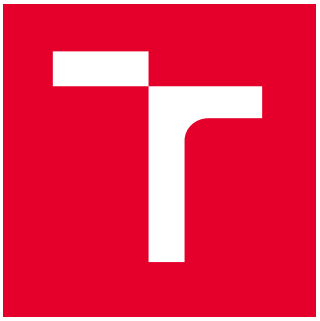


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

MODUL PRO TESTOVÁNÍ PROTOKOLU HTTP/2 V PROGRAMU JMETER

JMETER MODULE FOR HTTP/2 PROTOCOL TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Martin Smetana**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Michael Jurek**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Martin Smetana ***ID:*** 230668

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Modul pro testování protokolu HTTP/2 v programu JMeter

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout, implementovat a verifikovat modul na testování protokolu HTTP/2 v programu Apache JMeter. Dále vytvořit modul pro testování existujících DoS útoků na protokol HTTP/2. V teoretické části se seznamte s protokolem HTTP/2 a platformou Apache JMeter. Dále prostudujte známé útoky na protokol HTTP/2 se zaměřením na Slow DoS. V praktické části realizujte experimentální pracoviště obsahující webový server a vývojové prostředí s JMeterem. Dále implementuje nový modul v jazyce Java pro Apache JMeter, který bude umožňovat provést testování HTTP(S)/2 spojení, tvorbu a spuštění vybraných DoS útoků. Výsledky testovacích scénářů analyzujte a přehledně zobrazte v patřičných vizualizacích.

DOPORUČENÁ LITERATURA:

[1] POLLARD, Barry. HTTP/2 in Action. Manning, 2019. ISBN 9781617295164.,

[2] GRABOVSKY, Stepan, Petr CIKA, Vaclav ZEMAN, Vlastimil CLUPEK, Milan SVEHLAK a Jan KLIMES. Denial of Service Attack Generator in Apache JMeter. 2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT) [online]. IEEE, 2018, 2018, 1-4 [cit. 2022-09-08]. ISBN 978-1-5386-9361-2. Dostupné z: doi:10.1109/ICUMT.2018.8631212.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Michael Jurek

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce pojednává o komunikačních protokolech a komplexně popisuje útoky typu Slow DoS a DDoS. Zabývá se problematikou protokolu HTTP/2, popisem jeho rámců a využité komprese hlavičky HPACK. Dále se zabývá tvorbou Slow DoS útoků v nástroji JMeter, konfigurací serveru pro podporu HTTP/2 protokolu a následnému otestování vytvořených modulů na Apache serveru ve verzi 2.4.52.

KLÍČOVÁ SLOVA

http2, jmeter, slowdos, slowread, slowpost, slowheaders, slowpreface, slowsettings, hpack, dos, apache

ABSTRACT

The bachelor thesis deals with communication protocols and describes Slow DoS and DDoS attacks in a comprehensive way. It deals with HTTP/2 protocol, description of its frames and used HPACK header compression. It also deals with creating Slow DoS attacks in JMeter, configuring the server to support HTTP/2 protocol and then testing the created modules on Apache server version 2.4.52.

KEYWORDS

http2, jmeter, slowdos, slowread, slowpost, slowheaders, slowpreface, slowsettings, hpack, dos, apache

SMETANA, Martin. *Modul pro testování protokolu HTTP/2 v programu JMeter*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 66 s. Bakalářská práce. Vedoucí práce: Ing. Michael Jurek

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Martin Smetana
VUT ID autora:	230668
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Modul pro testování protokolu HTTP/2 v programu JMeter

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Michaelu Jurkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

V době neustále se zvyšující náročnosti výpočetní techniky je téměř žádoucí být obeznámen s možnými útoky, které mohou na webu nastat, pokud web podporuje HTTP/2 (Hyper Text Transfer Protocol ver. 2) komunikaci.

První kapitola se věnuje komunikačním modelům, zejména tedy ISO/OSI (International Organization for Standardization/Open Systems Interconnection) a také TCP/IP (Transmission Control Protocol over Internet Protocol) a jejich návaznost k protokolu HTTP/2, také popisuje jak probíhá navázání spojení pomocí „three-way handshake“.

Práce se ve druhé kapitole věnuje základním rozdílům mezi DoS (Denial of Service) a DDoS (Distributed Denial of Service), jejich charakteristikami a následným dělením těchto útoků.

Třetí kapitola je zaměřena na pomalé útoky v protokolu HTTP/2 a jejich kompletním teoretickým popisem.

Čtvrtá kapitola se věnuje historii obecně HTTP (Hyper Text Transfer Protocol) protokolu, jak se v čase vyvíjel a následně se věnuje historickým verzím tohoto protokolu.

Pátá kapitola se věnuje HTTP/2 protokolu, jsou zde popsány zejména, jak se na tuto verzi přešlo, jeho historii, jak tato verze vznikala, jsou zde popsány jednotlivé rámce, chybové kódy nebo navázání spojení.

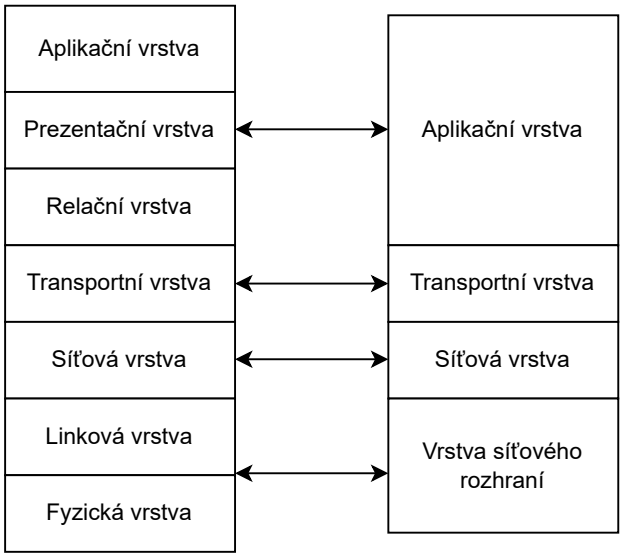
Šestá kapitola se věnuje formátu HPACK (Header Compression for HTTP/2) pro kompresi hlaviček v protokolu HTTP/2 je zde uvedeno, jakým způsobem má kompresní tabulka vzniknout.

Sedmá kapitola se věnuje obecnému popisu nástroje JMeter. Věnuje se definici, jak probíhá vývoj nového modulu v nástroji JMeter, realizaci webového serveru Apache, který podporuje HTTP/2 verzi a následné otestování na existujícím modulu od firmy BlazeMeter.

