



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

APACHE MODULE FOR THE DOS ATTACK MITIGATION

APACHE MODUL PRO MITIGACI DOS ÚTOKŮ

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Róbert Ruman

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. Michael Jurek

BRNO 2023

Bachelor's Thesis

Bachelor's study program **Information Security**

Department of Telecommunications

Student: Róbert Ruman

ID: 231273

**Year of
study:** 3

Academic year: 2022/23

TITLE OF THESIS:

Apache module for the DoS attack mitigation

INSTRUCTION:

The aim of this thesis is to create and test a module to detect and mitigate Slow DoS attacks on Apache web server in the current version.

Tasks:

- Analysis of current solutions for mitigating Slow DoS attacks on Apache web server.
- Design and implementation of a custom module for Apache HTTP server that will detect and block Slow DoS attacks (Slowloris, Slow Read, Slow Post, Slow Drop, Slow Next and RUDY).
e.g. The module will use a combination of various methods, such as limiting the number of requests from individual clients or blacklisting IP addresses.
- Thorough testing of the developed module using unit tests, integration tests, functional and performance tests.
- Verification of the module functionality on simulated Slow DoS attacks, evaluation and clear processing of the results (suitable graphical representation will be chosen).
- Comparison of the created module with other solutions.
- Preparation of documentation for the module, including instructions for installation and use.

RECOMMENDED LITERATURE:

[1] LAURIE, Ben a Peter LAURIE. Apache: The Definitive Guide. 3rd ed. online: O'Reilly Media, 2002. ISBN 9780596002039.,

[2] MACEACHERN, Doug a Lincoln STEIN. Writing Apache Modules with Perl and C. online: O'Reilly Media, 1999. ISBN 9781565925670.

**Date of project
specification:** 6.2.2023

**Deadline for
submission:** 26.5.2023

Supervisor: Ing. Michael Jurek

WARNING:

The author of the Bachelor's Thesis claims that by creating this thesis he/she did not infringe the rights of third persons and the personal and/or property rights of third persons were not subjected to derogatory treatment. The author is fully aware of the legal consequences of an infringement of provisions as per Section 11 and following of Act No 121/2000 Coll. on copyright and rights related to copyright and on amendments to some other laws (the Copyright Act) in the wording of subsequent directives including the possible criminal consequences as resulting from provisions of Part 2, Chapter VI, Article 4 of Criminal Code 40/2009 Coll.

doc. Ing. Jan Hajný, Ph.D.

Chair of study program board

ABSTRACT

This thesis is devoted to the mitigation of multiple types of DoS attacks. Our aim was to create a custom apache module that is able to mitigate flood attacks as well as logical attacks. The module was created in C language using VS Code. After creating the module we ran multiple tests to gather data in order to be able to compare our module to already existing apache modules. Comparing the test result we concluded that our module is able to mitigate both types of attacks. The results of the tests are visualized using graphs in the appendix.

KEYWORDS

DoS, UDP flood, ICMP flood, SYN flood, Slowloris, HTTP flood, attacker, RUDY, Apache, module, server

ABSTRAKT

Táto práca sa venuje mitigácii viacerých typov útokov DoS. Naším cieľom bolo vytvoriť vlastný modul apache, ktorý dokáže zmierniť útoky typu flood, ako aj logické útoky. Modul bol vytvorený v jazyku C pomocou programu VS Code. Po vytvorení modulu sme vykonali viacero testov na získanie údajov, aby sme mohli náš modul porovnať s už existujúcimi modulmi apache. Porovnaním výsledkov testov sme dospeli k záveru, že náš modul dokáže zmierniť oba typy útokov. Výsledky testov sú vizualizované pomocou grafov v prílohe.

KLÍČOVÁ SLOVA

DoS, UDP flood, ICMP flood, SYN flood, Slowloris, HTTP flood, útočník, RUDY, Apache, modul, server

ROZŠÍŘENÝ ABSTRAKT

Cieľom tohto projektu bolo vytvoriť Apache modul ktorý dokáže detekovať a mitigovať DoS (Denial of Service) útoky rôznych typov. Zatiaľ sú dostupné iba Apache moduly ktoré sa zamerávajú iba na jednotlivé DoS útoky.

DoS útoky a jej typy

DoS útoky sú typy kybernetických útokov, ktoré majú za cieľ prerušiť, obmedziť alebo znemožniť normálnu prevádzku systému, služby alebo siete tak, že zasypávajú zdroje, ktoré by mali byť dostupné pre legitímnych používateľov. Typy DoS útokov sú:

- **Záplavové útoky** - Tieto útoky zaplavujú cieľ obrovským množstvom prevádzky, čím zahlcujú sieťové linky, prepínače, smerovače a iné sieťové komponenty. Tým, že sa spotrebuje dostupná šírka pásma a sieťové zdroje, legitímni používatelia nemajú prístup k systému alebo sieti.
- **Protokolové útoky** - Útoky na protokoly využívajú zraniteľnosti alebo slabiny v sieťových protokoloch na narušenie cieľového systému alebo siete. Tieto útoky zvyčajne využívajú chyby v implementácii sieťových protokolov, čím sa systém alebo sieť obeť nedokážu správne spracovať prichádzajúce požiadavky. Medzi príklady protokolových útokov DoS patria útoky SYN Flood, pri ktorých útočník zaplavuje cieľ veľkým počtom paketov SYN, vyčerpá zdroje cieľa a zabráni mu dokončiť protokol TCP procesu odovzdávania príkazov TCP
- **Aplikačné útoky** - Tieto útoky sa zameriavajú na zraniteľnosti v aplikačnej vrstve cieľového systému alebo siete. Tieto útoky využívajú slabiny v spôsobe, akým aplikácie alebo služby spracúvajú a spracovávajú prichádzajúce požiadavky, pričom ich cieľom je vyčerpať možnosti aplikácie zdrojov a spôsobiť nedostupnosť

Ako vytvoriť vlastný Apache modul

Ak chcete vytvoriť vlastný modul pre Apache2, budete potrebovať určité znalosti programovania a konfigurácie Apache2 servera. Tu je základný postup na vytvorenie vlastného modulu:

1. **Príprava prostredia** - Uistite sa, že máte nainštalovaný Apache2 server a vývojové nástroje na vašom systéme. Skontrolujte, či máte nainštalované balíčky potrebné pre vývoj modulov Apache2, napríklad apache2-dev (pre systémy založené na Debiane/Ubuntu) alebo podobné.
2. **Vytvorenie nového adresára pre váš modul** - Vytvorte si adresár pre váš modul v priečinku modules v adresári, kde sa nachádza konfiguračný súbor Apache2 servera. Napríklad: /usr/src/apache2/modules/moj_modul/.

3. **Vytvorenie zdrojových súborov** - Vytvorte súbory so zdrojovým kódom pre váš modul. Typicky sa používa jazyk C. Napríklad `moj_modul.c` pre zdrojový kód a `moj_modul.h` pre hlavičkový súbor. Implementujte požadovanú funkcionality vášho modulu v zdrojovom kóde.
4. **Príprava Makefile** - Vytvorte súbor Makefile v priečinku vášho modulu na kompiláciu modulu.
5. **Kompilácia a inštalácia modulu** - V termináli prejdite do priečinka vášho modulu a spustíte príkaz `make`, ktorý skompiluje váš modul. Ak kompilácia prebehne úspešne, spustíte príkaz `make install`, ktorý nainštaluje váš modul do adresára Apache2 modulov. Ak všetko prebehne správne, môžete pokračovať krokom nasledujúcim.
6. **Konfigurácia Apache2** - Otvorte konfiguračný súbor Apache2, ktorý sa zvyčajne nachádza na ceste `/etc/apache2/apache2.conf`. V konfiguračnom súbore pridajte riadok `LoadModule moj_modul modules/moj_modul.so`, ktorý načíta váš modul do Apache2. Uložte a zatvorte konfiguračný súbor.
7. **Reštartujte Apache2 server** - V termináli spustíte príkaz `sudo service apache2 restart`, aby sa zmeny aplikovali a váš vlastný modul bol načítaný.

Po týchto krokoch by váš vlastný modul mal byť úspešne vytvorený a načítaný v Apache2 serveri. Odporúča sa dôkladne testovať a overiť správne fungovanie modulu.

Implementácia nášho modulu

Môj modul je možné stiahnuť z adresára na [GitHub.com](https://github.com). Po stiahnutí je potrebné vstúpiť do stiahnutej zložky pomocou príkazu "cd"

```
sudo cd <cesta_k_stiahnutému_súboru>
```

Je dôležité, že "`<cesta_k_stiahnutému_súboru>`", by mala byť nahradená príslušnou hodnotou. Pre inštaláciu a kompiláciu zdrojového kódu modulu je potrebné použiť administratívne práva pomocou príkazu `sudo` a pomocou nej vykonať nasledujúci príkaz:

```
sudo apxs -i -a -c mod_apache_module.c
```

Po úspešnej kompilácii by mal terminál zobraziť príslušnú správu, ktorá indikuje, že proces bol dokončený, sme žiadaný o reštartovanie serveru Apache, čo môžeme dosiahnuť nasledujúcim príkazom:

```
systemctl restart apache2
```

Simulácia útokov

Po úspešnej kompilácii modulu sme simulovali rôzne útoky. Pre toto testovanie sme použili dva virtuálne počítače:

- Ubuntu s verziou 22.04.2 LTS - Server
- Kali s verziou 2023.1 - Útočník

Pre vykonanie útokov som použil súbor nástrojov hping, test.pl (testovací kód ktorý patrí k modulu mod_evasive) a verejný projekt slowloris ktorý sa dá stiahnuť z GitHub-u.

Najprv som vykonal záplavové útoky pomocou hping a test.pl, ak nebol žiadny z dvoch modulov (mod_evasive a mod_apache2) tak útok úspešne prebehol a došlo k nedostupnosti serveru. Vyskúšal som oba moduly individuálne a dostal som identické výsledky, že útok bol zastavený a pre legitímneho používateľa bol server dostupný, týmto sme overili že modul je schopný mitigovať záplavové útoky. Ďalší na rade bol slowloris. Ak boli moduly mod_apache2 a mod_antiloris vypnuté, útoky prebehli úspešne a server bol nedostupný. Ak sme aktivovali jednotlivé moduly zvlášť, mohol som vidieť že útoky boli úspešne zastavené, to znamená že modul je použiteľný aj na mitigovanie logických útokov.

Záver

Po vykonaní testov a porovnaní modulov, je možné vyhlásiť že naše zadanie bolo úspešne splnené. Môj vlastný modul je možné použiť na mitigovanie rôznych DoS útokov. Jednotlivé grafy nasnímané počas simulovania útokov je možné pohliadať v prílohe. Na grafoch je možné vidieť že počas záplavových útokov aj keď sa jedná o záplavový útok s väčšou kapacitou pri dosiahnutí limitu počet žiadostí za sekundu náhle klesne, je to z dôvodu že bola adresa útočníka blacklistovaná. Pri slowloris útoku je možné na grafu vidieť že náš modul automaticky zamietá žiadosti od útočníka, totiž bol dosiahnutý maximálny počet spojení z jednej adresy. V tomto prípade som simuloval útok s 400 socketmi a náš modul zamietal 360 z nich. V modulu bolo nastavené aby každý používateľ mohol mať maximálne 20 pripojení na čítanie dat resp. GET request, a 20 pripojení na písanie resp. POST request.

Author's Declaration

Author: Róbert Ruman
Author's ID: 231273
Paper type: Bachelor's Thesis
Academic year: 2022/23
Topic: Apache module for the DoS attack mitigation

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 advisor of my thesis, Ing. Michael Jurek for his valuable comments, support, guidance and expertise throughout the entire process of creating this project.

