]
7	0.001703964	192.168.1.118	192.168.1.139	TCP	7306	65152	80 → 59518 [PSH, ACK] Seq=7241 Ack=39 Win=65152 Len=7240 TSval=2555334555 TSecr=976382831 [TCP segment of a reassembled PDU]
8	0.001735530	192.168.1.139	192.168.1.118	TCP	66	60672	59518 → 80 [ACK] Seq=39 Ack=7241 Win=60672 Len=0 TSval=976382833 TSecr=2555334555
9	0.001778019	192.168.1.139	192.168.1.118	TCP	66	55808	59518 → 80 [ACK] Seq=39 Ack=14481 Win=55808 Len=0 TSval=976382833 TSecr=2555334555
10	0.002229958	192.168.1.118	192.168.1.139	TCP	10202	65152	80 → 59518 [PSH, ACK] Seq=14481 Ack=39 Win=65152 Len=10136 TSval=2555334556 TSecr=976382833 [TCP segment of a reassembled PDU]
11	0.002230111	192.168.1.118	192.168.1.139	TCP	14546	65152	80 → 59518 [PSH, ACK] Seq=24617 Ack=39 Win=65152 Len=14480 TSval=2555334556 TSecr=976382833 [TCP segment of a reassembled PDU]
12	0.002250970	192.168.1.139	192.168.1.118	TCP	66	58752	59518 → 80 [ACK] Seq=39 Ack=24617 Win=58752 Len=0 TSval=976382833 TSecr=2555334556
13	0.002651834	192.168.1.139	192.168.1.118	TCP	66	49152	59518 → 80 [ACK] Seq=39 Ack=39097 Win=49152 Len=0 TSval=976382834 TSecr=2555334556
14	0.002691862	192.168.1.118	192.168.1.139	TCP	4410	65152	80 → 59518 [PSH, ACK] Seq=39097 Ack=39 Win=65152 Len=4344 TSval=2555334556 TSecr=976382833 [TCP segment of a reassembled PDU]
15	0.002701329	192.168.1.139	192.168.1.118	TCP	66	46080	59518 → 80 [ACK] Seq=39 Ack=43441 Win=46080 Len=0 TSval=976382834 TSecr=2555334556
16	0.003403879	192.168.1.118	192.168.1.139	TCP	2962	65152	80 → 59518 [PSH, ACK] Seq=43441 Ack=39 Win=65152 Len=2896 TSval=2555334556 TSecr=976382833 [TCP segment of a reassembled PDU]
17	0.003404105	192.168.1.118	192.168.1.139	TCP	14546	65152	80 → 59518 [PSH, ACK] Seq=46337 Ack=39 Win=65152 Len=14480 TSval=2555334557 TSecr=976382834 [TCP segment of a reassembled PDU]
18	0.003435253	192.168.1.139	192.168.1.118	TCP	66	44160	59518 → 80 [ACK] Seq=39 Ack=46337 Win=44160 Len=0 TSval=976382834 TSecr=2555334556
19	0.003492820	192.168.1.139	192.168.1.118	TCP	66	56320	59518 → 80 [ACK] Seq=39 Ack=60817 Win=56320 Len=0 TSval=976382834 TSecr=2555334557
20	0.003510486	192.168.1.118	192.168.1.139	TCP	13098	65152	80 → 59518 [PSH, ACK] Seq=60817 Ack=39 Win=65152 Len=13032 TSval=2555334557 TSecr=976382834 [TCP segment of a reassembled PDU]
21	0.003623428	192.168.1.139	192.168.1.118	TCP	66	57344	59518 → 80 [ACK] Seq=39 Ack=73849 Win=57344 Len=0 TSval=976382835 TSecr=2555334557
22	0.003959340	192.168.1.118	192.168.1.139	TCP	4410	65152	80 → 59518 [PSH, ACK] Seq=73849 Ack=39 Win=65152 Len=4344 TSval=2555334557 TSecr=976382834 [TCP segment of a reassembled PDU]
23	0.003959554	192.168.1.118	192.168.1.139	TCP	13098	65152	80 → 59518 [PSH, ACK] Seq=78193 Ack=39 Win=65152 Len=13032 TSval=2555334557 TSecr=976382834 [TCP segment of a reassembled PDU]
24	0.003986823	192.168.1.139	192.168.1.118	TCP	66	72832	59518 → 80 [ACK] Seq=39 Ack=78193 Win=72832 Len=0 TSval=976382835 TSecr=2555334557
25	0.004060286	192.168.1.139	192.168.1.118	TCP	66	85376	59518 → 80 [ACK] Seq=39 Ack=91225 Win=85376 Len=0 TSval=976382835 TSecr=2555334557
26	0.004130157	192.168.1.118	192.168.1.139	HTTP	9099	65152	HTTP/1.1 200 OK (text/html)
27	0.004152051	192.168.1.139	192.168.1.118	TCP	66	85376	59518 → 80 [ACK] Seq=39 Ack=100258 Win=85376 Len=0 TSval=976382835 TSecr=2555334558
28	5.006881483	192.168.1.118	192.168.1.139	TCP	66	65152	80 → 59518 [FIN, ACK] Seq=100258 Ack=39 Win=65152 Len=0 TSval=2555339560 TSecr=976382835
29	5.008280215	192.168.1.139	192.168.1.118	TCP	66	85376	59518 → 80 [FIN, ACK] Seq=39 Ack=100259 Win=85376 Len=0 TSval=976387839 TSecr=2555339560
30	5.008656913	192.168.1.118	192.168.1.139	TCP	66	65152	80 → 59518 [ACK] Seq=100259 Ack=40 Win=65152 Len=0 TSval=2555339562 TSecr=976387839