V poslední osmé kapitole se teoretické znalosti z tvorby pomalých DoS modulů ve třetí kapitole aplikují prakticky a bude tedy vytvořeno testovací prostředí, kde se naprogramuje těchto pět pomalých útoků na protokolu HTTP/2 a následně budou otestovány na Apache serveru a výstup bude ve formě tabulky a grafu naměřených hodnot.

1 Komunikační modely

Pro pochopení co je HTTP protokol a jak pracuje je nutné specifikovat konkrétní komunikační modely. HTTP je obvykle zprostředkováván komunikačním protokolem TCP/IP. Obrázek 1.1 specifikuje, jak vychází TCP/IP z modelu ISO/OSI. [1]



Obr. 1.1: Zobrazení přechodu TCP/IP z ISO/OSI.

Z důvodu složitosti internetu jako takového, je potřeba jej rozdělit do vrstev, kde každá se stará o určitou část internetu. Vrstvy komunikují mezi sebou, každá vrstva musí být schopna rozpoznat jaký tok dat jde ze sousední vrstvy a vykonat s ním tu činnost, kterou má tato vrstva za úkol. Vrstva může komunikovat oběma směry. V další kapitole bude blíže popsán referenční model ISO/OSI. [1]

1.1 Referenční model ISO/OSI

V rámci modelu ISO/OSI tento komunikační model obsahuje 7 vrstev – aplikační vrstvu, prezentační vrstvu, relační vrstvu, transportní vrstvu, síťovou vrstvu, linkovou (někdy též spojovou) vrstvu a poslední fyzickou vrstvu. [2]

- **Fyzická** vrstva se velmi zjednodušeně zabývá využitou přenosovou technologií (např. kabel nebo wi-fi), typem konektoru a data na této vrstvě se nazývají bity. Bity nabývají logické hodnoty 0 nebo 1.
- **Linková (spojová)** vrstva se zabývá zabezpečením rámců (tzn. data zde jsou rámce).

- **Síťová** vrstva se zabývá směrováním paketů, využívá k tomu síťové adresy (IP adresy), aby paket došel do koncového zařízení. Data na této vrstvě jsou pakety.
- **Transportní** vrstva zaručuje doručení segmentů ve správném pořadí, segmenty jsou očíslovány a díky tomu je zaručeno doručení ve správném pořadí, pokud dojdou špatně, přenos se opakuje. Data se zde nazývají segmenty.
- **Relační** vrstva již z názvu zajišťuje vytvoření relace mezi dvěma zařízeními, zaručuje udržování této relace po celou dobu trvání spojení mezi dvěma zařízeními a přenášení dat v rámci této relace. V relaci je možné vytváření kontrolních bodů, které umožňují v případě výpadku přenosu, aby se pokračovalo v přenosu od tohoto kontrolního bodu.
- **Prezentační** vrstva má za úkol kódování, zašifrování a kompresi přenášených dat mezi dvěma zařízeními.
- **Aplikační** vrstva je jako jediná používaná přímo koncovým uživatelem, jde přímo o klienty na email, webové prohlížeče nebo přenos souborů. Protokoly, které na této vrstvě pracují jsou právě například zmiňovaný HTTP, který pracuje na portu 80, a nebo DNS (Domain Name Server), který pracuje na portu 53. [1]

1.2 Rodina protokolů TCP/IP

Rodina protokolů TCP/IP (Transmission Control Protocol/Internet Protocol) je poněkud méně obsáhlá oproti ISO/OSI rozebraném v předešlé kapitole. Jak lze vidět na obrázku 1.1, tak:

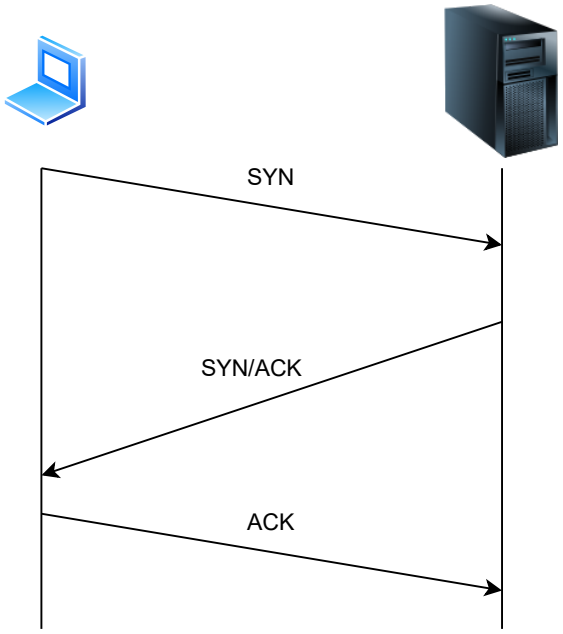
- **Vrstva síťového rozhraní**, která je úplně dole, tak shlukuje fyzickou a linkovou vrstvu. Z toho tedy plyne, že má podobný úkol jako fyzická a linková vrstva v rámci komunikačního modelu *ISO/OSI*, zpracovává datagram z internetové vrstvy, aby je zaslala jinému zařízení přes médium, kterým může být kabel nebo bezdrátová komunikace.
- **Síťová** vrstva přidává další hlavičku do zprávy, kterou přijala z transportní vrstvy. Nejdůležitější věc, kterou do této zprávy přidává je zdrojová a cílová IP (Internet Protocol) adresa, která je unikátní virtuální číslo, které umožňuje vyhledat zařízení v rámci sítě.
- **Transportní** vrstva přijímá zprávy z aplikační vrstvy, rozděluje je na menší pakety a přidává hlavičky a posílá je dolů na síťovou vrstvu. Transportní vrstva využívá čísla portu a díky tomu identifikuje jaký proces příchozí zpráva vykonává.
- **Aplikační** vrstva shlukuje z referenčního modelu ISO/OSI tři vrstvy, a sice aplikační, prezentační a relační. Vytváří komunikaci mezi počítačovými pro-

gramy nižšími vrstvami protokolu. Programy mohou využívat protokoly aplikační vrstvy a žádat o vykonání odlišných akcí. [3]

1.2.1 Three-Way Handshake

Jedná se o metodu používanou v rámci TCP/IP pro vytvoření spojení mezi klientem a serverem. Využívá se pro přenos dat mezi dvěma zařízeními. Je podporovaná komunikace mezi webovým serverem a klientem.
Sestavení spojení:

- **První krok** – sestavení spojení mezi serverem a klientem, server otevře porty, které přijímají a inicializují nové spojení. Klient zasílá SYN (Synchronize Sequence Number) paket serveru, jedná se o náhodnou sekvenci čísel, které klient chce použít pro komunikaci.
- **Druhý krok** – ve chvíli, kdy server příjme SYN paket zasílá klientovi jako potvrzení zaslaného SYN paketu paket ACK (Acknowledge Sequence Number) nebo SYN/ACK paket. Tento paket obsahuje dvě sekvenční čísla. První číslo je číslo paketu ACK, které je o jedna větší než číslo, které si zvolil klient v rámci paketu SYN při sestavování komunikace. Další číslo je číslo paketu SYN zasílaného od serveru ke klientovi, kde si server volí náhodné číslo.
- **Třetí krok** – klient obdrží od serveru SYN/ACK paket paket a reaguje na to zasláním ACK paketu, jehož číslo bude o jedno větší než číslo SYN paketu zaslaného ze serveru. A po tomto kroku je celé spojení sestaveno. [4]

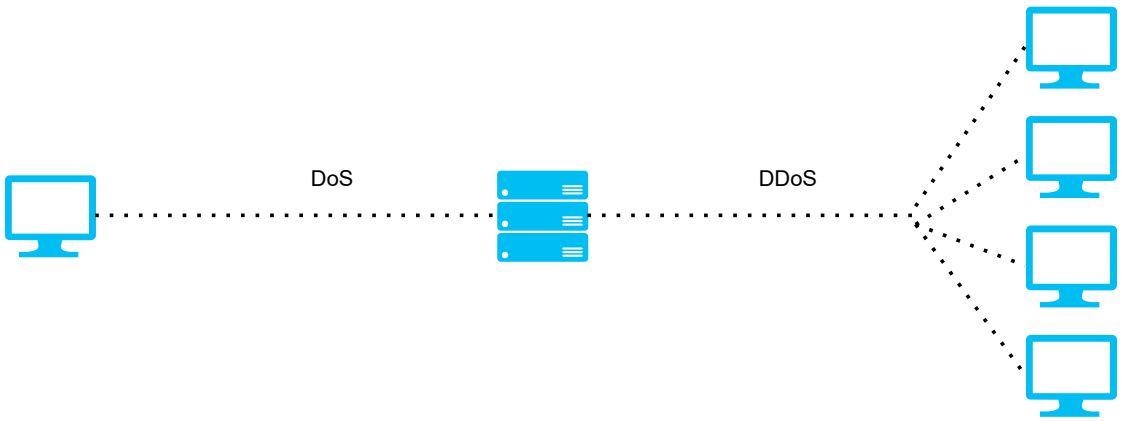


Obr. 1.2: Three-way handshake.

2 Útoky DoS a DDoS

Oba výše zmíněné útoky jak DoS a DDoS mají za cíl znepřístupnit službu např. webovou, může být i spoustu jiných služeb, které má útočník za cíl shodit (změnit jejich dostupnost). Ve chvíli, kdy se webová stránka, nebo jiná uživatelská služba tváří nedostupně je velmi pravděpodobné, že byla cílem DDoS útoku. Oba typy útoku DoS i DDoS se snaží vyprodukovat takové množství požadavků, které nebude schopen cílový server zpracovat.

Klíčový rozdíl mezi těmito dvěma typy je ve způsobu provedení, útok DoS je prováděn z jedné stanice a útok DDoS přes botnet (síť počítačů infikovaných speciálním softwarem, který umožňuje řízení této sítě). Na obr. 2.1 lze vidět hlavní logiku a způsob provedení těchto dvou útoků. [17]



Obr. 2.1: DoS vs DDoS.

2.1 Hlavní rozdíly mezi DoS a DDoS

- **Velikost provozu** – útoky DDoS jsou zpravidla schopny vygenerovat řádově vyšší provoz než útoky DoS, neboť jsou distribuované více počítači zaráz, viz popis výše.
- **Provedení útoku** – v rámci útoku DDoS jsou počítače koordinovány přes „command-and-control“ server, který umožňuje útočníkovi jejich vzdálenou správu. Naproti tomu útoky DoS jsou spouštěny z jedné stanice.
- **Dohledávání zdroje útoku** – v případě DDoS útoku je to prakticky nemožné, neboť zdrojové IP adresy jsou adresy stovek tisíc počítačů.
- **Rychlost útoku** – Rychlost útoku DDoS je daleko větší, neboť je distribuován mezi více počítačů (útočí více zařízení, útok je větší). [17]

2.2 Dělení DoS útoků

Lze je rozdělit na tři základní typy:

1. **Záplavové** – cílem tohoto typu útoku je přetížit celou šířku pásma, tento útok je zároveň nejjednodušší, protože jeho provedení je otázka vygenerování co největší množství paketů a následné zahlcení sítě, udává se v jednotkách bps (bits per second), mezi nejznámější útoky v této kategorii patří UDP Flood (User Datagram Protocol Flood), ICMP Flood (Internet Control Message Protocol)...
2. **Protokolové** – využívají zranitelnosti nějakého protokolu, např. TCP/IP, velikost se udává v pps (packets per second). Konkrétní útoky tohoto typu jsou např. SYN Flood, Ping Of Death...
3. **Útoky na aplikační vrstvě** – tento provoz se tváří legitimně, nicméně se jedná o útok na veřejně přístupnou aplikaci, rozdíl oproti záplavovým útokům je v tom, že server může mít dostatečnou šířku pásma, ale zpracovává nelegitimní provoz, velikost útoku se udává v rps (requests per second). Příkladem může být HTTP Flood. [18] [19]

2.2.1 Ping of Death

V následující části bude popsán útok Ping of Death. Jedná se o DDoSútok, který má za cíl poškodit nebo znepřístupnit část služby, případně službu celou a docílí toho tím, že na server posílá buď poškozené pakety nebo pakety které jsou příliš velké pomocí jednoduchého příkazu ping. Nejznámější útok tohoto typu je ICMP Flood, který posílá dotazy na server bez čekání na odpověď, čímž dochází k extrémnímu zahlcení daného serveru.