Contents

Introduction	13
1 Theory of DoS attacks and Apache2 modules	14
1.1 What is a DoS (Denial of Service) attack	14
1.1.1 Types of DoS attacks	14
1.2 Volumetric Attacks	15
1.2.1 UDP Flood	15
1.3 HTTP flood attack	17
1.3.1 How it works	17
1.3.2 How can it be detected	18
1.4 Protocol Attacks	18
1.4.1 ICMP (Ping) Flood	18
1.4.2 SYN Flood Attack	20
1.5 Application Layer Attacks	23
1.5.1 Slowloris	23
1.5.2 RUDY	25
1.6 Application of Apache2 modules	26
1.6.1 mod_evasive	26
1.6.2 mod_antiloris	26
1.7 Introduction to creating a custom module for Apache2	27
1.7.1 Parts of a module	27
1.7.2 Planning and Design	28
1.8 Module Development	29
1.8.1 Creating the Module Structure	29
1.8.2 Configuring the Module	29
1.8.3 Implementing Module Functionality	30
1.8.4 Error Handling and Logging	30
1.9 Module Integration	30
1.9.1 Compilation	30
1.9.2 Installation	30
1.9.3 Loading the Module	31
1.9.4 Configuring Apache for the Custom Module	31
1.10 Creation of my module	32
1.10.1 Principle of the program	32
1.11 Parameters of the module	32
1.12 Download and setup of my module	35

2	Thesis Results	37
2.1	Research	37
2.2	Basic usage of Apache2	37
2.2.1	Installing modules on Apache2	38
2.3	Implementation	38
2.3.1	Mitigation of DoS attacks using mod_evasive	38
2.3.2	Mitigation of Slowloris using mod_antiloris	40
2.4	Tests and Evaluation	40
2.4.1	Testing a UDP flood attack	41
2.5	Tests and Evaluation of my custom module	42
2.5.1	Datadog	42
2.5.2	Simulating a Slowloris attack	43
2.5.3	Simulating a Flood attack	44
	Conclusion	45
	Bibliography	46
	Symbols and abbreviations	50
	List of appendices	51
A	Graphs of Apache metrics during attacks	52
B	Content of the electronic attachment	57

List of Figures

1.1	An attacker committing a UDP flood attack on a server through a bot device[3].	16
1.2	Principle of a HTTP flood attack[6].	17
1.3	Principle of an ICMP flood attack[8].	18
1.4	Principle of a SYN flood attack.[10]	20
1.5	Settings for the module mod_reqtimeout	22
1.6	Difference between a legitimate connection and a Slowloris Attack.[12]	23
1.7	Expected displayed message after a successful compilation.	35
2.1	Basic commands for the Apache2 server.	37
2.2	Commands to interact with modules in Apache.	37
2.3	Creating the log directory for mod_evasive.	39
2.4	Source code of test.pl.	40
2.5	Steps to download and install mod_antiloris.	40
2.6	Error message received by the attacker.	42
A.1	Apache CPU usage during a flood attack with my module turned off.	52
A.2	Bytes served by the server during a flood attack with my module turned off.	52
A.3	Bytes served by the server during a flood attack with my module turned on.	53
A.4	Connections closed during a slowloris attack with my module turned on.	53
A.5	Connections during a slowloris attack with my module turned on. . .	53
A.6	CPU usage of server during an attack with my module turned on. . .	54
A.7	Amount of hits on the server with my module turned off.	54
A.8	Rate of bytes on the server during a flood attack with my module turned on.	55
A.9	Rate of bytes served by the server with my module turned off.	55
A.10	Rate of requests during a flood attack with my module turned off. . .	55
A.11	Rate of requests during a flood attack with my module turned on. . .	56

List of Tables

2.1	Comparison of the mitigation capabilities of the modules.	45
-----	---	----

Introduction

This thesis is devoted to the **Mitigation of DoS (Denial of Service) attacks** on the Apache2 servers.

These attacks are becoming more common and they have the potential to cause billions of dollars worth of damage. It is impossible to completely protect web servers from DoS attacks as there is not much control the admin has over the traffic coming to your site.

Experiencing DoS attacks may appear inevitable when operating online, especially if your site is successful, increasing the likelihood of becoming a target at some point. However, there are measures you can take to decrease the probability of a DoS attack affecting your website.

There are various motives behind an attacker's desire to render your website inoperable through a DoS attack. These motives include attacks from competitors and attacks based on the content of your website. In an ideal scenario, competitors would strive to outperform you online through legitimate means. However, in certain cases, competitors may resort to more extreme measures. They might hire someone to launch a DoS attack on your site, knowing that it will not only impact your website but also harm your business. Proving the identity of the perpetrator behind any DoS attack is undoubtedly a challenging task.

The aim of the thesis was to get a better understanding of how Apache and its modules work and to create a custom Apache module that is able to mitigate multiple types of DoS attacks such as flood attacks and logical attacks. Later both the already available modules and my custom module were applied to the server and different types of attacks were simulated in aim of comparing each module and their capabilities.

1 Theory of DoS attacks and Apache2 modules

1.1 What is a DoS (Denial of Service) attack

A Denial-of-Service (DoS) attack is a cyber-attack on a system or device with a goal to make its service unavailable to legitimate users. The attacker achieves it by overwhelming the target with traffic or by sending information causing it to crash. The outcome of this attack causes the target to be unavailable to legitimate users.

DoS attacks can be classified into two general groups: flood attacks and logical attacks. A flood attack occurs when a system capacity and ability to process requests caused by an excessive amount of data received. As a result, the server slows down and eventually becomes unresponsive.

Alternatively, certain DoS attacks make use of weaknesses or flaws in the targeted system or service, intentionally leading to its failure. These attacks occur by sending input that capitalizes on vulnerabilities or errors in the target, leading to crashes or significant disruptions that render the system unattainable or inoperable.[1]

1.1.1 Types of DoS attacks

The categorization of Denial of Service (DoS) attacks encompasses multiple principles, and broadly speaking, they are commonly classified in the following manner:

1. **Volumetric Attacks:** Volumetric attacks, also known as flood attacks, target the network bandwidth and infrastructure resources of the victim system or network. These attacks flood the target with an overwhelming amount of traffic. By consuming the available bandwidth and network resources, legitimate users are unable to access the system or network.
2. **Protocol Attacks:** Protocol attacks exploit vulnerabilities or weaknesses in network protocols to disrupt the targeted system or network. These attacks typically exploit flaws in the implementation of network protocols, making the victim's system or network unable to handle incoming requests properly.

An example of protocol-based DoS attacks is SYN Flood attack, where the attacker floods the target with a large number of SYN packets, exhausting the target's resources and preventing it from completing the TCP handshake process.

3. **Application Layer Attacks:** Application layer attacks, also known as layer 7 attacks, target vulnerabilities in the application layer of the target system or network. These attacks exploit weaknesses in the way applications or services handle and process incoming requests, aiming to exhaust the application's resources and cause a denial of service.

Slowloris and RUDY (R-U-Dead-Yet) are examples of application layer attacks. Slowloris maintains multiple connections to a target web server and sends partial HTTP requests, keeping the connections open and consuming server resources, ultimately rendering the server unavailable to legitimate users. RUDY, on the other hand, sends specially crafted HTTP POST requests with large payloads that are designed to tie up server resources, causing delays and potentially leading to a denial of service.[2]

1.2 Volumetric Attacks

1.2.1 UDP Flood

A UDP flood is a type of DoS attack in which the attacker sends spoofed User Datagram Protocol (UDP) packets at a very high packet rate to a host with the aim of overpowering the systems resource causing it to be unavailable to users.

How it works

A UDP flood works exploits the steps that a server takes when it responds to a UDP packet sent to one of its ports. Under normal conditions, when a server receives a UDP packet at a particular port, it goes through two steps in response:

1. The server verifies whether any active programs are currently listening for requests on the designated port.
2. If no programs are detected to be actively receiving packets on that particular port, the server promptly sends an ICMP (ping) packet as a response, indicating to the sender that the destination is unreachable.

When a server receives a new UDP packet, it goes through a series of operations to handle the request, which in turn consumes server resources. Each UDP packet transmitted includes the IP address of the device that sent it. In the context of a DDoS attack, the attacker typically alters or falsifies the source IP address of the UDP packets, employing this deceptive tactic to avoid disclosing their actual location. By doing so, they mitigate the risk of their location being overwhelmed by the response packets generated by the targeted server.

Consequently, due to the server's utilization of resources to inspect and respond to each received UDP packet, its resources can quickly deplete when faced with a

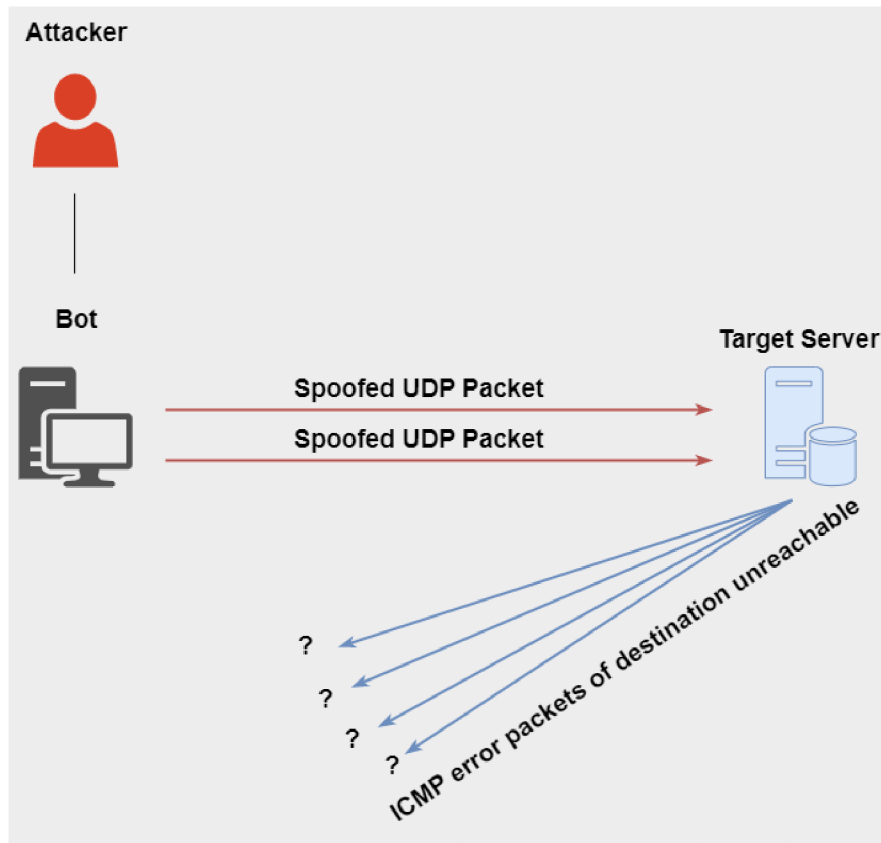


Fig. 1.1: An attacker committing a UDP flood attack on a server through a bot device[3].

large influx of UDP packets. This depletion ultimately results in a denial-of-service situation, impeding the regular flow of traffic.[4]

Mitigation of UDP Flood

Most operating systems limit the response rate of ICMP packets in part to disrupt DDoS attacks that require ICMP response. One drawback of this type of mitigation is that during an attack legitimate packets may also be filtered in the process. If the UDP flood has a volume high enough to saturate the state table of the targeted server's firewall, any mitigation that occurs at the server level will be insufficient as the bottleneck will occur upstream from the targeted device.

Using the command Iftop on linux it is possible to continuously monitor the bandwidth usage of the server.

1.3 HTTP flood attack

HTTP flood attack is an attack, which floods a server with process-intensive requests until it no longer has the capacity to respond to legitimate user requests. Opposed to SYN or ACK flood attacks which are carried out on the network and application layer (Layers 3 and 4) HTTP flood attacks target the application layer (Layer 7) in order to penetrate the weakest component of an infrastructure and thus cause overload. Unlike other attacks, HTTP floods are based on technically correctly formulated (valid) requests to the web server being attacked.[5]

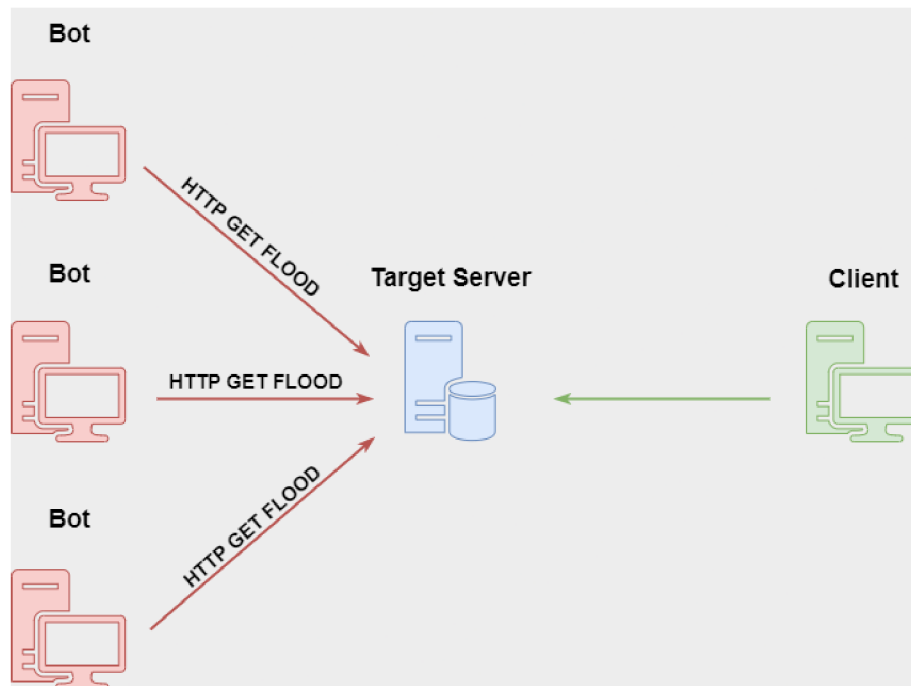


Fig. 1.2: Principle of a HTTP flood attack[6].