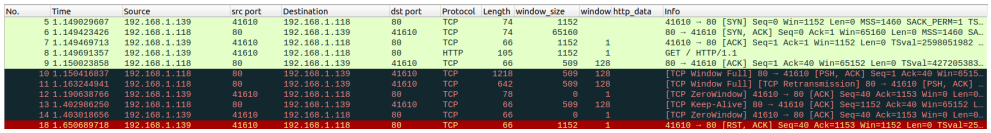
Fig. B.3: Example of HTTP GET traffic using module `socket`

B.2 Connection closures



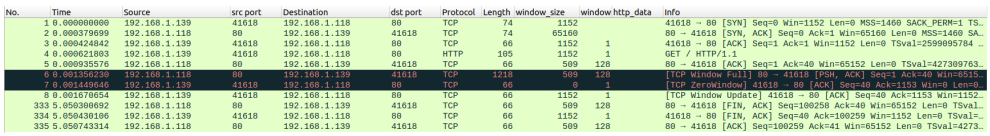
No.	Time	Source	src port	Destination	dst port	Protocol	Length	window_size	window	http_data	Info
5	1.635847229	192.168.1.139	41612	192.168.1.118	80	TCP	74	1152			41612 → 80 [SYN] Seq=0 Win=1152 Len=0 MSS=1460 SACK_PERM=1 TS…
6	1.636227263	192.168.1.118	80	192.168.1.139	41612	TCP	74	65160			80 → 41612 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
7	1.636276133	192.168.1.139	41612	192.168.1.118	80	TCP	66	1152	1		41612 → 80 [ACK] Seq=1 Ack=1 Win=1152 Len=0 TSval=2598099459 …
8	1.636532953	192.168.1.139	41612	192.168.1.118	80	HTTP	105	1152	1		GET / HTTP/1.1
9	1.636603762	192.168.1.139	41612	192.168.1.118	80	TCP	66	1152	1		41612 → 80 [FIN, ACK] Seq=40 Ack=1 Win=1152 Len=0 TSval=25980…
10	1.636859575	192.168.1.118	80	192.168.1.139	41612	TCP	66	509	128		80 → 41612 [ACK] Seq=1 Ack=40 Win=65152 Len=0 TSval=427210131…
11	1.637768847	192.168.1.118	80	192.168.1.139	41612	TCP	1218	509	128		[TCP Window Full] 80 → 41612 [PSH, ACK] Seq=1 Ack=41 Win=6515…
12	1.637804900	192.168.1.139	41612	192.168.1.118	80	TCP	66	0	1		[TCP ZeroWindow] 41612 → 80 [ACK] Seq=41 Ack=1153 Win=0 Len=0…
13	1.638201397	192.168.1.139	41612	192.168.1.118	80	TCP	66	1152	1		[TCP Window Update] 41612 → 80 [ACK] Seq=41 Ack=1153 Win=1152…
14	1.638628406	192.168.1.118	80	192.168.1.139	41612	TCP	1218	509	128		[TCP Window Full] 80 → 41612 [PSH, ACK] Seq=1153 Ack=41 Win=6…
15	1.638717034	192.168.1.139	41612	192.168.1.118	80	TCP	66	0	1		[TCP ZeroWindow] 41612 → 80 [ACK] Seq=41 Ack=2305 Win=0 Len=0…
16	1.638970135	192.168.1.139	41612	192.168.1.118	80	TCP	66	1152	1		[TCP Window Update] 41612 → 80 [ACK] Seq=41 Ack=2305 Win=1152…