Útok je velmi jednoduchý, neboť útočník nepotřebuje žádnou širokou znalost systému, stačí mu k tomu pouze IP adresa. Další výhodou takto vedeného útoku je snadné podvrhnutí zdrojové IP adresy. A také flexibilita útoku, jelikož jej může vykonat kterékoliv zařízení, které je schopné zasílat ping na server. [27]

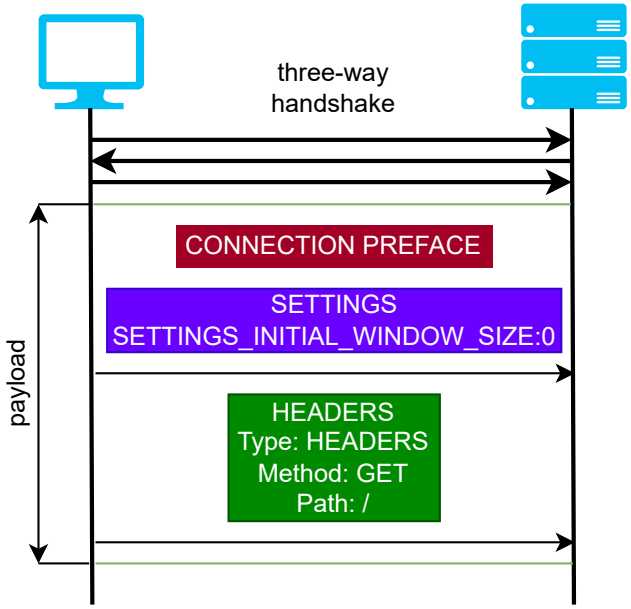
3 HTTP/2 Slow DoS útoky

Jde o útok na aplikační vrstvě. Hlavním rozdílem pomalých DoS útoků od záplavových a protokolových je ten, že nevyužívají hrubou sílu, tedy nesnaží se převýšit šířku pásma, ale útok je konstruován tak, aby se nedal odlišit od legitimního provozu. Další ze základních rozdílů je ten, že útok lze spustit pouze jako DoS, není potřeba většího množství útoků a bývá proveden se stejnými následky. Z hlediska útočníka je tento typ útoku nejvíce dostupný z hlediska výpočetní techniky. Některé typy pomalých DoS útoků budou blíže popsány níže v textu. [20]

3.1 HTTP/2 Slow READ

Průběh útoku spočívá v tom, že se naváže „three-way“ handshake a útočník zasílá HTTP/2 payload, který obsahuje rámec `SETTINGS` s příznakem `SETTINGS_INITIAL_WINDOW_SIZE` a nastaví tento příznak na nulovou hodnotu. Následně posílá v rámci `HEADERS` zbytek žádosti typu GET.

Příznak `SETTINGS_INITIAL_WINDOW_SIZE` říká serveru, jaký objem dat je klient schopen přijímat, jelikož je tento příznak nastaven na nulovou hodnotu, klient indikuje, že v tuto chvíli v rámci spojení není schopen zasílat žádná data. Server tedy vyčkává na rámec `WINDOW_UPDATE` a po tuto dobu je stále navázané spojení. Útočník tento rámec pochopitelně nezašle, neboť cílem útoku je, nechat spojení otevřené po co nejdelší dobu, aby se co nejvíce zatížil server. [21]



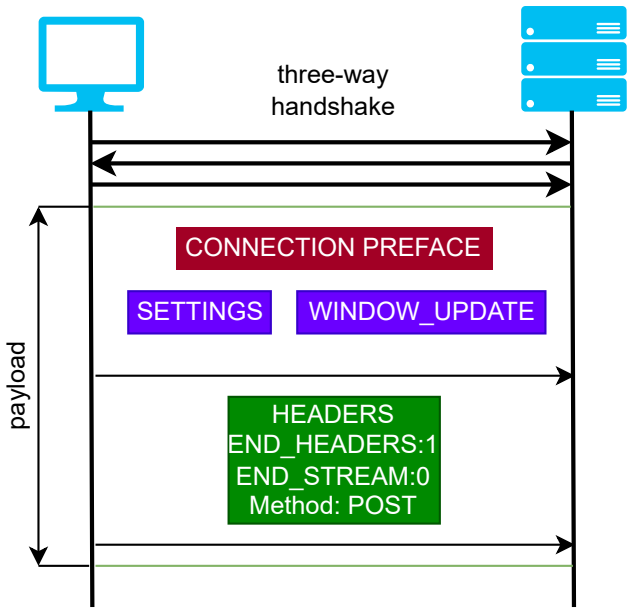
Obr. 3.1: Slow Read DoS.

3.2 HTTP/2 Slow POST

Pro pochopení tohoto typu útoku je nutné popsat příznaky v rámci `HEADERS`, obsahuje celkem čtyři, nicméně k pochopení útoku stačí definovat dva z nich:

1. Příznak `END_STREAM` – resetování tohoto rámce indikuje, že má klient stále připravená data, která budou zaslána.
2. Příznak `END_HEADERS` – v případě nastavení tohoto příznaku klient indikuje, že rámec `HEADERS` obsahuje všechna data hlavičky a nebude následovat rámec `CONTINUATION`.

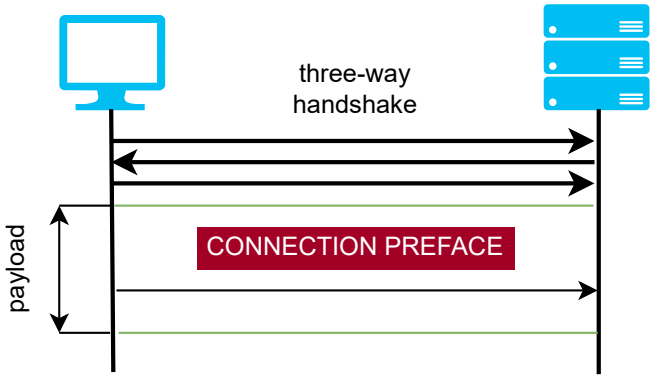
Průběh útoku tedy začíná, nastavuje a resetuje příznaky `END_HEADERS` společně s `END_STREAM` v rámci `HEADERS` v uvedeném pořadí a zasílá kompletní žádost POST, která tento rámec obsahuje a v něm zmíněné příznaky. Při přijetí žádosti POST si server myslí, že přijal kompletní žádost (díky příznaku `END_HEADERS`) a očekává rámec `DATA`, vzhledem k nastavenému příznaku `END_STREAM`. Server opět tedy nechává spojení otevřené a vyčkává komunikace ze strany útočníka. [21]



Obr. 3.2: Slow POST DoS.

3.3 HTTP/2 Slow PREFACE

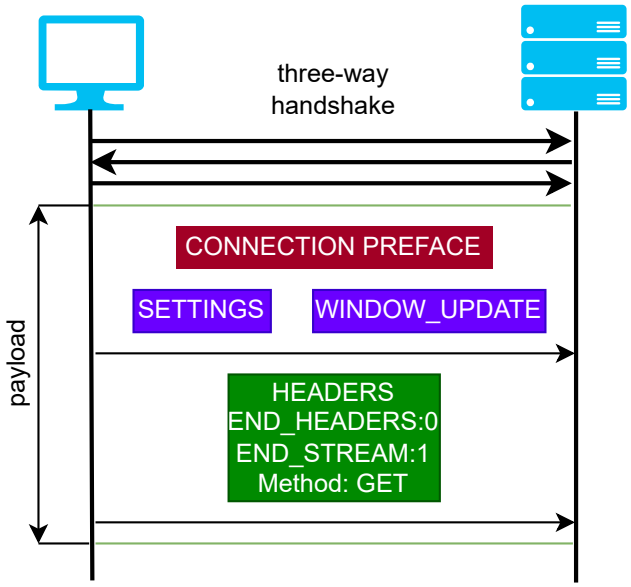
Útok probíhá způsobem, že po sestavení spojení útočník zasílá pouze connection preface. Server drží spojení a vyčkává na HTTP žádost typu GET nebo POST. Útočník pochopitelně tuto zprávu nikdy serveru nezašle, spojení tedy pokračuje až do definovaného času, kdy server spojení zruší. [21]



Obr. 3.3: Slow PREFACE DoS.