1.3.1 How it works

In an HTTP flood attack attackers flood a web server with HTTP requests that specifically request pages with large loading volumes. This causes the server to overload to the point it is no longer able to process legitimate requests. As a result, the website or web application is no longer accessible for users. Attackers often employ botnets for such attacks to maximize the efficiency and impact of their attack. Botnets usually consist of thousands of commandeered and then remotely controlled computers and networked systems from the IoT (Internet of Things).

1.3.2 How can it be detected

To reliably distinguish attack traffic from legitimate user requests, it is essential to understand the content of the requests and put them in context. It can be done by analyzing all incoming requests before they reach the web server. This enables them to automatically detect abnormal traffic patterns and ward off HTTP flood attacks at an early stage. Once the attack traffic is identified, the requests associated with it can be rigorously blocked or discarded. It can be achieved by multiple Apache2 modules.

1.4 Protocol Attacks

1.4.1 ICMP (Ping) Flood

An ICMP flood also known as Ping flood is a denial-of-service attack in which the attacker floods a targeted device with ICMP packets, causing the target to become inaccessible to normal traffic.[7]

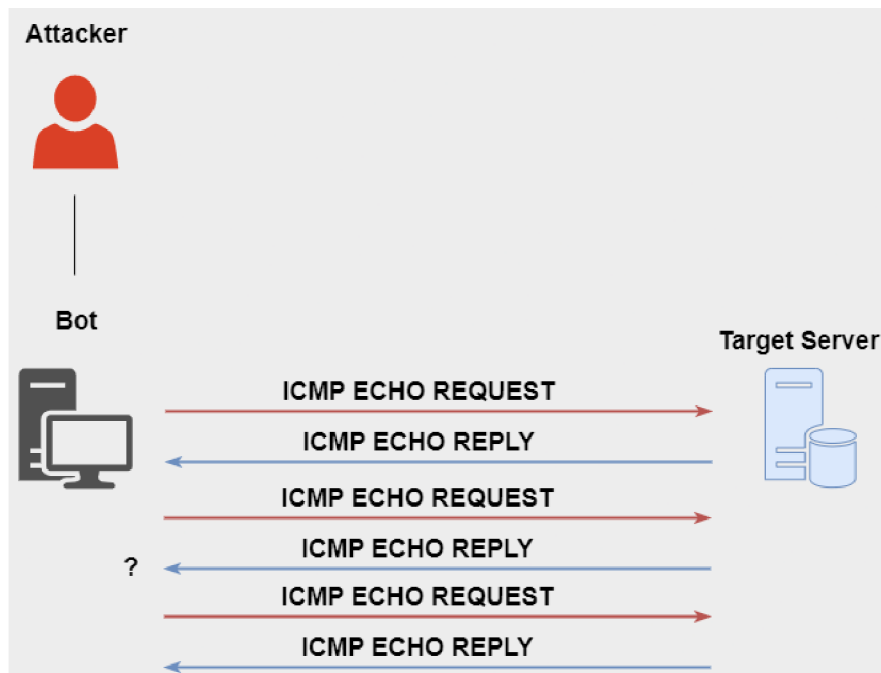


Fig. 1.3: Principle of an ICMP flood attack[8].

How it works

The Internet Control Message Protocol (ICMP), which is utilized in a Ping Flood attack, is an internet layer protocol used by network devices to communicate. The network diagnostic tools traceroute and ping both operate using ICMP. Commonly, ICMP echo-request and echo-reply messages are used to ping a network device for the purpose of diagnosing the health and connectivity of the device and the connection between the sender and the device.

When an ICMP request is made, server resources are required to process each request and send a response. Bandwidth is also needed for both the incoming message (echo-request) and outgoing response (echo-reply). The Ping Flood attack seeks to overwhelm the targeted device's capacity to handle the high volume of requests and potentially overload the network connection with fraudulent traffic. By utilizing numerous devices in a botnet to direct ICMP requests at the same internet property or infrastructure component, the attack traffic is significantly increased, potentially leading to disruption of normal network operations. In the past, attackers would often falsify the IP address to conceal the source device. However, in modern botnet attacks, malicious actors no longer find it necessary to mask the bot's IP (Internet Protocol) and instead rely on a large network of unspoofed bots to saturate the target's capacity.

The extent of damage caused by a Ping Flood attack is directly proportional to the number of requests directed at the targeted server. Unlike reflection-based DDoS attacks such as NTP (Network Time Protocol) amplification and DNS amplification, Ping Flood attack traffic is symmetrical. The bandwidth received by the targeted device simply reflects the cumulative traffic sent from each bot.

Mitigation of ICMP Flood attack

- Ping flood attack utilizes Internet Control Message Protocol (ICMP), an internet layer protocol used by network devices to communicate. Disabling a ping flood becomes easiest by disabling the ICMP functionality of the victim device. However, doing this will disable all activities that use ICMP like ping requests, traceroute requests, and other network activities.
- Blocking ping floods from outside your network can be achieved by reconfiguring firewall to disallow pings. However, internal attacks from within your network cannot be mitigated by firewall configurations.
- A robust mitigation strategy against ICMP floods will put a cap on the number of requests and the rate at which they are received.

1.4.2 SYN Flood Attack

A SYN Flood, also known as a half-open attack, is a form of Denial of Service (DoS) attack that seeks to render the server inaccessible to legitimate users by depleting all available server resources. The attacker achieves this by continuously sending SYN (connection request) packets, overwhelming the server's available ports. As a result, the server's responsiveness to legitimate users is greatly diminished or completely disrupted.[9]

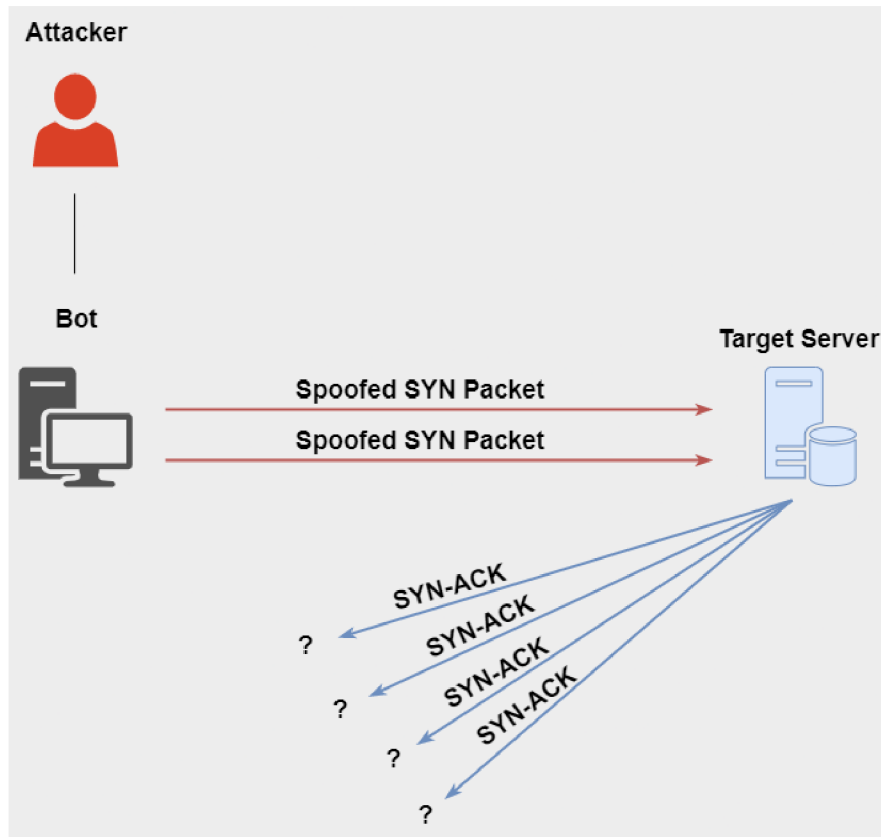


Fig. 1.4: Principle of a SYN flood attack.[10]

How it works

The attacker floods the server with a large number of SYN packets, utilizing a falsified IP address. Upon receiving each request, the server responds and keeps a port open, expecting a final ACK (Acknowledgment) packet that never materializes. Meanwhile, the attacker persists in sending additional SYN packets. With each new SYN packet, the server temporarily establishes a new open port connection for a specific duration. As this process continues and all available ports become occupied, the server's normal functionality is impaired, rendering it unable to operate effectively.

Three different ways a SYN flood can occur

- **Direct Attack** – A SYN Flood attack where the IP address is not spoofed is known as a direct attack. In this case, the attacker does not mask their IP address. The attacker is using a single source device with a real IP address to create the attack. This results in the attacker highly vulnerable to discovery and mitigation. To do this attack, the attacker prevents their device from responding to the server's SYN-ACK packets. This is can be achieved by firewall rules. In practice this method is used rarely, as mitigation is fairly straightforward (blocking the IP address of each malicious system).
- **Spoofed Attack** – A user can spoof the IP address on each SYN packet they send in order to inhibit mitigation efforts and makes them less discoverable. These packets are spoofed, however they can potentially be traced back to their source.
- **Distributed Attack (DDoS)** – In this case the attacker is using some kind of botnet, which makes the likelihood of tracking the attack back to its source is low. For an added level of obfuscation the attacker may have each device also spoof the IP address from which it sends packets.

By using SYN flood attack, the attacker can attempt a DoS attack in a target device or service with a substantially less traffic than other DoS attacks. Instead of volumetric attacks, SYN attacks only need to be larger than the available backlog in the target's operating system. If the attacker is able to determine the size of the backlog and how long each connection will be left open before timing out, the attacker can target the exact parameters needed to disable the system. This way he can reduce the traffic to the minimum necessary amount to deny the servers functioning or its service.

Mitigation of SYN Flood attack

The limit on the number of simultaneous requests that will be served by Apache is decided by the `MaxRequestWorkers` (called `MaxClients` before version 2.3.13) directive, and is set to 256, by default. Any connection attempts over this limit will normally be queued, up to a number based on the `ListenBacklog` directive, which is 511, by default. However, it is best to increase this, to prevent TCP SYN flood attacks.

In this example the system is configured to wait up to 20 seconds for header data. If the client sends data, the server increases the timeout by 1 second for every 500 bytes received, but does not allow more than 40 seconds for the request header. The server is also set to wait up to wait 10 seconds to receive the request body. If

```
root@kali: /etc/apache2/mods-enabled
File Actions Edit View Help
<IfModule reqtimeout_module>

# mod_reqtimeout limits the time waiting on the client to prevent an
# attacker from causing a denial of service by opening many connections
# but not sending requests. This file tries to give a sensible default
# configuration, but it may be necessary to tune the timeout values to
# the actual situation. Note that it is also possible to configure
# mod_reqtimeout per virtual host.

# Wait max 20 seconds for the first byte of the request line+headers
# From then, require a minimum data rate of 500 bytes/s, but don't
# wait longer than 40 seconds in total.
# Note: Lower timeouts may make sense on non-ssl virtual hosts but can
# cause problem with ssl enabled virtual hosts: This timeout includes
# the time a browser may need to fetch the CRL for the certificate. If
# the CRL server is not reachable, it may take more than 10 seconds
# until the browser gives up.
RequestReadTimeout header=20-40,minrate=500

# Wait max 10 seconds for the first byte of the request body (if any)
# From then, require a minimum data rate of 500 bytes/s
RequestReadTimeout body=10,minrate=500

</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
~
~
~
~
~
~
~

15,1-8 All
```

Fig. 1.5: Settings for the module mod_reqtimeout

the client sends data, the system increases the timeout by 1 second for every 500 bytes received, with no upper limit for the timeout.

Apart from the above, an alternative solution exists, which is *mod_devsive*. This module will allow you to specify a maximum number of requests executed by the same IP address. If the threshold is reached, the IP address is blacklisted for the time period you specify. The only problem with this module is that users, in general, do not have unique IP addresses. Many users browse through proxies, or are hidden behind a NAT (network address translation) system. Blacklisting a proxy will cause all users behind it to be blacklisted.