Fig. B.4: Example of one-side standard client closure



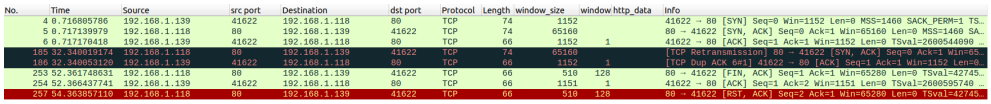
No.	Time	Source	src port	Destination	dst port	Protocol	Length	window_size	window	http_data	Info
5	1.149029607	192.168.1.139	41610	192.168.1.118	80	TCP	74	1152			41610 → 80 [SYN] Seq=0 Win=1152 Len=0 MSS=1460 SACK_PERM=1 TS…
6	1.149423426	192.168.1.118	80	192.168.1.139	41610	TCP	74	65160			80 → 41610 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
7	1.149469713	192.168.1.139	41610	192.168.1.118	80	TCP	66	1152	1		41610 → 80 [ACK] Seq=1 Ack=1 Win=1152 Len=0 TSval=2598051982 …
8	1.149691357	192.168.1.139	41610	192.168.1.118	80	HTTP	105	1152	1		GET / HTTP/1.1
9	1.150023858	192.168.1.118	80	192.168.1.139	41610	TCP	66	509	128		80 → 41610 [ACK] Seq=1 Ack=40 Win=65152 Len=0 TSval=427205383…
10	1.150416837	192.168.1.118	80	192.168.1.139	41610	TCP	1218	509	128		[TCP Window Full] 80 → 41610 [PSH, ACK] Seq=1 Ack=40 Win=6515…
11	1.163244941	192.168.1.118	80	192.168.1.139	41610	TCP	642	509	128		[TCP Window Full] [TCP Retransmission] 80 → 41610 [PSH, ACK] …
12	1.190638766	192.168.1.139	41610	192.168.1.118	80	TCP	78	0	1		[TCP ZeroWindow] 41610 → 80 [ACK] Seq=40 Ack=1153 Win=0 Len=0…
13	1.402986256	192.168.1.118	80	192.168.1.139	41610	TCP	66	509	128		[TCP Keep-Alive] 80 → 41610 [ACK] Seq=1152 Ack=40 Win=65152 L…
14	1.403018656	192.168.1.139	41610	192.168.1.118	80	TCP	66	0	1		[TCP ZeroWindow] 41610 → 80 [ACK] Seq=40 Ack=1153 Win=0 Len=0…
18	1.650689718	192.168.1.139	41610	192.168.1.118	80	TCP	66	1152	1		41610 → 80 [RST, ACK] Seq=40 Ack=1153 Win=1152 Len=0 TSval=25…

Fig. B.5: Example of forced client closure



No.	Time	Source	src port	Destination	dst port	Protocol	Length	window_size	window	http_data	Info
1	0.000000000	192.168.1.139	41618	192.168.1.118	80	TCP	74	1152			41618 → 80 [SYN] Seq=0 Win=1152 Len=0 MSS=1460 SACK_PERM=1 TS…
2	0.000379699	192.168.1.118	80	192.168.1.139	41618	TCP	74	65160			80 → 41618 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
3	0.000424842	192.168.1.139	41618	192.168.1.118	80	TCP	66	1152	1		41618 → 80 [ACK] Seq=1 Ack=1 Win=1152 Len=0 TSval=2599095784 …
4	0.000621803	192.168.1.139	41618	192.168.1.118	80	HTTP	105	1152	1		GET / HTTP/1.1
5	0.000935576	192.168.1.118	80	192.168.1.139	41618	TCP	66	509	128		80 → 41618 [ACK] Seq=1 Ack=40 Win=65152 Len=0 TSval=427309763…
6	0.001356230	192.168.1.118	80	192.168.1.139	41618	TCP	1218	509	128		[TCP Window Full] 80 → 41618 [PSH, ACK] Seq=1 Ack=40 Win=6515…
7	0.001449646	192.168.1.139	41618	192.168.1.118	80	TCP	66	0	1		[TCP ZeroWindow] 41618 → 80 [ACK] Seq=40 Ack=1153 Win=0 Len=0…
8	0.001670654	192.168.1.139	41618	192.168.1.118	80	TCP	66	1152	1		[TCP Window Update] 41618 → 80 [ACK] Seq=40 Ack=1153 Win=1152…
333	5.050300692	192.168.1.118	80	192.168.1.139	41618	TCP	66	509	128		80 → 41618 [FIN, ACK] Seq=100258 Ack=40 Win=65152 Len=0 TSval…
334	5.050430106	192.168.1.139	41618	192.168.1.118	80	TCP	66	1152	1		41618 → 80 [FIN, ACK] Seq=40 Ack=100259 Win=1152 Len=0 TSval=…
335	5.050743314	192.168.1.118	80	192.168.1.139	41618	TCP	66	509	128		80 → 41618 [ACK] Seq=100259 Ack=41 Win=65152 Len=0 TSval=4273…

Fig. B.6: Example of server timeout closure with `RST` flag



No.	Time	Source	src port	Destination	dst port	Protocol	Length	window_size	window	http_data	Info
4	0.716805786	192.168.1.139	41622	192.168.1.118	80	TCP	74	1152			41622 → 80 [SYN] Seq=0 Win=1152 Len=0 MSS=1460 SACK_PERM=1 TS…
5	0.717139979	192.168.1.118	80	192.168.1.139	41622	TCP	74	65160			80 → 41622 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA…
6	0.717170418	192.168.1.139	41622	192.168.1.118	80	TCP	66	1152	1		41622 → 80 [ACK] Seq=1 Ack=1 Win=1152 Len=0 TSval=2600544090 …
185	32.340019174	192.168.1.118	80	192.168.1.139	41622	TCP	74	65160			[TCP Retransmission] 80 → 41622 [SYN, ACK] Seq=0 Ack=1 Win=65…
186	32.340053120	192.168.1.139	41622	192.168.1.118	80	TCP	66	1152	1		[TCP Dup ACK 6#1] 41622 → 80 [ACK] Seq=1 Ack=1 Win=1152 Len=0…
253	52.361748631	192.168.1.118	80	192.168.1.139	41622	TCP	66	510	128		80 → 41622 [FIN, ACK] Seq=1 Ack=1 Win=65280 Len=0 TSval=42745…
254	52.366437741	192.168.1.139	41622	192.168.1.118	80	TCP	66	1151	1		41622 → 80 [ACK] Seq=1 Ack=2 Win=1151 Len=0 TSval=2600595740 …
257	54.363857110	192.168.1.118	80	192.168.1.139	41622	TCP	66	510	128		80 → 41622 [RST, ACK] Seq=2 Ack=1 Win=65280 Len=0 TSval=42745…