3.4 HTTP/2 Slow HEADERS

V tomto typu útoku může útočník zaslat jak žádost GET, tak POST. Princip je v nastavení příznaků resetovaného (s hodnotou nula) `END_HEADERS` a nastaveného (s hodnotou jedna) `END_STREAM` zaslaného rámce `HEADERS`, tedy opačně jako v případě Slow POST. Příznak `END_HEADERS` v resetované hodnotě indikuje, že po rámci `HEADERS` musí dojít k zaslání jednoho nebo více rámců `CONTINUATION` (poslední se určuje nastavením příznaku `END_HEADERS`). [21]

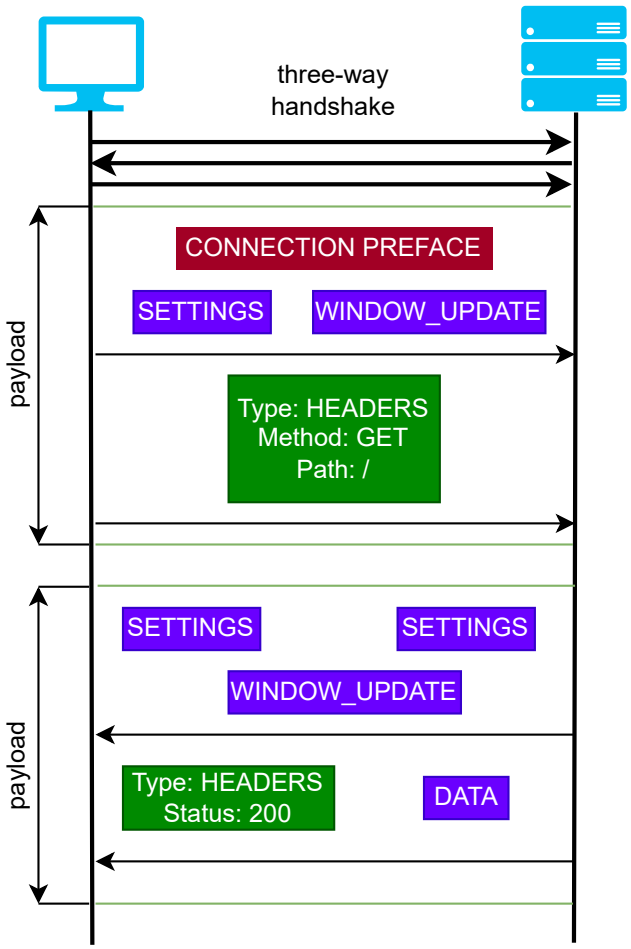


Obr. 3.4: Slow HEADERS DoS.

Při obdržení rámce `HEADERS` server předpokládá, že rámec není kompletní a očekává zaslání dalšího rámce s kompletními daty (toto určuje `END_STREAM`). V případě žádosti typu POST jsou oba příznaky rámce `HEADERS` resetovány. [21]

3.5 HTTP/2 Slow SETTINGS

Tento útok je založen na nutnosti potvrzení rámce `SETTINGS` oběma stranami komunikace. Pokud potvrzující nepotvrdí rámec v definovaném čase, spojení může být ukončeno. Útok probíhá způsobem, že klient zasílá běžnou žádost typu GET nebo POST. [21]



Obr. 3.5: Slow SETTINGS DoS.

Oproti ostatním pomalým útokům je zde rozdíl v tom, že server zde normálně odpovídá, tedy zasílá rámce `DATA` a potvrzující rámec `SETTINGS`, tento zasílá dvakrát, ve druhém zasílá své nastavení komunikace a od útočníka chce potvrzení. Útočník potvrzení serveru nezasílá a komunikace trvá, dokud není serverem ukončena. [21]

4 Historie jednotlivých verzí HTTP protokolů

Hlavním tématem této práce je popsání HTTP protokolu. V první části je nutné uvést vývoj tohoto protokolu v průběhu posledních let. Další část bude zaměřena na typy HTTP protokolu.

Vyvíjení tohoto protokolu začal Tim Berners-Lee se svým týmem v organizaci *CERN* v roce 1989. Šlo o způsob, kde v rámci sítě propojených počítačů, které by poskytovaly přístup k obsahu a tyto obsahy propojovaly, aby se vzájemně odkazovaly na sebe v reálném čase, tedy otevření odkazu zobrazí připojený dokument.

Během roku 1989 a 1990 Berners-Lee publikoval návrh tvorby systému, založil první webový server založený na HTTP protokolu a první webový prohlížeč, který byl schopen přijímat HTTP požadavek dokumentů a zobrazovat je v tomto webovém prohlížeči. [14]

4.1 HTTP/0.9

Prvním publikovaným protokolem byla verze 0.9 v roce 1991. Tato publikovaná specifikace měla dokumentaci okolo 700 slov. Specifikace definovala, že komunikace probíhá přes TCP/IP protokol přes server a doporučený port (v tomto případě se jedná o port 80, jelikož jde o nezabezpečenou verzi HTTP). Jednotlivé řádky ASCII (American Standard Code for Information Interchange) budou zaslány přes metodu *GET*. Tedy jediná dostupná metoda v této verzi 0.9 byla **GET** – přes tuto metodu lze získat data z webového serveru. Zároveň odpověď serveru na metodu *GET* byla velmi jednoduchá. Nebyly zde ani HTTP hlavičky, na rozdíl od dalších verzí, kde je už nalezneme. Verze neobsahovala ani chybové kódy a kódy stavů. [5]

4.2 HTTP/1

Tato verze je založena na TCP spojení. Každý dotaz na stejný server vytváří samostatné TCP spojení. Další podporované metody v této verzi byly:

- **HEAD** – jedná se o podobnou metodu jako *GET*, ale server nemusí v odpovědi vrátit *Entity-Body*.
- **POST** – metoda, která umožňuje klientovi zaslat na server data. [9]