1.5 Application Layer Attacks

1.5.1 Slowloris

Slowloris is a DOS attack program which allows an attacker to overwhelm a targeted server by opening and maintaining many simultaneous HTTP keep-alive connections between the attacker and the target.[11]

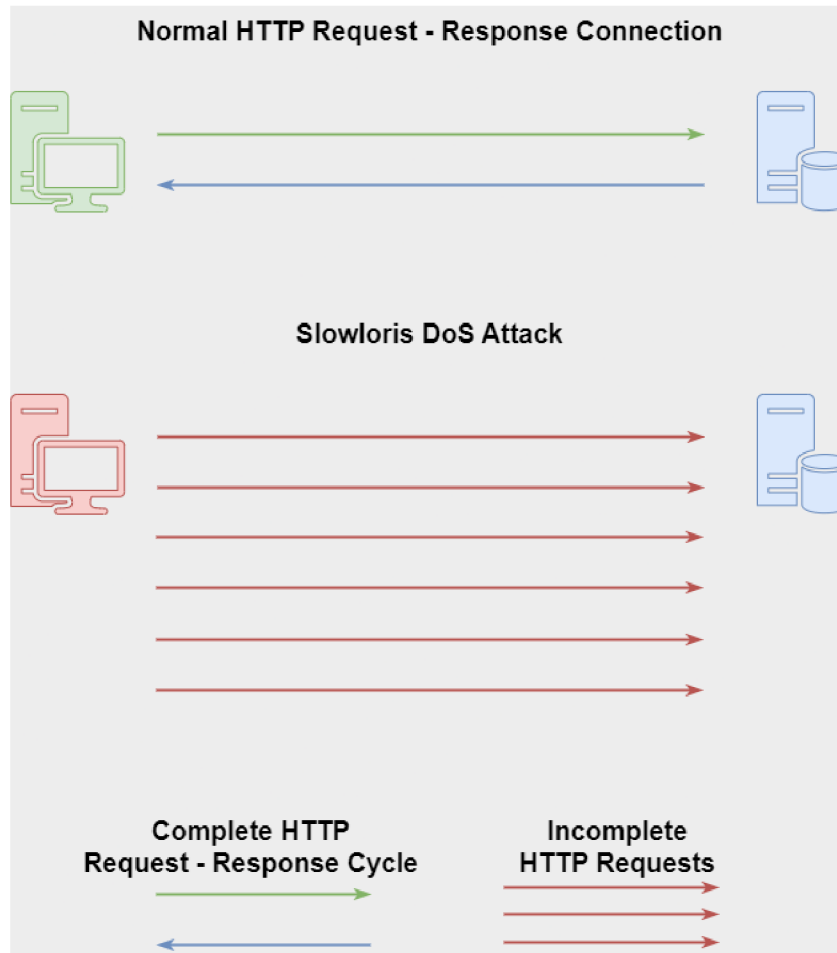


Fig. 1.6: Difference between a legitimate connection and a Slowloris Attack.[12]

How it works

Slowloris is an application layer attack that operates by leveraging fragmented HTTP requests. The attack strategy involves establishing connections with a targeted server and prolonging them for as long as possible. Slowloris is a specific attack tool devised to bring down a server using minimal bandwidth from a single device. This attack consumes server resources by generating requests that appear

slower than usual but mimic regular traffic. The server possesses a limited number of threads to handle existing connections. Each server thread strives to remain active while awaiting the completion of the slow request, which never transpires. Once the server's maximum allowable connections have been surpassed, any additional connection attempts go unanswered, thereby preventing legitimate users from establishing a connection with the server.

Four steps of a Slowloris attack

- The attacker opens multiple connections to the targeted server by sending partial HTTP request headers.
- The server opens a thread for each incoming request, with the intent of closing the thread once the connection is completed. If a connection takes too long, the server will timeout the exceedingly long connections in order to free the thread up for new requests.
- To prevent the target from timing out the connections, the attacker periodically sends requests to the server in order to keep the connection alive.
- The targeted server is never able to release any of the open connections while waiting for the termination of the request. Once all threads are in use, the server is unable to respond to new additional requests made from legitimate users, resulting in a Denial Of Service

Mitigation of Slowloris

- **Increase server availability** – By increasing the maximum number of clients the server will allow, the attacker is forced to increase the number of connections used to overload the server.
- **Rate limit incoming requests** – Restricting access based on certain usage factors will help mitigate a Slowloris attack. Techniques such as limiting the maximum number of connections a single IP address is allowed to make, restricting low transfer speeds, limiting the maximum time a client is allowed to stay connected, are all possible solutions to limit the effectiveness of this attack.
- **Cloud-based protection** – Use a service that can function as a reverse proxy, protecting the origin server

1.5.2 RUDY

RUDY (R-U-Dead-Yet) is a type of application layer Denial-of-Service (DoS) attack that specifically targets web servers and exploits vulnerabilities in the way they handle HTTP (Hypertext Transfer Protocol) requests. This attack aims to exhaust the server's resources, rendering it unresponsive to legitimate users. The RUDY attack method capitalizes on the server's behavior of allocating resources, such as memory and processing power, for each incoming request. By keeping these connections open for an extended period, the attack exhausts the server's resources, preventing it from accepting new connections or serving legitimate requests. The slow and persistent nature of RUDY makes it particularly effective against servers that are not adequately equipped to handle long-duration requests or lack proper mitigation mechanisms. During a RUDY attack, the attacker sends partial HTTP POST requests, incrementally sending small chunks of data, which keeps the connection open without completing the request. Additionally, the attacker periodically sends headers with the "Content-Length" field set to a large value, further exacerbating the resource exhaustion by tricking the server into allocating additional resources to handle the incoming request.[13]

Mitigation of RUDY

Mitigating RUDY attacks requires a multi-layered defense strategy. Web administrators can employ various techniques such as rate limiting, connection timeouts, and request validation to identify and block suspicious or slow connections. Intrusion detection and prevention systems can also be utilized to detect and block abnormal HTTP traffic patterns associated with RUDY attacks.

How it works

The RUDY attack method revolves around creating a deliberate but gradual HTTP POST connection with the targeted server. In contrast to conventional flood-based attacks that prioritize a large number of requests, RUDY takes advantage of the server's resource allocation mechanisms by transmitting HTTP POST requests that appear legitimate but proceed at an exceptionally slow pace. The attacker deliberately sends these requests at a sluggish rate, aiming to prolong the duration of the connection and occupy server resources.

1.6 Application of Apache2 modules

Modules are plugins that add a specific functionality to the web server. These modules can be turned on/off by the administrators. Debian stores these plugins in special directories in `/etc/apache2`, there two directories can be found **mods-available** which consists of all the modules with a known configuration, and **mods-enabled** which contains links to certain modules found in the mods-available (these modules are in use).

1.6.1 `mod_evasive`

This module is used to stop the attackers from bringing down the server with a SYN Flood attack. It creates a hash table in which the hashes of connected devices are stored. The module keeps count of each repeated connection from a same address, if the number of connections exceeds a certain amount in a short amount of time the address gets timed out for a limited time (these parameters are set by the server administrator). These parameters can be changed `/etc/apache2/mods-enabled/evasive.conf`.

1.6.2 `mod_antiloris`

This module is a convenient way to limit the number of simultaneous connections per IP address that are in the "reading request" state on Apache 2.x systems. It is best to use in pair with `mod_reqtimeout`.

1.7 Introduction to creating a custom module for Apache2

1.7.1 Parts of a module

Apache modules usually consist of the following parts:

- **Module structure** - The module structure defines the organization of code and resources within the module. It typically includes source code files, headers, configuration files, and makefiles. The structure ensures proper encapsulation and separation of concerns, making it easier to maintain and extend the module.
- **Module hooks** - Apache modules interact with the server through hooks. Hooks are specific points in the request/response processing lifecycle where modules can intervene and modify the server's behavior. Examples of hooks include request parsing, authentication, response generation, and logging. Modules register their interest in specific hooks, and Apache calls the module's associated functions at those points during request processing.
- **Module initialization** - When Apache starts, it initializes each module by calling its initialization function. This function allows the module to perform any necessary setup tasks, such as allocating memory, registering hooks, and initializing data structures. The initialization function ensures that the module is ready to handle requests effectively.
- **Module configuration** - Modules can define their configuration directives, allowing administrators to customize the module's behavior through Apache's configuration files. Configuration directives specify parameters and settings for the module. When Apache starts, it processes the configuration files, parses the directives, and assigns the values to the corresponding module configuration data structures.
- **Module functions** - Modules implement various functions to handle specific tasks and provide the desired functionality. These functions can include request/response processing, data manipulation, filtering, logging, and more. Modules can also define callback functions to be executed at specific events, such as connection establishment or request completion. By implementing these functions, modules extend the server's capabilities and modify its behavior as required.[14]

1.7.2 Planning and Design

Before diving into the development process, it's crucial to plan and design your custom module. This stage sets the foundation for a successful implementation. Here are the key aspects to consider:

- **Defining Objectives** - Clearly articulate the objectives and goals of your custom module. Determine the specific functionality you intend to add or modify within Apache. Consider whether the objective is to enhance performance, introduce new features, or integrate with external systems. Establishing clear objectives will provide a solid foundation for the development process.
- **Understanding Apache Architecture** - Thoroughly acquaint yourself with the underlying architecture of Apache to gain a comprehensive understanding of how your custom module fits into the server's ecosystem. Study the request/response lifecycle, including stages such as request parsing, URI mapping, content generation, and response delivery. Identify the precise stages where your module will intervene and modify the server's behavior.
- **Apache Module API Overview** - Familiarize yourself with the Apache Module API, which offers a set of functions, data structures, and hooks for interacting with the server. Diligently study the API documentation to comprehend the available functions and hooks that can be utilized within your custom module. Identify the appropriate hooks for achieving your module's objectives and establish their placement within the request/response lifecycle.
- **Module Development Environment Setup** - Establish an appropriate development environment for building your custom module. Ensure that you have a compatible version of Apache installed on your system. Install the necessary development libraries, headers, and tools required for module compilation. Gain familiarity with Apache's directory structure and configuration files to ensure a smooth development workflow.
- **Creating the Module Structure** - Define the structure of your custom module, adhering to industry best practices. Typically, a module comprises source code files, headers, and makefiles for compilation purposes. Organize these files within a dedicated directory for your module. Employ a modular and well-organized approach, separating distinct functionalities into separate files for clarity and maintainability.
- **Configuring the Module** - Determine the configuration options for your module, taking into account the needs of your intended users. Establish the mechanism for exposing these options within Apache's configuration system. This may involve defining custom directives that users can include in their configuration files. Pay careful attention to the syntax, validation rules, and

default values of these directives.

- **Implementing Module Functionality** - Implement the desired functionality within your custom module, aligning with the defined objectives. Utilize the Apache Module API to interact with the server and modify the request/response flow as required. Develop the necessary code to handle the identified hooks, ensuring that it adheres to coding standards, is thoroughly documented, and follows established best practices.
- **Error Handling and Logging** - Employ robust error handling and logging mechanisms within your custom module. Gracefully handle error conditions, providing clear and informative error messages to users. Leverage Apache's logging facilities to record pertinent information, aiding in debugging efforts during both development and deployment stages. Effective error handling and logging contribute to the stability and maintainability of your custom module.

By meticulously planning and designing your custom module, you establish a strong foundation for its successful implementation. Comprehending Apache's architecture, the Module API, and the development environment is of paramount importance. In the subsequent section, I will delve into the actual development process, guiding you in coding your custom module for Apache2.

For deeper understanding of how to create a custom module it is recommended to visit the official documentation of Apache.

1.8 Module Development

1.8.1 Creating the Module Structure

A well-organized module structure is crucial for maintaining code clarity and modularity. Create a dedicated directory for your custom module and define the appropriate file structure within it. This typically includes source code files, header files, and any additional resources required by your module.

1.8.2 Configuring the Module

To provide flexibility and customization, your custom module may require configuration options. Define the configuration directives that your module will support and specify their behavior. This allows administrators to modify the module's behavior through Apache's configuration files. Ensure proper validation and handling of the configuration values in your module code.

1.8.3 Implementing Module Functionality

Identify the specific functionality you want to add or modify within Apache. Determine the appropriate hooks provided by the Apache module API to intervene and modify the server's behavior at specific points in the request/response lifecycle. Write the necessary code within your module to implement the desired functionality. Utilize the available API functions and data structures provided by Apache to interact with the server effectively.

1.8.4 Error Handling and Logging

Robust error handling and logging are essential aspects of module development. Implement proper error checking and reporting mechanisms in your module code to handle exceptional scenarios gracefully. Use Apache's logging facilities to provide informative and actionable logs to aid in troubleshooting and debugging.

Remember to follow coding best practices, including proper documentation, modular design, and adherence to coding standards.

1.9 Module Integration

1.9.1 Compilation

- Ensure that the Apache development libraries and tools are installed in your development environment.
- Use the appropriate compiler commands to compile the module source code into a shared object (.so) file. Make sure to link against the necessary Apache libraries and include the required headers.

1.9.2 Installation

- Copy the compiled shared object (.so) file to the appropriate directory in your Apache installation. The exact location may vary depending on your system configuration, but it is typically the modules directory.
- Set the correct file permissions and ownership for the module file to ensure that it is accessible by the Apache server process.