Fig. B.7: Example of server timeout closure with `RST` flag

C Slow DoS Attacks Generator

C.1 Generator flowcharts

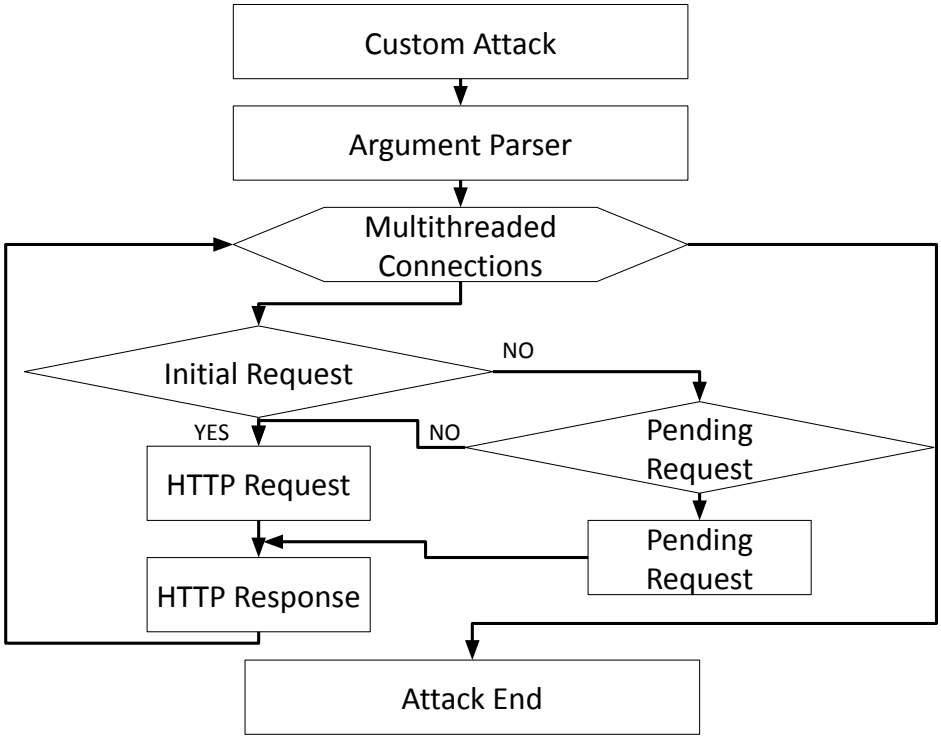


Fig. C.1: Flowchart of Custom Slow DoS Attack

C.2 Generator arguments

Common arguments:

- Positional arguments:
  - `TARGET` – URL or IP address of the victim
- Optional arguments:
  - `-p <int>` – TCP port of target application (default: 80)
  - `-c <int>` – number of TCP connections (default: $1\,conn$)
  - `-d <float>` – duration of the attack (default: $60\,s$)
  - `-open <float>` – duration of connected TCP socket (default: $0\,s$)
  - `-rec` – option to reconnect closed TCP connections (default: $False$)
  - `-close` – option to close connection if no data are incoming (default: $False$)

  - `-tdelay <float>` – delay between thread creation, if negative it chooses random value (default: 0.0)
  - `-l` – number of arguments stands for the logging level (default: 0 – error logging level)
  - `-log <string>` – logging type set as file logging, (default: *cmdline* – default bash logging)

Slow DoS Attack generator modes:

- `slow_read`
  - `-r <int>` – size of receiver buffer, TCP window size (default: 24)
- `slow_drop`
  - `-D <float>` – response drop rate (default: 0.6)
- `slow_next`
  - `-k <float>` – time interval between two consecutive requests (default: $4.5\,s$)
- `custom`
  - `-r <int>` – size of receiver buffer, TCP window size (default: -1 – not set)
  - `-rt <float>` – receiving chunk time interval, time interval between receiving data (default: $0.0\,s$)
  - `-s <int>` – size of sender buffer (default: -1 – not set)
  - `-st <float>` – sending chunk interval (default: $0.0\,s$)
  - `-dstart <float>` – value of Δ_{start} parameter (default: $0.0\,s$)
  - `-dreq <float>` – value of Δ_{req} parameter (default: $0.0\,s$)
  - `-ddelay <float>` – value of Δ_{delay} parameter (default: $0.0\,s$)
  - `-dresp <float>` – value of Δ_{resp} parameter (default: $0.0\,s$)
  - `-dnext <float>` – value of Δ_{next} parameter (default: $0.0\,s$)
  - `-http_request <string>` – custom http request with structure (`METHOD REQUEST-URL HTTP-VERSION CRLF`) (default: $None$)
  - `-http_headers <string>` – custom http request headers (default: $None$)
  - `-http_data <string>` – custom http request data (default: $None$)
  - `-pending_request <string>` – custom http pending request (default: $None$)
  - `-pending_headers <string>` – custom http pending headers (default: $None$)
  - `-pending_data <string>` – custom http pending data (default: $None$)
  - `-D <float>` – response drop rate (default: 0.0)
  - `-k <float>` – time interval between two consecutive requests (default: $0.0\,s$)