4.3 HTTP/1.1

Tato verze začala používat „keep-alive“ metodu, díky které je možné v rámci jedné TCP komunikace zasílat požadavky a není nutné vytvářet další spojení pro každý další požadavek. Další podporované metody v této verzi byly:

- **OPTIONS** – představuje žádost o informace dostupné v řetězci žádost/odpověď.
- **PUT** – tato metoda se využívá k zaslání dat na server a tím je buď vytvoří, jako metoda *POST* a nebo tato data přepíše, jako metoda *UPDATE*, rozdíl mezi *PUT* a *POST* je, že při využití metody *PUT* se data nevkládají stále znovu.
- **DELETE** – metoda, která zasílá žádost o smazání daných dat. [10]

4.3.1 Rozdíly oproti HTTP/1

- Hlavička hosta – slouží ke směrování zpráv přes proxy server, je důležitá pro rozlišování domén, které ukazují na stejnou IP adresu.
- Trvající spojení popsané viz výše.
- Nové metody popsané výše.
- Pokračující statusy – předchází odmítnutí zprávy, klient může nejprve odeslat pouze hlavičku požadavku a obdrží stavový kód.

Mezi další rozdíly patří komprese, dekomprese hlavičky a podpora jazyků. [22]

4.4 HTTP/3

Standard, který definuje mechanismus dotaz-odpověď mezi klientem a serverem. Nejprve byl protokol HTTP/3 nazýván jako „HTTP-over-QUIC“, hlavní myšlenkou je, aby syntaxe a protokol HTTP/2 a jeho funkcionality byly kompatibilní s protokolem transportní vrstvy QUIC (Quick UDP Internet Connections). S novým protokolem plynou nové výhody oproti HTTP/2. Jedná se zejména o potlačení HOL blocking, nový kryptografický handshake a zabudované šifrování. [23]

4.4.1 QUIC

QUIC funguje na UDP protokolu. UDP protokol vytváří nespojovanou a nespolehlivou cestu. Hlavním rozdílem mezi HTTP/2 a HTTP/3 verzemi protokolu HTTP je, že nová verze přešla na UDP protokol. QUIC vznikl zhruba rok před novou verzí HTTP/3. Díky tomuto protokolu je možná implementace protokolu TLS 1.3 (Transport Layer Security). [23]

4.5 HTTPS

Jedná se o zabezpečenou verzi HTTP protokolu. V jeho přenosu je komunikace šifrována, nelze tedy zprávu zaslanou pomocí tohoto protokolu odposlechnout. Protokol pracuje na portu 443. Použití HTTPS (Hypertext Transfer Protocol Secure) komunikace lze jednoduše zjistit v prohlížeči, kde ve vyhledávači lze vidět předponu „https://“. [24]
Výhody:

- Šifrovaná komunikace – příkladem může být vyplňování formulářů na webu, které při odposlechu nepůjde přečíst.
- Integrita dat – odolné vůči odposlechu.
- Důvěrnost – stránka není podvržena útočníkem. [24]

4.5.1 TLS 1.2

Oproti svému předchůdci TLS 1.1 nabízí zvýšené zabezpečení, TLS 1.1 zabezpečení obsahovalo kombinaci MD5/SHA-1, nová verze toto nahradila jedinou funkcí HASH. Dále je umožněna lepší možnost výběru algoritmu pro šifrování ověřování u klienta a serveru. [25]

4.5.2 TLS 1.3

Jedná se o standard šifrování internetové komunikace. Doplňuje nedostatky TLS 1.2. Rozdílem je, že protokoly pro šifrování ověřování, které už nebyly bezpečné, tak tento bezpečnostní standard eliminuje (např. DES, MD5). Kvalitnější je zejména TLS Handshake. TLS 1.2 využíval se svými předchůdci 2-RTT (Round Trip Time Resumption), kde doba zpoždění činila 500 ms. TLS 1.3 využívá 0-RTT, toto významně redukuje zpoždění, neboť ve chvíli, kdy je na stránce už navázáno spojení, tak při opětovném navštívení stránky, nebo obnovení už ověřování už neprobíhá. Toto je významná výhoda oproti TLS 1.2. [26]

5 Protokol HTTP/2

HTTP/2 poskytuje optimalizovaný přenos pro architekturu HTTP. Byla potřeba vytvořit efektivnější verzi tohoto protokolu, protože předešlá verze HTTP1.1 už neměla dostatečné funkcionality. Obecně tato nová verze podporuje všechny funkce verzí předešlých. Jedná se o protokol aplikační vrstvy, který běží na aplikační vrstvě přes TCP spojení a klient je zde iniciátor spojení se serverem na portu 80.

Základní a nejmenší jednotkou v protokolu HTTP/2 je **rámec**. Každý typ rámce dělá odlišnou funkcionalitu. Například rámce `HEADERS` a `DATA` tvoří základ HTTP dotazů a odpovědí. Další rámce jako `SETTINGS`, `WINDOW_UPDATE` a `PUSH_PROMISE` se používají pro podporu dalších HTTP/2 funkcionalit, budou blíže popsány v rámci této práce.

Protokol HTTP/2 využívá schéma jak HTTP, tzn. nezabezpečené verze HTTP protokolu, tak HTTPS (Hypertext Transfer Protocol Secure), tento protokol je využit v nových počítačových doménách pro zabezpečenou komunikaci s možností certifikátu. [12]