1.9.3 Loading the Module

After installing the module, you need to configure Apache to load and use the custom module. Follow these steps:

1. Locate the Apache configuration file, commonly named `httpd.conf` or `apache2.conf`. This file is usually found in the Apache installation directory or the `/etc/apache2` directory.
2. Open the configuration file in a text editor and search for the section where Apache loads modules. This section is typically labeled `LoadModule` or `Dynamic Shared Object (DSO) Support`.
3. Add a new line to load the custom module. The line should follow the format:

```
LoadModule <module_name> <path_to_module_file>
```

Replace `<module_name>` with the desired name for your module and `<path_to_module_file>` with the absolute path to the module shared object (.so) file.

4. Save the configuration file and exit the text editor.

1.9.4 Configuring Apache for the Custom Module

To utilize the functionality provided by the custom module, you may need to configure Apache to enable specific features or behavior. The configuration options for your module should be clearly defined and documented. Follow these steps to configure Apache:

1. Open the Apache configuration file (`httpd.conf` or `apache2.conf`) in a text editor.
2. Locate the appropriate configuration section for your module. This section may already exist or you may need to create it. Consult the Apache documentation for guidance on the configuration options available for your module.
3. Set the desired configuration options based on your requirements. Ensure that the configuration syntax adheres to Apache's guidelines and is error-free.
4. Save the configuration file and exit the text editor.

1.10 Creation of my module

1.10.1 Principle of the program

As inspiration for my module I used the already available modules, `mod_evasive` and `mod_antiloris`. The module implemented in the system performs continuous monitoring of the number of established connections originating from a specific IP address within a specified time period. In this particular example, it tracks if there are 200 connections made within a duration of 10 seconds.

When the number of connections exceeds the allowed threshold, the IP address is added to an array of blocked addresses. Each time a connection attempt is made, the module verifies whether the IP address is in the blocked list. If the address has been on the block list for a duration longer than the time period defined by the system administrator, the connection is allowed to proceed, and the IP address is removed from the blacklist.

However, if the time spent on the block list is less than the predefined duration, the server automatically returns an `HTTP_FORBIDDEN` response, indicating a 403 error code. This response signifies that the server denies access to the requested resource due to the IP address being temporarily blacklisted.

Furthermore, the system continually monitors the number of connections made for `READ` requests as well as `WRITE` requests. The system imposes a limit of 10 connections for both types. This means that a single user is only permitted to establish a maximum of 10 connections to the server, regardless of the request type.

In the event that a user attempts to establish a new connection of the same type (`READ` or `WRITE`) after reaching the limit, the system rejects the new connection request and returns an appropriate error code to the user. However, it is important to note that the previously established connections of the same type will remain functional and unaffected by the limit reached.

1.11 Parameters of the module

The module in question offers several configurable parameters that allow for customization and fine-tuning of its behavior. These parameters are designed to control various aspects of the module's functionality and adapt it to specific requirements. Here is an elaboration of these parameters:

Parameters used for the mitigation of Volumetric Attacks

- `MAX_HITS`: This parameter specifies the maximum allowable number of connections a client can make within a certain time interval. It sets an up-

per limit on the number of connections to prevent abuse or excessive usage. The value of `MAX_HITS` defines the threshold that triggers the monitoring mechanism.

- **INTERVAL:** `INTERVAL` refers to the time interval, measured in seconds, during which the module monitors the number of connections made by clients. It determines the frequency at which the module checks for the total number of connections established by a client.
- **BLOCKING_PERIOD:** `BLOCKING_PERIOD` represents the duration, measured in seconds, for which a user's IP address is blacklisted if they exceed the `MAX_HITS` threshold. During this period, any further connection attempts from the blacklisted IP address are denied. After the `BLOCKING_PERIOD` elapses, the IP address is automatically removed from the blacklist.

Parameters used for the mitigation of Application Layer Attacks

- **ANTILORIS_COUNTER_TYPE_COUNT:** This is a variable that stores the total number of connection type variables used by the module. These connection type variables allow tracking and monitoring of different types of connections, such as `READ` and `WRITE`.
- **ANTILORIS_READ_COUNT_INDEX:** This variable serves as an index within an array that stores two variables. The first variable tracks the number of connections a user has established for reading data. The second variable, `ANTILORIS_WRITE_COUNT_INDEX`, stores the number of connections a user has established for writing data.
- **MAX_CONN:** `MAX_CONN` denotes the maximum number of simultaneous connections that a user can have. It restricts the total number of connections a user can establish at any given time. This parameter helps manage server resources and prevents excessive resource utilization by limiting the concurrent connections from a single user.
- **MAX_READ:** `MAX_READ` represents the maximum number of connections that a user is allowed to establish specifically for reading data. It sets a threshold on the number of read connections to ensure fair usage and prevent excessive resource consumption.
- **MAX_WRITE:** `MAX_WRITE` defines the maximum number of connections that a user can have specifically for writing data. It sets a limit on the number of write connections to prevent abuse or disproportionate usage of server resources.

These parameters provide administrators with flexibility and control over the module's behavior, allowing them to define appropriate limits and thresholds based on the specific requirements and capacity of their system. By adjusting these parameters, administrators can ensure efficient resource allocation and mitigate potential abuses or disruptions caused by excessive connections from individual clients.

1.12 Download and setup of my module

To acquire the module, it can be downloaded from a designated location provided via a specific link. Once downloaded, the user can navigate to the downloaded folder through the terminal by utilizing the "cd" command, followed by the appropriate path.

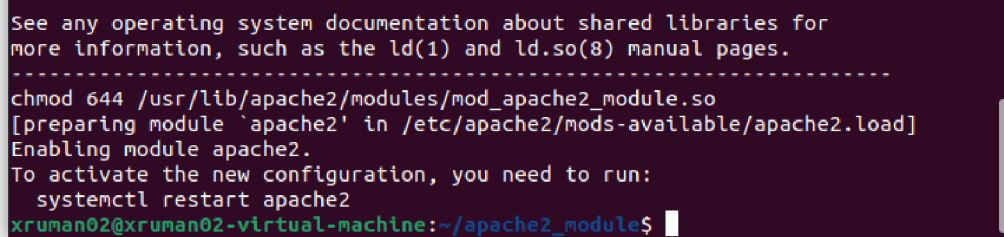
```
sudo cd <path_to_downloaded_file>
```

To proceed with the installation, compilation, and execution of the code, it is advisable to employ administrative privileges by utilizing the "sudo" command. The following command can be executed in the terminal to install, compile, and run the code:

```
sudo apxs -i -a -c mod_apache_module.c
```

The aforementioned command initiates the installation process, integrating the module into the Apache server by utilizing the Apache Extension Tool ("apxs"). The parameters specified include "-i" to install the module, "-a" to add the module to the server's configuration, and "-c" to compile the provided C file, which should be denoted as "mod_apache_module.c".

Upon successful compilation, the terminal should display a corresponding message, indicating that the process has been completed. The specific message can vary, and its content would depend on the implementation and customization of the code, but it is expected to adhere to a specific format, which is as follows:[1.7]



```
See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
-----
chmod 644 /usr/lib/apache2/modules/mod_apache2_module.so
[preparing module `apache2' in /etc/apache2/mods-available/apache2.load]
Enabling module apache2.
To activate the new configuration, you need to run:
  systemctl restart apache2
xruman02@xruman02-virtual-machine:~/apache2_module$
```

Fig. 1.7: Expected displayed message after a successful compilation.

Following the successful compilation and integration of the module, it is necessary to restart the Apache server to ensure the changes take effect. The appropriate command for restarting the Apache server may differ depending on the operating system and server configuration. However, a common command used for this purpose is:

```
systemctl restart apache2
```

By executing this command, the Apache server will be restarted, allowing the newly installed module to be loaded and utilized effectively.

It is important to note that the instructions provided above assume a certain level of familiarity with terminal commands, compilation processes, and server administration. Users should exercise caution and ensure that they have the necessary permissions and understanding of the steps involved before proceeding with the module installation. Additionally, "<path_to_downloaded_file>", should be replaced with the appropriate value.

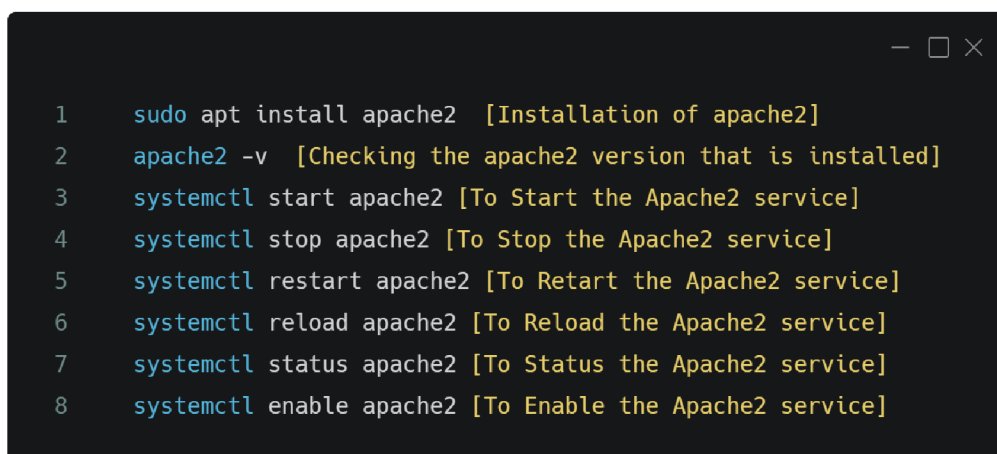
2 Thesis Results

2.1 Research

Detailed research was needed to be able to solve this problem. The first step was to thoroughly learn about the attacks, how they work, what are they used for and how to mitigate them. Using the official documentation of Apache we took a look at the list of already existing modules used for mitigation, after that we investigated them in order of finding their weak points.

2.2 Basic usage of Apache2

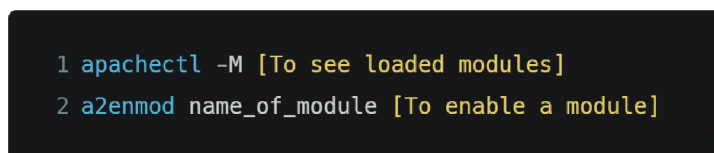
When using linux, the following commands can be run in the terminal to interact with the server:

A terminal window with a dark background and light text. It contains a list of 8 numbered commands for managing the Apache2 service. The commands are: 1. sudo apt install apache2 [Installation of apache2], 2. apache2 -v [Checking the apache2 version that is installed], 3. systemctl start apache2 [To Start the Apache2 service], 4. systemctl stop apache2 [To Stop the Apache2 service], 5. systemctl restart apache2 [To Retart the Apache2 service], 6. systemctl reload apache2 [To Reload the Apache2 service], 7. systemctl status apache2 [To Status the Apache2 service], 8. systemctl enable apache2 [To Enable the Apache2 service].

```
1 sudo apt install apache2 [Installation of apache2]
2 apache2 -v [Checking the apache2 version that is installed]
3 systemctl start apache2 [To Start the Apache2 service]
4 systemctl stop apache2 [To Stop the Apache2 service]
5 systemctl restart apache2 [To Retart the Apache2 service]
6 systemctl reload apache2 [To Reload the Apache2 service]
7 systemctl status apache2 [To Status the Apache2 service]
8 systemctl enable apache2 [To Enable the Apache2 service]
```

Fig. 2.1: Basic commands for the Apache2 server.

To interact with the modules you can use the commands:

A terminal window with a dark background and light text. It contains two numbered commands for interacting with Apache modules: 1. apachectl -M [To see loaded modules], 2. a2enmod name_of_module [To enable a module].

```
1 apachectl -M [To see loaded modules]
2 a2enmod name_of_module [To enable a module]
```

Fig. 2.2: Commands to interact with modules in Apache.

2.2.1 Installing modules on Apache2

Before installing any modules on Apache2 it is best to update the package repository with the following command:

```
$ sudo apt update
```

After finishing the first step, a helper utility needs to be installed with the command:

```
$ sudo apt install apache2-utils
```

The next step is to install the selected module, but the command for this action varies on the selected module.

2.3 Implementation

2.3.1 Mitigation of DoS attacks using mod_evasive

First mod_evasive had to be installed using the command: **\$ sudo apt install libapache2-mod-evasive**

Like most Linux software packages, mod_evasive is controlled by a configuration file. This configuration file can be accessed with any in-terminal text editor, depending on the preferences of the user. After opening the configuration file which is located at /etc/apache2/mods-enabled/ (with the command **\$ sudo vi /etc/apache2/mods-enabled/evasive.conf** for example) I am presented with a list of variables which are:

- **DOSHashTableSize** - Increase this for busier web servers. This configuration allocates space for running the lookup operations. Increasing the size improves the speed at the cost of memory.
- **DOSPageCount** - The number of requests for an individual page that triggers blacklisting. This is set to 2, which is low (and aggressive) – increase this value to reduce false-positives.
- **DOSSiteCount** - The total number of requests for the same site by the same IP address. By default, this is set to 50. You can increase to 100 to reduce false-positives.
- **DOSPageInterval** - Number of seconds for DOSPageCount. By default, this is set to 1 second. That means that if you don't change it, requesting 2 pages in 1 second will temporarily blacklist an IP address.
- **DOSSiteInterval** - Similar to DOSPageInterval, this option specifies the number of seconds that DOSSiteCount monitors. By default, this is set to 1 second. That means that if a single IP address requests 50 resources on the same website in a single second, it will be temporarily blacklisted.