C.3 UML diagram

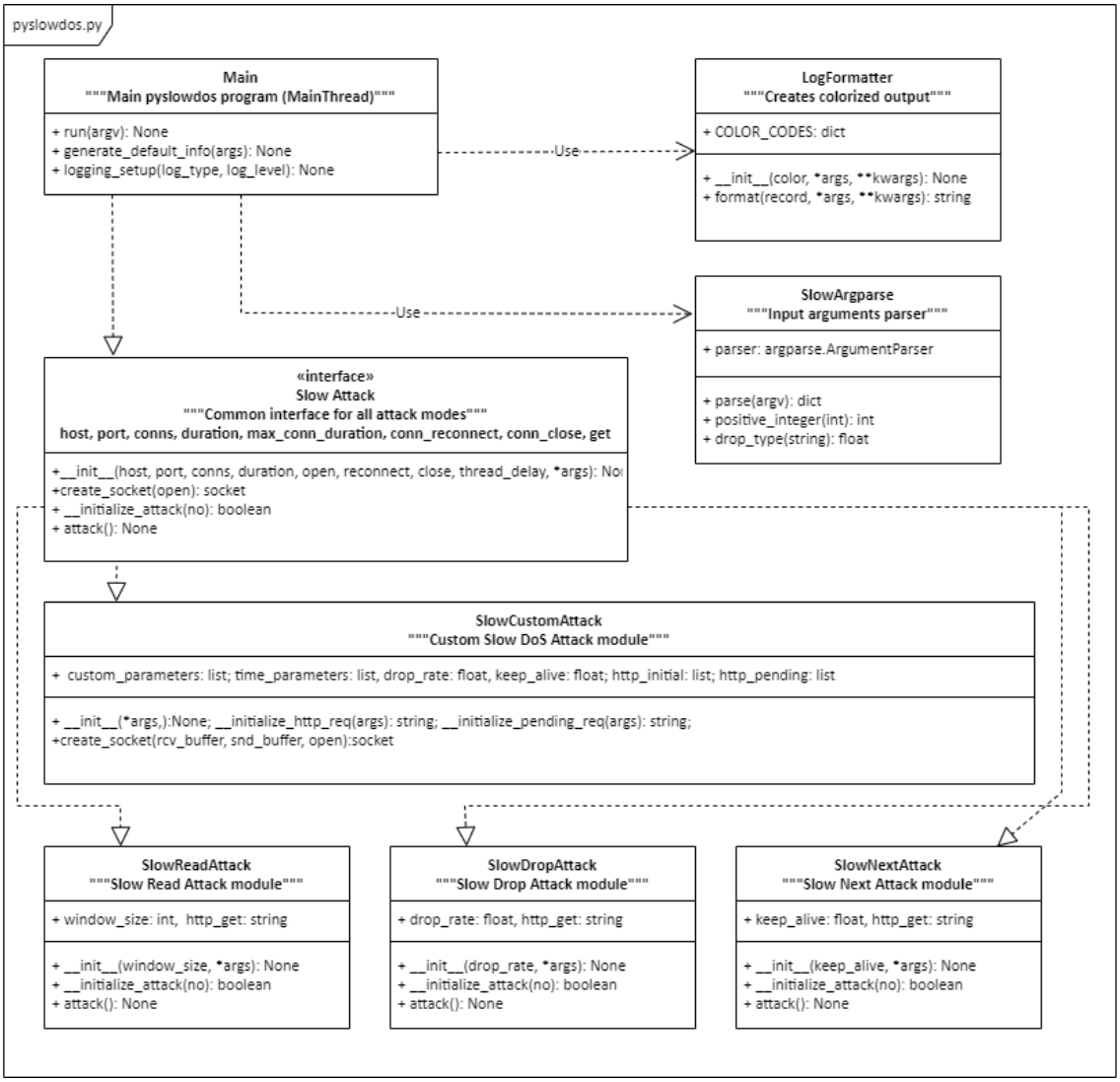


Fig. C.2: Slow DoS Generator UML diagram

C.4 Examples of generated SDAs

Fig. C.3: Slow DoS Generator output in Slow Read mode

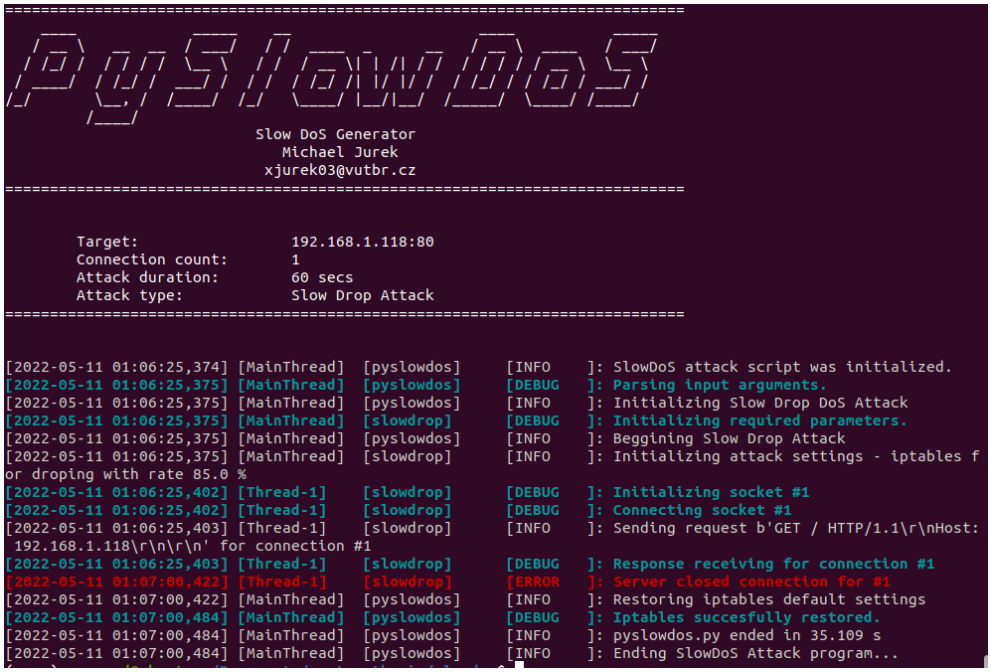


Fig. C.4: Slow DoS Generator output in Slow Drop mode

Fig. C.5: Slow DoS Generator output in Slow Next mode

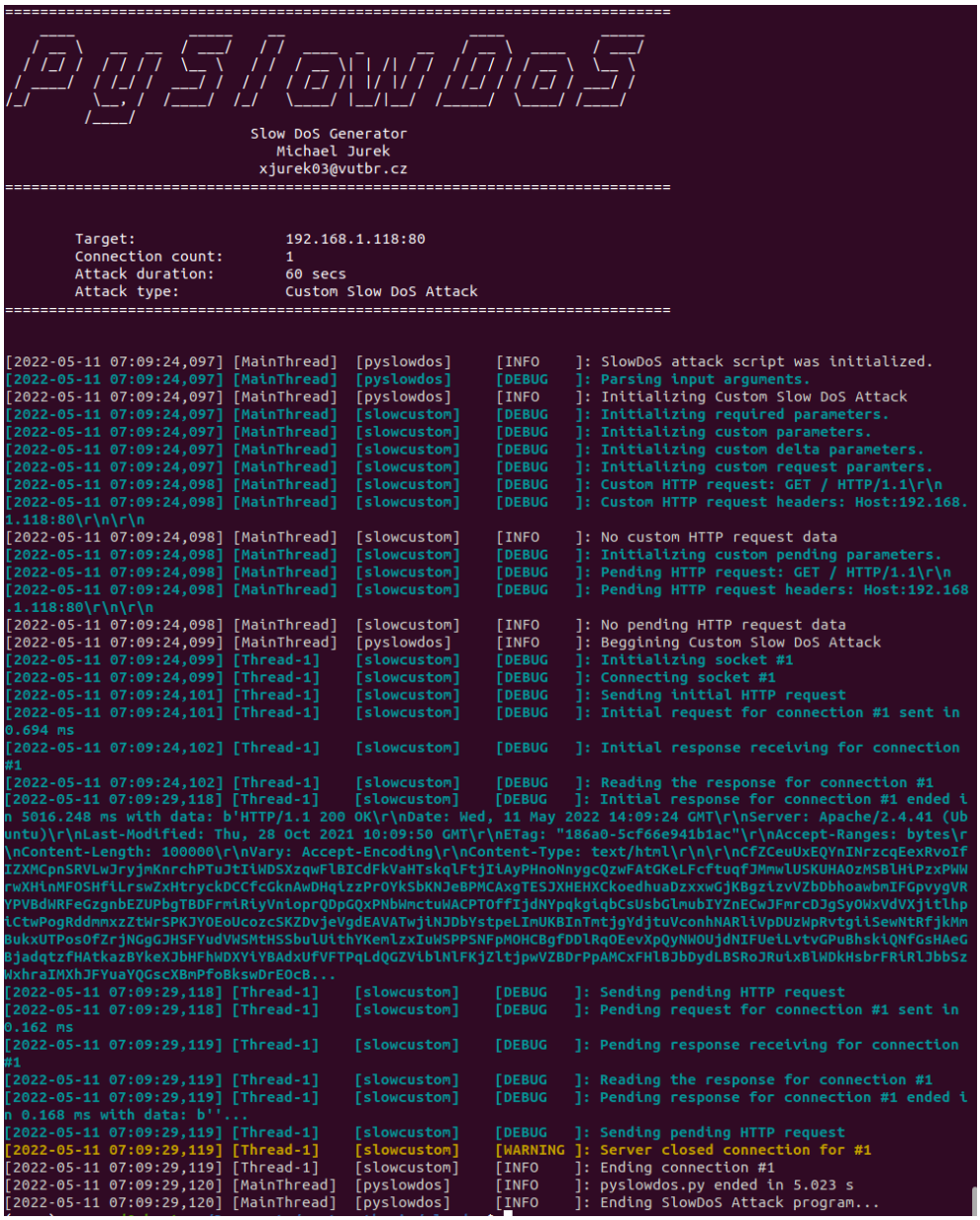


Fig. C.6: Slow DoS Generator output in Custom mode

D Examples of SDAs detection

D.1 Signature-based detection

```
{
    "name": "Slow Read",
    "initial_request" {
        "content": "GET / HTTP/1.1",
        "window_size": 16
    },
    "response": {
        "status_code": 200,
        "content": "Keep-Alive",
        "keep_alive": 5
    },
    "connection": {
        "connections": 250,
        "connection_rate": 70,
        "threads": 10,
        "sending_buffer": 100000
    },
    "timeouts": [0,3,0,0]
}
```

Listing D.1: Specification example of Slow Read attack

```
{
    "name": "Slow Drop",
    "initial_request": "GET / HTTP/1.1",
    "connection": {
        "connections": 200,
        "connection_rate": 100,
        "threads": 5,
        "sending_buffer": 100000,
        "retransmissions": 180
    },
    "timeouts": [0,0,0,10]
}
```

Listing D.2: Specification example of Slow Drop attack

```
{
    "name": "Slow Next",
    "initial_request": "HEAD / HTTP/1.1",
    "response": {
        "status_code": 200,
    },
    "timeouts": [0,0,0,4]
}
```