5.1 Přechod z SPDY na protokol HTTP/2

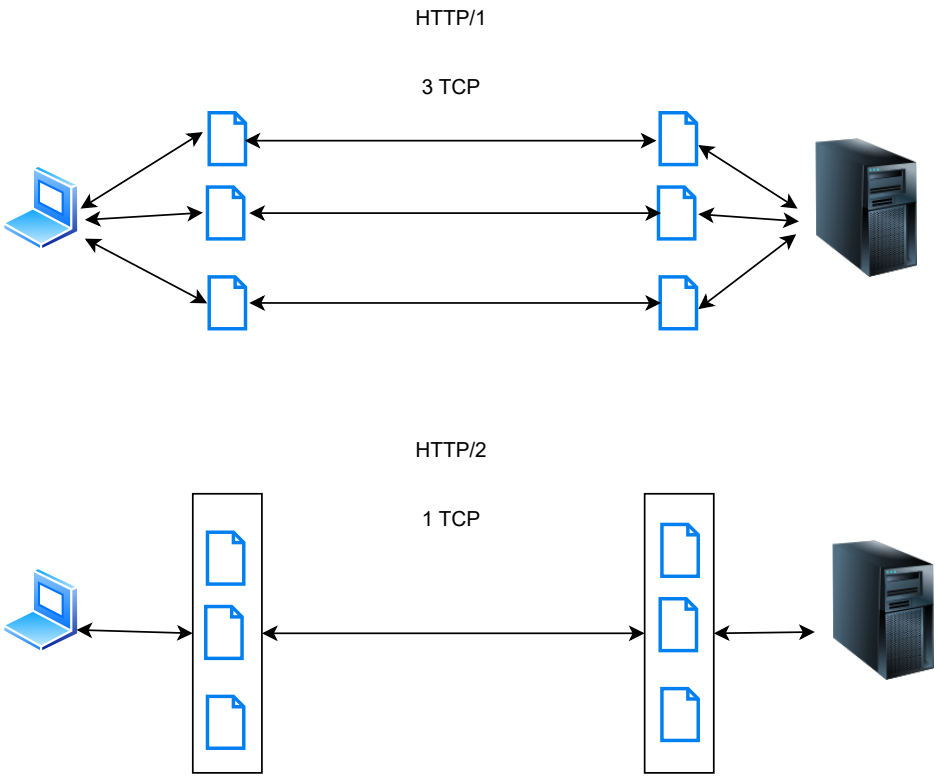
Protokol *SPDY* vytvořil Google, 8. února 2016 se rozhodl stáhnout *SPDY* a uvolnit cestu nástupci HTTP/1.1, tedy protokolu HTTP/2, který byl dokončen a schválen.

SPDY Google začal vyvíjet z důvodu, že verze HTTP protokolu už nestačila na rychlý vývoj webů ze statických, které téměř nic neuměly na dynamické weby, které obsahovaly hodně složitějších věcí, na které tento původně navrhovaný protokol nestačil. *Google* tedy v tomto ohledu přišel s komplexnějším protokolem SPDY, na kterém je HTTP/2 založený, ale tento také nestačil na celou dobu, proto se rozhodli uvolnit cestu původní specifikaci.

Nikdy ani nebylo zamýšleno, že by *SPDY* byl plnohodnotnou náhradou, pouze vytvořil základy pro HTTP/2 zlepšením přenosu dat bez narušení zpětné kompatibility. [11]

5.2 Rozdíly mezi HTTP/1.1 a HTTP/2

Jedním ze základních rozdílů mezi těmito dvěma verzemi protokolu HTTP je multiplexování. Zatímco u verze HTTP/1.1 se při každém jednom požadavku otevírá nové TCP spojení, tak u HTTP/2 jsou všechny soubory přenášeny v rámci jednoho TCP spojení mezi lokálním klientem a serverem. Tohoto si lze všimnout přímo na obr. 5.1. [8]



Obr. 5.1: Zobrazení TCP spojení na jednotlivých protokolech.

HTTP/1.1 zasílá zprávy jako prostý text, ale HTTP/2 zprávy zakóduje do binárních dat a pečlivě je uspořádá.

Načítání dalších zdrojů ze serveru vyžaduje, aby klient zaslal opakovaně požadavky a zároveň opakovaně tvořil a přerušoval TCP spojení. [8]

5.2.1 HTTP/1.1

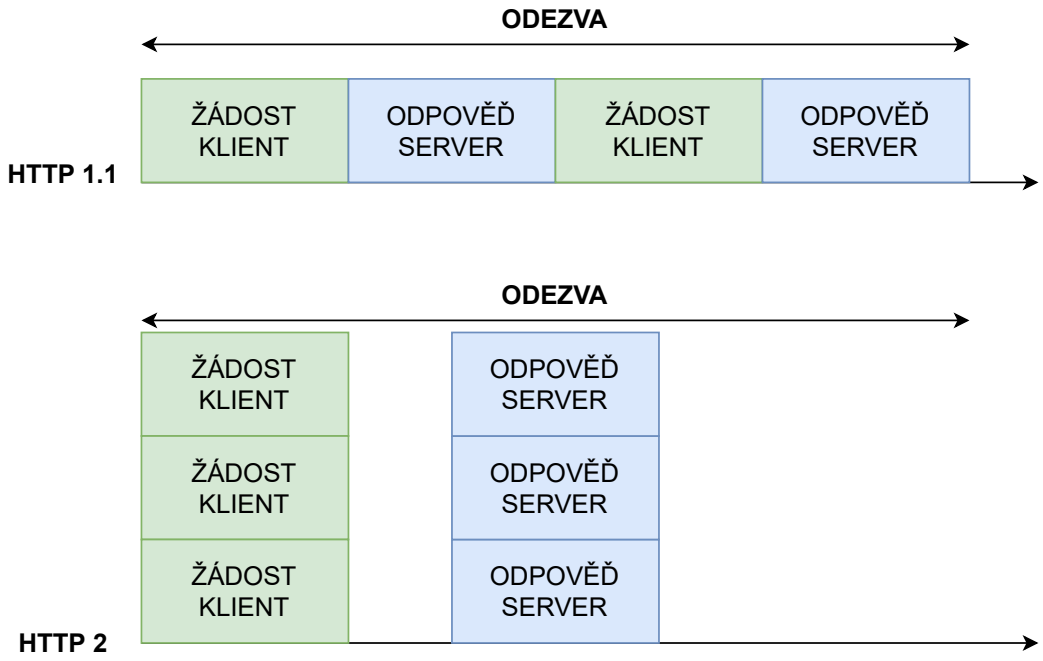
Tato verze protokolu tuto situaci popsanou výše řeší vytvořením trvalého spojení mezi serverem a klientem, dokud není pevně uzavřeno jednou z komunikujících stran. Klient tedy v této verzi může používat vytvořené TCP spojení opakovaně.

Nicméně problematický je zde HOL blocking (Head-Of-Line blocking).

Tedy při zasílání jednotlivých dotazů a odpovědí se může stát, že jeden dotaz není vyřešený a tím pádem ostatní čekají ve frontě a nejsou připuštěné. [10]

5.2.2 HTTP/2

Byla zavedena vrstva binárního rámce, tato vrstva rozděluje dotazy a odpovědi na malé datové pakety a kóduje je, díky této vrstvě může běžet souběžně více dotazů a odpovědí a tím je vyřešený problém HOL blocking.



Obr. 5.2: Zobrazení zasílání dotazů a odpovědí v HTTP.