- **DOSBlockingPeriod** - The amount of time an IP address stays on the blacklist. Set to 10 seconds by default, you can change this to any value you like. Increase this value to keep blocked IP addresses in time-out for a more extended period.
- **DOSEmailNotify** - An email address where the user gets sent an email, notifying him about blocking an IP address. However this is advised to not be active, because there may be lots of robots trying to brute force authentication and spamming can occur this way even though they are not committing a targeted attack on the server.
- **DOSSystemCommand** - First, you may have noticed that this option was left disabled as a comment. This command allows you to specify a system command to be run when an IP address is added to the blacklist. You can use this to launch a command to add an IP address to a firewall or IP filter.
- **DOSLogDir** - By default, this is set to write logs to `/var/log/mod_evasive`. These logs can be reviewed later to evaluate client behavior.

The log files need to be saved in a directory, which you can create by the following commands:

```
1 sudo mkdir /var/log/apache/mod_evasive [To create a directory]
2 sudo chown -R apache:apache /var/log/apache/mod_evasive [The owner must be changed to Apache]
3 sudo nano /etc/apache2/mods-enabled/evasive.conf [It is recommended to double-check whether the location
of the directory and the value set in the variable "DOSLogDir" correspond]
```

Fig. 2.3: Creating the log directory for `mod_evasive`.

After configuring these settings, the server should be restarted in order to make sure the module is running. As a next step it can be tested by running a pre-built code which is located at `/usr/share/doc/libapache2-mod-evasive/examples` named **test.pl** [2.4]. When opening the code with a text editor of our choice, a source code is displayed.

Here the amount of requests sent by changing how many times does the **for** cycle run can be changed, this value is set to **100** by default. It also possible change the IP address of the server to be attacked, and the protocol used to carry out the attack. [15]

2.4.1 Testing a UDP flood attack

I tested UDP flood using the hping3 network tool. Running the command:

```
$hping3 --udp --flood 192.168.112.130
```

- **--udp** – is the mode of attack
- **--flood** – the tool sends packets as fast as possible to bring the server down, and the command does not show replies

After running this command, the tool is not able to bring the server down and the server is still accessible for legitimate users.

Testing an ICMP flood attack

The same method is used as in the UDP flood attack simulation with a small change: **\$ hping3 --icmp --flood 192.168.112.130** Where I changed the parameter **--udp** to **--icmp** in order to commit an ICMP attack.

However the outcome does not change. The tool keeps flooding the server but it is not able to cause any harm.

Testing a SYN flood attack

The same method is used as in the previous two situations, but with a small change again:

```
$ hping3 --syn --flood 192.168.112.130
```

Testing a Slowloris attack

I simulated a Slowloris attack using the slowloris toolkit on linux, which can be installed using the commands:

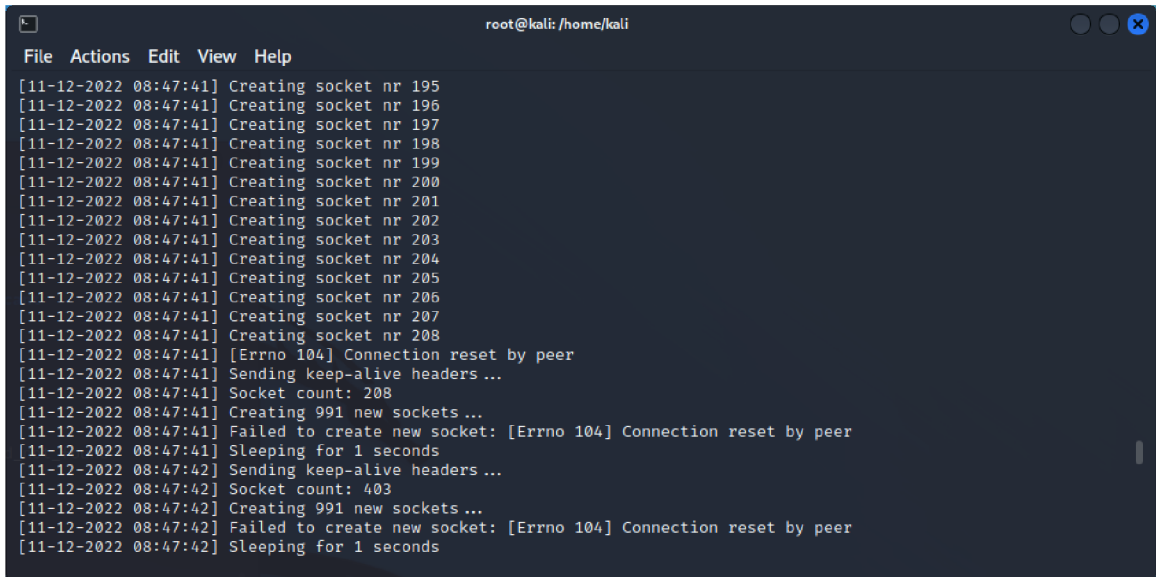
```
sudo apt install python3-pip  
sudo pip3 install slowloris
```

This command generates 1000 connections on the server with a 1 second delay between each connection:

```
slowloris -v -s 1000 192.168.112.130 --sleeptime 1
```

This tool starts occupying each socket, but after reaching a certain amount of keep-alive connections, the server disables this option and the attack is not successful because there are sockets still available for legitimate users. Then the attacker gets an error message [2.6]. Connection reset by peer meaning the server has sent the

attacker an RST packet indicating an immediate dropping of the connection. The attack has failed to bring the server down.



```
root@kali: /home/kali
File Actions Edit View Help
[11-12-2022 08:47:41] Creating socket nr 195
[11-12-2022 08:47:41] Creating socket nr 196
[11-12-2022 08:47:41] Creating socket nr 197
[11-12-2022 08:47:41] Creating socket nr 198
[11-12-2022 08:47:41] Creating socket nr 199
[11-12-2022 08:47:41] Creating socket nr 200
[11-12-2022 08:47:41] Creating socket nr 201
[11-12-2022 08:47:41] Creating socket nr 202
[11-12-2022 08:47:41] Creating socket nr 203
[11-12-2022 08:47:41] Creating socket nr 204
[11-12-2022 08:47:41] Creating socket nr 205
[11-12-2022 08:47:41] Creating socket nr 206
[11-12-2022 08:47:41] Creating socket nr 207
[11-12-2022 08:47:41] Creating socket nr 208
[11-12-2022 08:47:41] [Errno 104] Connection reset by peer
[11-12-2022 08:47:41] Sending keep-alive headers ...
[11-12-2022 08:47:41] Socket count: 208
[11-12-2022 08:47:41] Creating 991 new sockets ...
[11-12-2022 08:47:41] Failed to create new socket: [Errno 104] Connection reset by peer
[11-12-2022 08:47:41] Sleeping for 1 seconds
[11-12-2022 08:47:42] Sending keep-alive headers ...
[11-12-2022 08:47:42] Socket count: 403
[11-12-2022 08:47:42] Creating 991 new sockets ...
[11-12-2022 08:47:42] Failed to create new socket: [Errno 104] Connection reset by peer
[11-12-2022 08:47:42] Sleeping for 1 seconds
```

Fig. 2.6: Error message received by the attacker.

2.5 Tests and Evaluation of my custom module

2.5.1 Datadog

For the purpose of monitoring and visualizing the metrics of the server I used the tool Datadog. Datadog is a monitoring and analytics platform designed to provide comprehensive visibility into the performance and health of various components within an organization’s infrastructure, applications, and services. It allows businesses to collect, analyze, and visualize data from a wide range of sources, including servers, databases, cloud services, containers, and more. I chose this tool for multiple reasons:

- **Centralized Monitoring:** Datadog offers a centralized platform to monitor the entire IT stack, enabling businesses to gain real-time insights into the health, performance, and availability of their systems. It provides a holistic view of infrastructure, applications, and services, allowing teams to identify and resolve issues efficiently.
- **Scalability and Flexibility:** With the increasing complexity and scale of modern IT environments, Datadog is designed to handle large-scale deployments. It can monitor thousands of hosts and automatically adapt to dynamic cloud environments. This scalability and flexibility make it suitable for

organizations of all sizes, from startups to large enterprises.

- **Collaboration and Integration:** Datadog promotes collaboration among teams by providing shared visibility into monitoring data. It supports integrations with various tools and services commonly used in IT environments, such as cloud providers, orchestration platforms, collaboration tools, and more, enabling seamless data exchange and workflow automation.

In order of better understanding of how Datadog works and for its installation and setup I used their official blog as a guide.[16]

2.5.2 Simulating a Slowloris attack

I employed the project hosted at the following GitHub repository, specifically accessible at <https://github.com/gkbrk/slowloris>, as an instrumental tool to replicate and simulate a slowloris attack within the context of my research endeavor.

Download and Setup of Slowloris

To download and set up the before mentioned project enter the command shown below:

```
git clone https://github.com/gkbrk/slowloris.git
```

Upon successfully downloading the tool, it is possible to simulate a slowloris attack using the following command:

```
slowloris <IP to be attacked> -s <number of sockets>  
eg. slowloris 192.168.112.130 -s 200
```

Simulation of an attack

After successfully setting up the Slowloris tool, first I simulated an attack on the server with my module disabled, then I repeated the same steps but with the custom module running this time.

Attack with the module disabled

Upon attacking the server using the before mentioned steps, I measured the capacity of the servers resources such as its memory. The server has reached a state of resource depletion, specifically exhaustion of available memory. The results can be seen on the graphs.

Attack with the module enabled

With the module enabled, the steps mentioned in the previous section were repeated. The module was designed to limit the number of simultaneous connections from a single IP address, hence the attacker was only able to establish a limited number of connections. This resulted in the slowloris attack being unsuccessful. The module logged the denial of the request to establish a connection which can be found in the `/var/log/apache2/error.log` file. An error phrase should be displayed, varying based on the IP of the attacker and the type of connection the attacker exceeded:

```
Connection limit exceeded by user: 192.168.112.128!  
CONNECTION LIMIT FOR READING EXCEEDED
```

2.5.3 Simulating a Flood attack

Attack with my module disabled

With the module disabled an attack is simulated. For the simulation of these types of attacks I used multiple tools. I used the tool provided by `mod_evasive` (**test.pl**). Upon running the source code 200 connections are tried to be establish.

After finishing the test with the `test.pl` source code I also did tests using packages available for Debian devices.

Attack with the module enabled

I enabled my module, and ran the same tests as in the previous section. When the attacker is trying to exceed the limit set by the administrator, the connections get rejected and the IP address from which the requests are coming gets blacklisted. A timestamp is stored in the module which works as a timeout counter. If the attacker tries to establish a connection before the expiry of the timeout counter, the connections get rejected and the timer is reset. In case the timeout expires, new connections can be made to the server from the attacker but only under the same circumstances as before.

Conclusion

This thesis was devoted to the mitigation of DoS attacks on Apache2 servers. The aim of the thesis was to thoroughly make research on the types of attacks and their mitigation.

The first chapter focuses on both the theoretical and practical principle of various DoS attacks. Each attack was individually described. After describing the attacks, there are sections to introduce us to the Apache modules which are used to mitigate these attacks. Furthermore a detailed guide was documented what parts does an Apache module have and how one should be created.

The second chapter is devoted to the implementation and demonstration of the already available Apache modules as well as of my custom module. Multiple attacks were simulated on the server, comparing the capabilities of each module. The available Apache modules were only able to mitigate one type of DoS attack individually, but my module was able to mitigate both the Volumetric Attacks (eg. SYN FLOOD) as well as Application layer attacks (eg. Slowloris) 2.1. The metrics of the server such as its CPU usage, average requests per second, the rate of bytes served by the server were all measured using the combination of `mod_status` and Datadog which is a tool to graphically display the measured data.

	<code>mod_evasive</code>	<code>mod_antiloris</code>	<code>mod_apache2</code>
Volumetric Attacks	✓	X	✓
Logical Attacks	X	✓	✓

Tab. 2.1: Comparison of the mitigation capabilities of the modules.