Listing D.3: Specification example of Slow Next attack

D.2 Slow Read Attack testing

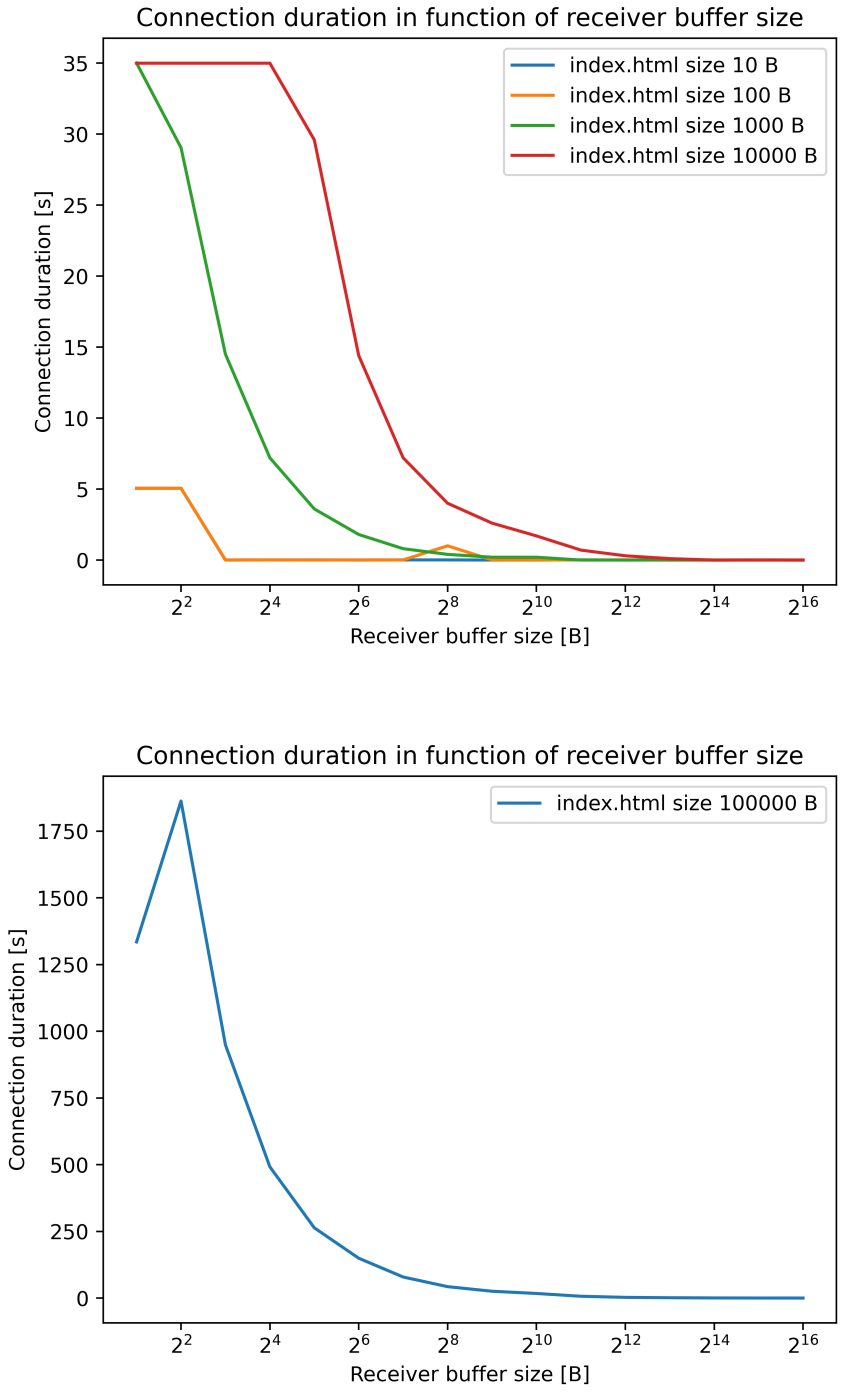


Fig. D.1: Connection duration for different sizes of receiver buffer size

D.3 Flow features

```
features=["src_ip", "dst_ip","src_port","dst_port","src_mac",
"dst_mac","protocol","timestamp","flow_duration","flow_byts_s",
"flow_pkts_s","fwd_pkts_s","bwd_pkts_s","tot_fwd_pkts",
"tot_bwd_pkts","totlen_fwd_pkts","totlen_bwd_pkts","fwd_pkt_\
len_max","fwd_pkt_len_min","fwd_pkt_len_mean","fwd_pkt_len_std",
"bwd_pkt_len_max","bwd_pkt_len_min","bwd_pkt_len_mean",
"bwd_pkt_len_std","pkt_len_max","pkt_len_min","pkt_len_mean",
"pkt_len_std","pkt_len_var","fwd_header_len","bwd_header_len",
"fwd_seg_size_min","fwd_act_data_pkts","flow_iat_mean",
"flow_iat_max","flow_iat_min","flow_iat_std","fwd_iat_tot",
"fwd_iat_max","fwd_iat_min","fwd_iat_mean","fwd_iat_std",
"bwd_iat_tot","bwd_iat_max","bwd_iat_min","bwd_iat_mean",
"bwd_iat_std","fwd_psh_flags","bwd_psh_flags","fwd_urg_flags",
"bwd_urg_flags","fin_flag_cnt","syn_flag_cnt","rst_flag_cnt",
"psh_flag_cnt","ack_flag_cnt","urg_flag_cnt","ece_flag_cnt",
"down_up_ratio","pkt_size_avg","init_fwd_win_byts","init_bwd\
_win_byts","active_max","active_min","active_mean","active_std",
"idle_max","idle_min","idle_mean","idle_std","fwd_byts_b_avg",
"fwd_pkts_b_avg","bwd_byts_b_avg","bwd_pkts_b_avg",
"fwd_blk_rate_avg","bwd_blk_rate_avg","fwd_seg_size_avg",
"bwd_seg_size_avg","cwe_flag_count","subflow_fwd_pkts",
"subflow_bwd_pkts","subflow_fwd_byts","subflow_bwd_byts"]
```

Listing D.4: Flow features generated by Python CICFlowMeter

D.4 `AppDDos.txt`

Attack	Target	After $[h]$	After $[min]$
slowbody2	75.127.97.72	00:53	53
slowread	75.127.97.72	01:58	118
ddossim	75.127.97.72	02:22	142
goldeneye	75.127.97.72	02:50	170
slowheaders	74.63.40.21	02:57	177
rudy	75.127.97.72	03:08	188
ddossim	97.74.144.108	03:28	208
rudy	208.113.162.153	03:29	209
*slow_read	10.0.0.100	03:50	230
hulk	69.84.133.138	04:38	278
slowheaders	67.220.214.50	06:00	360
goldeneye	97.74.144.108	07:06	426
slowbody2	69.192.24.88	08:13	493
slowbody2	97.74.144.108	09:03	543
slowbody2	203.73.24.75	09:09	549
rudy	97.74.144.108	09:20	560
*slow_drop	10.0.0.100	10:00	600
slowread	74.55.1.4	11:02	662
slowheaders	97.74.104.201	11:27	687
*slow_next	10.0.0.100	12:10	730
hulk	74.55.1.4	13:33	813
hulk	69.192.24.88	13:47	827
slowloris	97.74.144.108	15:20	920
slowheaders	97.74.144.108	15:47	947
slowloris	75.127.97.72	16:33	993
slowheaders	75.127.97.72	17:13	1033
goldeneye	69.192.24.88	19:23	1163
rudy	74.55.1.4	20:59	1259
TOTAL	-	23:59	1439

Tab. D.1: Extended CIC DoS dataset from 2017 attack time distribution

E Content of the electronic attachment

/
- `utils` ..... utility modules
  - `generate_index.py` ..... creates custom size index file
- `tests`
  - `tcp_window_size_test.sh` ..... server test on Slow Read connection
  - `tcp_window_size.py`
  - `tcp_window_graph.py`
- `pyslowdos` ..... Slow DoS attack generator
  - `requirements.txt` ..... required modules to be installed
  - `slowargparse.py` ..... module for custom system argument parsing
  - `pyslowdos.py` ..... main program of the generator
  - `slowread.py`
  - `slowdrop.py`
  - `slownext.py`
  - `slowcustom.py`
  - `README.md`
- `supervised.ipynb` ..... Jupyter notebook that implements ML methods
- `sda_ml_detection_model.pkl` ..... machine learning model for SDA detection