Bibliography

- [1] *What is a denial of service attack (DoS) [online]. [cit. 2022-12-9]. Dostupné z: <<https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos>>*
- [2] *DDoS Attacks [online]. [cit. 2022-12-10]. Dostupné z: <<https://www.imperva.com/learn/ddos/ddos-attacks/>>*
- [3] *Method of a HTTP flood attack [online]. [cit. 2022-12-10]. Dostupné z: <https://www.cloudflare.com/img/learning/ddos/udp-flood-ddos-attack/udp-flood-attack-ddos-attack-diagram.png>*
- [4] *UDP flood attack [online]. [cit. 2022-12-10]. Dostupné z: <<https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/>>*
- [5] *HTTP flood attack [online]. [cit. 2022-12-10]. Dostupné z: <https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/>*
- [6] *Method of a HTTP flood attack [online]. [cit. 2022-12-10]. Dostupné z: <https://www.cloudflare.com/img/learning/ddos/http-flood-ddos-attack/http-flood-attack.png>*
- [7] *What is an ICMP Flood Attack [online]. [cit. 2022-12-10]. Dostupné z: <https://www.netscout.com/what-is-ddos/icmp-flood>*
- [8] *Method of an ICMP flood attack [online]. [cit. 2022-12-10]. Dostupné z: <https://www.cloudflare.com/img/learning/ddos/ping-icmp-flood-ddos-attack/ping-icmp-flood-ddos-attack-diagram.png>*
- [9] *SYN flood attack [online]. [cit. 2022-12-12]. Dostupné z: <https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/>*
- [10] *Method of a SYN flood attack [online]. [cit. 2022-12-12]. Dostupné z: <https://www.cloudflare.com/img/learning/ddos/syn-flood-ddos-attack/syn-flood-attack-ddos-attack-diagram-2.png>*
- [11] *What is a Slowloris Attack? [online]. [cit. 2023-4-3]. Dostupné z: <https://www.netscout.com/what-is-ddos/slowloris-attacks>*
- [12] *Slowloris DDoS attack [online]. [cit. 2022-4-4]. Dostupné z: <https://www.cloudflare.com/img/learning/ddos/ddos-slowloris-attack/slowloris-attack-diagram.png>*

- [13] *R.U.D.Y. (R-U-Dead-Yet?) [online]. [cit. 2023-4-4]. Dostupné z: <https://www.imperva.com/learn/ddos/rudy-r-u-dead-yet/>*
- [14] STEIN, Lincoln a Doug MACEACHERN. *Writing apache modules with Perl and C*. Sebastopol, 1999. ISBN 15-659-2567-X.
- [15] *List of Apache2 packages [online]. [cit. 2022-4-7]. Dostupné z: <https://packages.debian.org/search?searchon=names&keywords=libapache2>*
- [16] *How to monitor Apache web server with Datadog [online]. [cit. 2022-4-7]. Dostupné z: <https://www.datadoghq.com/blog/monitor-apache-web-server-datadog/>*
- [17] *Developing modules for the Apache HTTP Server 2.4 [online]. [cit. 2022-4-5]. Dostupné z: <https://httpd.apache.org/docs/2.4/developer/modguide.html#page-header>*
- [18] ROBINSON, Scott. *Protect your Apache server from DoS attacks [online]. [cit. 2022-12-12]. Dostupné z: <https://www.techrepublic.com/article/protect-your-apache-server-from-dos-attacks/>*
- [19] *Types of Denial of Service Attacks [online]. [cit. 2022-12-12]. Dostupné z: https://developer.okta.com/books/api-security/dos/what/#_1-application-layer-flood*
- [20] BAJPAI, Arpit. *Securing Apache, DoS & DDoS Attacks [online]. [cit. 2022-12-12]. Dostupné z: <https://www.opensourceforu.com/2011/04/securing-apache-part-8-dos-ddos-attacks/>*
- [21] HELME, Scott. *Mitigating a HTTP GET DoS attack [online]. [cit. 2022-12-12]. Dostupné z: <https://scotthelme.co.uk/mitigating-http-get-dos-attack/>*
- [22] MUSCAT, Ian. *Mitigate Slow HTTP GET/POST Vulnerabilities in the Apache HTTP Server [online]. [cit. 2022-12-12]. Dostupné z: <https://bit.ly/3MgqZEC>*
- [23] *Slow Loris Attack [online]. [cit. 2022-12-12]. Dostupné z: <https://www.youtube.com/watch?v=XiFkyR35v2Y>*
- [24] *Defending Against DoS/DDoS Attacks in Apache Server With mod_evasive Module [online]. [cit. 2022-12-12]. Dostupné z: <https://codingshower.com/apache-mod-evasive/>*

- [25] MYERSCOUGH, Damian. *Protecting Apache against DOS attack with mod_evasive* [online]. [cit. 2022-12-12]. Dostupné z: <<https://www.suse.com/c/protecting-apache-against-dos-attack-modevasive/>>
- [26] PEDAMKAR, Priya. *Types of DOS Attacks* [online]. [cit. 2022-12-12]. Dostupné z: <<https://www.educba.com/types-of-dos-attacks/>>
- [27] FUND, Fraida. *Layer 7 DoS attack with slowloris* [online]. [cit. 2022-12-12]. Dostupné z: <<https://witestlab.poly.edu/blog/slowloris/>>
- [28] *How to best defend against slowloris* [online]. [cit. 2022-12-12]. Dostupné z: <<https://bit.ly/3IikiRt>>
- [29] *How to prevent DoS attacks against Apache - Practical Linux security* [online]. [cit. 2022-12-12]. Dostupné z: <<https://www.youtube.com/watch?v=XoNbgqgP-dc>>
- [30] *UNDERSTANDING THE THREE MAIN TYPES OF DOS ATTACKS* [online]. [cit. 2022-12-12]. Dostupné z: <<https://cwatch.comodo.com/types-of-dos-attack.php>>
- [31] DELGADO, Carlos. *How to perform a DoS attack "Slow HTTP" with SlowHTTPTest* [online]. [cit. 2022-12-12]. Dostupné z: <<https://bit.ly/42KH78p>>
- [32] *How to Launch an Untraceable DoS Attack with hping3* [online]. [cit. 2022-12-12]. Dostupné z: <<https://www.hackingloops.com/hping3/>>
- [33] *Ping Flooding DoS Attack in a Virtual Network* <<https://sandilands.info/sgordon/ping-flooding-dos-attack-in-a-virtual-network>>
- [34] *DOS Flood With hping3* <<https://linuxhint.com/hping3/>>
- [35] JOHN, Veena K. *How To Protect Against DoS and DDoS with mod_evasive for Apache on CentOS 7* [online]. [cit. 2022-12-12]. Dostupné z: <<https://bit.ly/3Wbr8xD>>
- [36] IMPE, Koen Van. *Defending Against Apache Web Server DDoS Attacks* [online]. [cit. 2022-12-12]. Dostupné z: <<https://bit.ly/3o8wEVo>>

- [37] VIJAYAKUMAR, Lakshmi. *Prevent DDoS in Apache – Steps to safeguard your web server from DDoS* [online]. [cit. 2022-12-12]. Dostupné z: <<https://bobcares.com/blog/apache-prevent-ddos/>>
- [38] *Installing Mod_Antiloris To Mitigate SlowLoris DOS Attack* [online]. [cit. 2022-12-12]. Dostupné z: <<https://www.bullten.com/knowledgebase/3/Installing-ModAntiloris-To-Mitigate-SlowLoris-DOS-Attack.html>>
- [39] MARSHALL, John. *Slowloris Attack Defense & Mitigation* [online]. [cit. 2022-12-12]. Dostupné z: <<https://www.fixscam.com/slowloris/>>
- [40] BROUCKE, Seppe. *Slowloris And Mitigations For Apache* [online]. [cit. 2022-12-12]. Dostupné z: <<https://blog.macuyiko.com/post/2011/slowloris-and-mitigations-for-apache.html>>
- [41] *Mod_antiloris | Mitigate Slowloris DoS attacks* [online]. [cit. 2022-12-12]. Dostupné z: <https://kandi.openweaver.com/c/Deltik/mod_antiloris#Code-Snippets>
- [42] *Apache Module mod_reqtimeout. Apache Module mod_reqtimeout* [online]. [cit. 2022-12-12]. Dostupné z: <https://httpd.apache.org/docs/2.4/mod/mod_reqtimeout.html>
- [43] *Module mod_reqtimeout* [online]. [cit. 2022-12-12]. Dostupné z: <https://www.ibm.com/docs/en/i/7.2?topic=ssw_ibm_i_72/rzaie/rzaiemod_reqtimeout.html>
- [44] *The most important steps to take to make an Apache server more secure* [online]. [cit. 2022-12-12]. Dostupné z: <<https://bit.ly/3BvSmFM>>
- [45] *How to configure mod_reqtimeout in Apache2* [online]. [cit. 2022-12-12]. Dostupné z: <<https://serverfault.com/questions/507542/how-to-configure-mod-reqtimeout-in-apache2>>
- [46] *Developing modules for the Apache HTTP Server 2.4* [online]. [cit. 2023-05-17]. Dostupné z: <https://httpd.apache.org/docs/2.4/developer/modguide.html#page-header>

Symbols and abbreviations

DoS	Denial of Service
RUDY	R-U-Dead-Yet?
API	Application programming interface
CPU	Central Processing Unit
IP	Internet Protocol
SYN	synchronization
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ICMP	Internet Control Message Protocol
HTTP	Hypertext Transfer Protocol
DDoS	Distributed Denial-of-Service
ACK	Acknowledgment
DNS	Domain Name System
NTP	Network Time Protocol

List of appendices

A	Graphs of Apache metrics during attacks	52
B	Content of the electronic attachment	57

A Graphs of Apache metrics during attacks

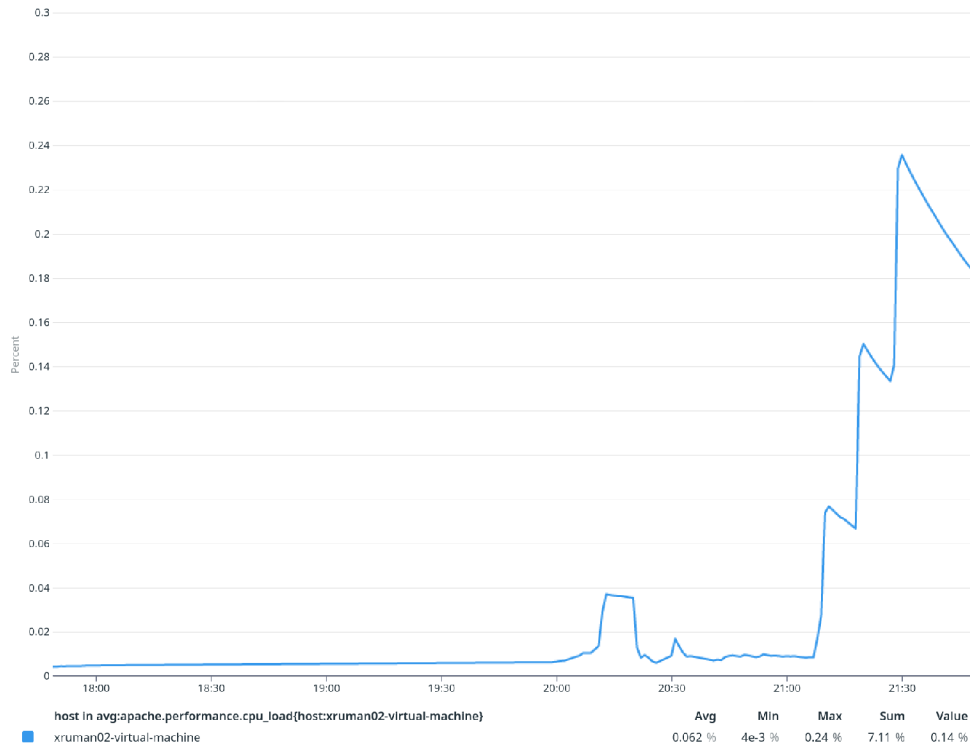


Fig. A.1: Apache CPU usage during a flood attack with my module turned off.

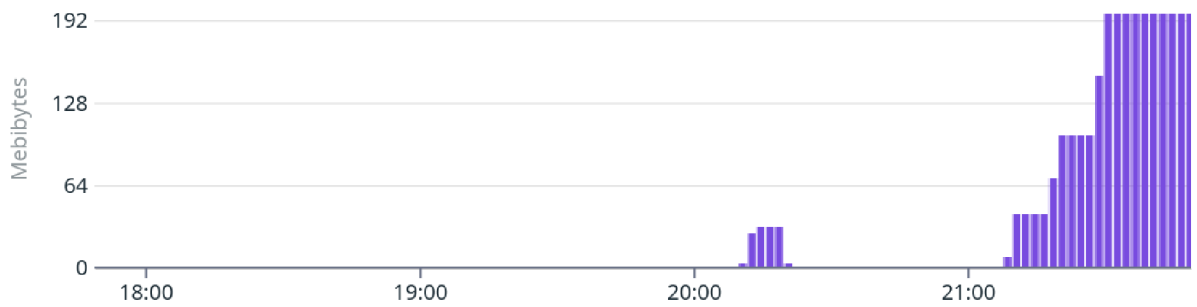


Fig. A.2: Bytes served by the server during a flood attack with my module turned off.

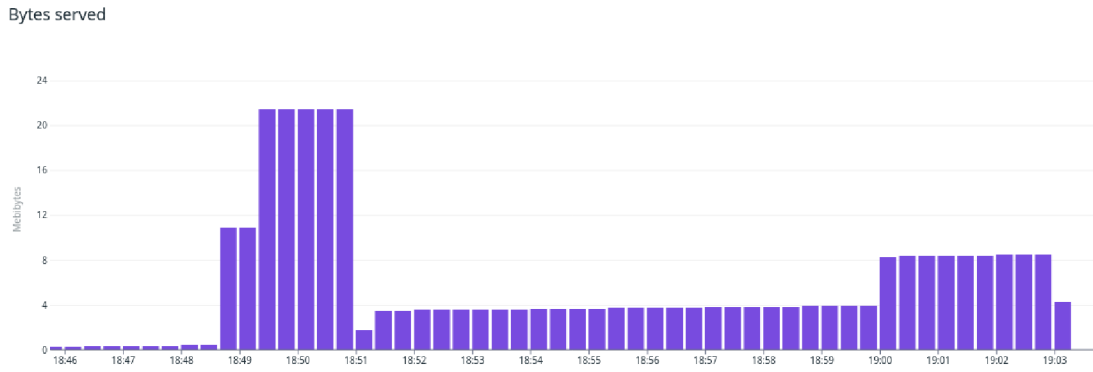


Fig. A.3: Bytes served by the server during a flood attack with my module turned on.

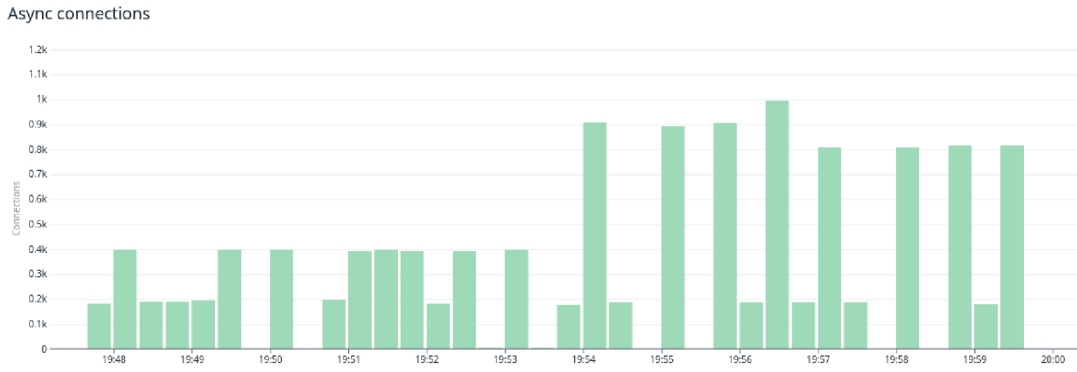


Fig. A.4: Connections closed during a slowloris attack with my module turned on.

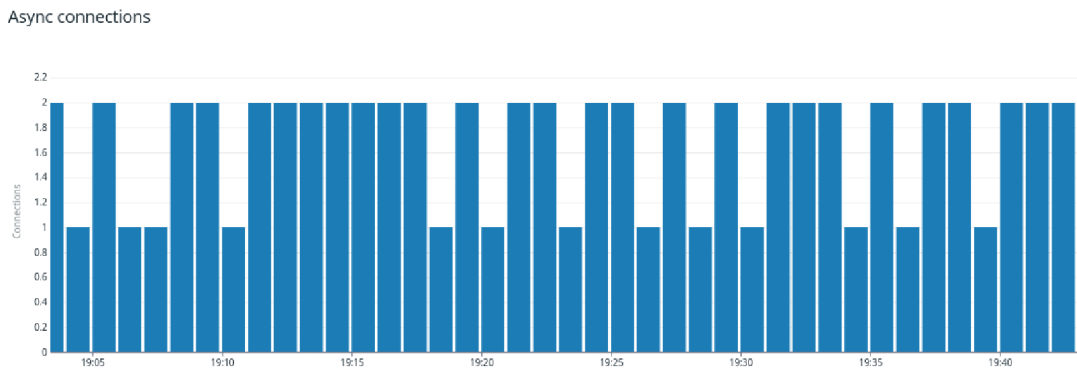


Fig. A.5: Connections during a slowloris attack with my module turned on.

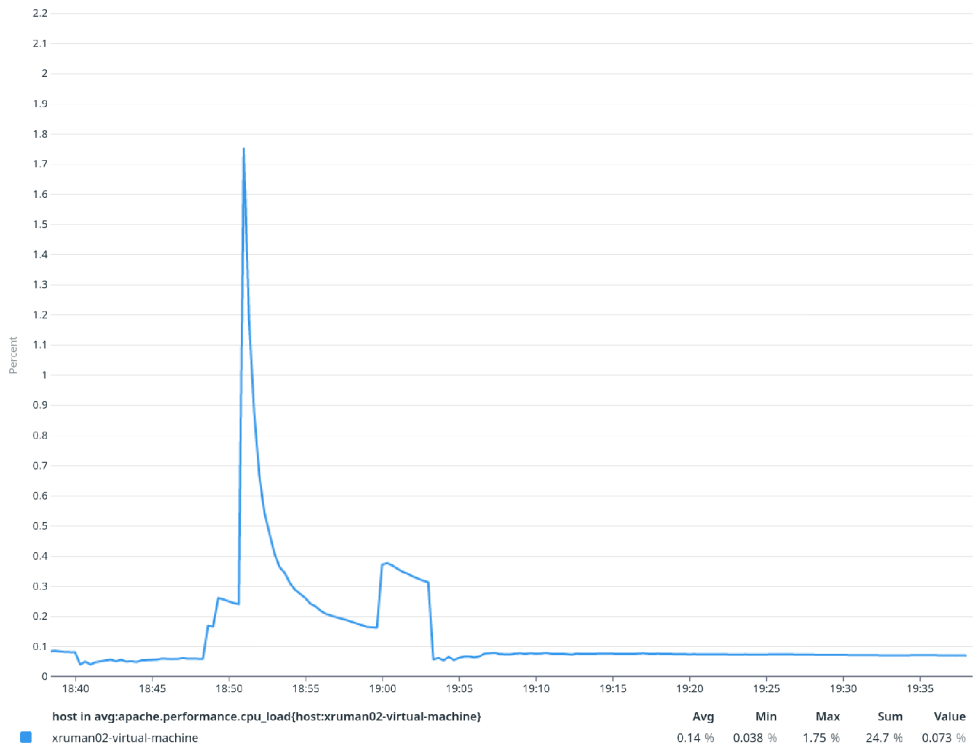


Fig. A.6: CPU usage of server during an attack with my module turned on.



Fig. A.7: Amount of hits on the server with my module turned off.

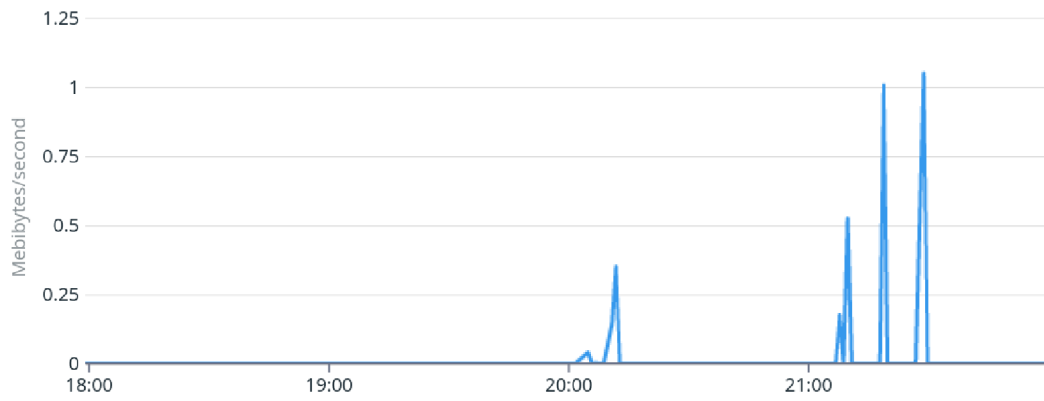


Fig. A.8: Rate of bytes on the server during a flood attack with my module turned on.



Fig. A.9: Rate of bytes served by the server with my module turned off.

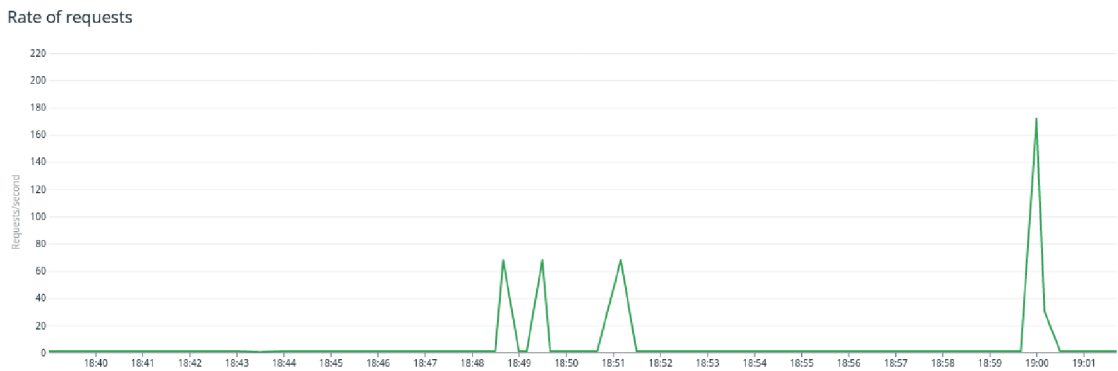


Fig. A.10: Rate of requests during a flood attack with my module turned off.

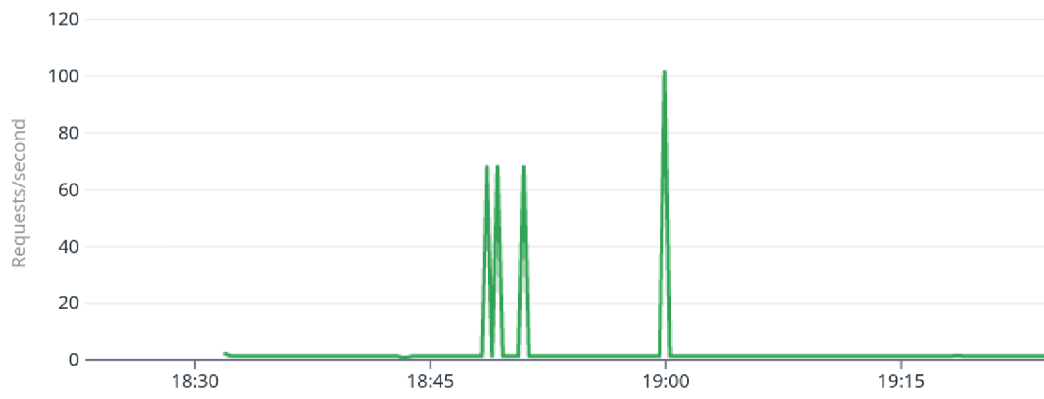


Fig. A.11: Rate of requests during a flood attack with my module turned on.

B Content of the electronic attachment

```
/ ..... root directory of the attached archive
├── apache2_module
│   ├── .git ..... root directory for Git
│   ├── .libs ..... Folder to store artifacts generated during the build process
│   │   ├── mod_apache2_module.la ..Metadata and configuration about the module
│   │   ├── mod_apache2_module.lai .....Index of symbols and their offsets
│   │   ├── mod_apache2_module.o ..... Compiled object code
│   │   └── mod_apache2_module.so ..... Represents the compiled module
│   ├── .vscode ..... Folder for configuration files
│   │   ├── c_cpp_properties.json ..... Settings related to the C language support
│   │   ├── settings.json ..... Project specific settings and configuration
│   │   └── tasks.json ..Definition of custom tasks executable from within VS Code
│   ├── .deps .....Dependencies-related files and information
│   ├── Makefile .....Defines and manages the build process of the project
│   ├── mod_apache2_module.c ..... Source code of the project
│   ├── mod_apache2_module.la
│   ├── mod_apache2_module.lo ..... Helps manage libraries and their compilation
│   ├── mod_apache2_module.o
│   ├── mod_apache2_module.so
│   └── modules.mk ..Build instructions and configurations for compiling and linking
│       the module
├── git_link.txt .....Link to the GitHub repository of the project
└── Bakalárska_práca_231273 .....Electronic version of this document
```