Dalším řešením je souběžná komunikace, ne jak u verze HTTP/1.1 viz obr. 5.2, je možné zasílat jednotlivé dotazy souběžně a server také bude odpovídat souběžně na dotazy, nevznikne tedy HOL blocking, jak bylo výše popsáno. [8]

5.3 Výhody a nevýhody protokolu HTTP/2

Výhody:

- HTTP/2 podporuje celkový multiplex pro dotazy i odpovědi přes TCP spojení, díky této podpoře multiplexace je snížena latence a zvýšena celková kapacita sítě spolu s dostupností sítě.
- S moderními dobou rostly požadavky na výkon zdroje. Vývojáři tedy v nové verzi HTTP/2 přišli s funkcemi na úrovni protokolu jako server push, závislost na streamování a upřednostňování, komprese hlavičky a binární formát vrstvy, tedy jde o lepší využití sítě.
- Díky přenosu více dat v komunikaci od klienta k serveru se zvýšil výkon webu.
- Všechny moderní webové prohlížeče podporují HTTP/2 přes zabezpečenou verzi HTTPS s SSL (Secure Sockets Layer) certifikátem.
- Využití HPACK algoritmu umožňuje překonat protokolu HTTP/2 různé zranitelnosti. Tento protokol má příkazy v binární formě a komprimuje metadata HTTP hlavičky, aby se chránila citlivá data v komunikaci mezi dvěma zařízeními.

Nevýhody

- Zatímco předchozí verze HTTP protokolu potlačily HOL blocking, TCP komunikace, na které je tato verze protokolu založena stále není bezchybně vyřešena.
- Pokud klientovo zařízení pracuje na velmi pomalé síti, tak se datové pakety začínají ztrácet a kvalita sítě je snížena na jediné HTTP/2 připojení. Díky tomuto problému na straně klienta je celá komunikace výrazně zpožděna.
- Špatné zabezpečení souboru cookie není vyřešeno ve verzi HTTP/2. Cookie jsou textový soubor, který obsahuje data od klienta, tato data byla získána serverem a webovou stránkou. Tento soubor díky nízkému zabezpečení může být ukraden a klientovi mohou být zneužity jeho citlivé údaje. [8]

5.4 HTTP/2 connection preface

Každý koncový bod je povinen zaslat connection preface jako finální potvrzení, že se protokol bude pro danou komunikaci využívat a stanovit počáteční nastavení HTTP/2 spojení. Klient i server každý zasílá rozdílný connection preface.

Klientův connection preface začíná se sekvencí 24 oktetů, které jsou v hexové notaci:

```
0x505249202a20485454502f322e300d0a0d0a534d0d0a0d0a
```

Connection preface začíná textovým řetězcem:

```
"PRI␣*␣HTTP/2.0\r\n\r\nSM\r\n\r\n"
```

tato sekvence musí obsahovat rámec `SETTINGS`, který může být prázdný. Klient zasílá svůj connection preface jako první data pro samotnou komunikaci. Connection preface serveru se skládá z rámce `SETTINGS`, který může být také prázdný a také se musí jednat o první rámec, který samotný server zašle.

K eliminaci zbytečné latence je klient oprávněn zasílat přídavné rámce na server bezprostředně po zaslání svého connection preface bez čekání, než obdrží connection preface od serveru. Nicméně se může stát, že rámec `SETTINGS` bude obsahovat nějaké úpravy nastavení, jak má klient komunikovat se serverem, v některých nastaveních je možné, aby server převedl tento rámec hned, aby klient věděl, jak má se serverem komunikovat, tím se zabrání, aby klient následně obdržel tento rámec a v této stávající komunikaci už serveru zaslal své další rámce a až potom obdržel nastavení, jak měl vlastně komunikovat.

Klienti a servery musí označit chybný connection preface jako chybu spojení rámcem `PROTOCOL_ERROR`. Rámec `GOAWAY` může být vynechán, protože chybný connection preface indikuje, že jedna ze stran nevyužívá protokol HTTP/2. [12]

5.5 HTTP rámce

Jakmile je navázáno HTTP/2 spojení, tak si koncové body mohou začít vyměňovat jednotlivé rámce.

Formát rámce: všechny rámce začínají s fixním 9 oktetovým záhlavím, které je doplněno payloadem rámce s proměnlivou délkou.

```
HTTP Frame {
Length (24),
Type (8),
Flags (8),
Reserved (1),
Stream Identifier (31),
Frame Payload (..), }
```

- **Length:** Délka payloadu rámce. Hodnoty větší než 2^{14} nemohou být odeslány, pokud příjemce nenastavil vyšší hodnotu rámce `SETTINGS_MAX_FRAME_SIZE`.
- **Type:** 8bitový typ rámce. Typ rámce popisuje formát a sémantiku rámce.
- **Flags:** 8bitové pole rezervováno pro booleovské specifické příznaky pro typ rámce.
- **Reserved:** Rezervované 1bitové pole. Sémantika tohoto bitu není definovaná a bit musí zůstat nenastavený, když se posílá a musí být ignorovaný když je přijat.
- **Stream Identifier:** Je vyjádřený jako netypová 31bitová celočíselná hodnota.
- **Frame Payload:** Jsou poslaná data ze zdroje k cíli. [12]

5.5.1 Velikost rámce

Velikost payloadu rámce je limitovaná maximální velikostí, kterou příjemce definuje v rámci `SETTINGS_MAX_FRAME_SIZE` nastavení. Toto nastavení může mít kteroukoliv hodnotu mezi 2^{14} a $2^{24} - 1$ oktetů.

Implementace musí být schopny přijímat a zpracovávat rámce až do 2^{14} oktetů a k tomu 9 oktetů rámec hlavičky. Při popisu velikostí rámců není zahrnuta velikost záhlaví rámce.

Koncový bod musí zaslat chybovou zprávu `FRAME_SIZE_ERROR`, pokud rámec přesahuje délku definovanou v `SETTINGS_MAX_FRAME_SIZE`, a to i v případě, že velikost překračuje definovaný limit nebo je příliš nízká na to, aby obsahovala minimálně povinné data rámce.

Koncové body nejsou povinné využít všechen dostupný prostor v rámci. Reakční doba může být zvýšena využitím rámců které jsou menší než povolená maximální velikost daného rámce. Zasílání rámců velké kapacity může ve výsledku způsobit

zpoždění v komunikaci mezi dvěma koncovými body, zpoždění může být problém, pokud se zasílá časově citlivý rámec např. `WINDOW_UPDATE`, který v případě zpoždění může snížit výkon. [12]

5.6 Komprese hlavičky

HTTP hlavičky jsou používané k zasílání přídavných informací ohledně dotazů a odpovědí od klienta k serveru a opačně. Některé hlavičky jsou zasílány s každým dotazem.

- Cookie – jsou zasílány s každým dotazem na každou doménu.
- User-Agent – obvykle uvádí používaný webový prohlížeč, nikdy se nemění v probíhající relaci a odesílá se s každým dotazem.
- Host – tato hlavička se využívá k úplné kvalifikaci dotazované URL (Uniform Resource Locator) a je vždy stejná pro každou žádost.
- Accept – tato hlavička definuje formát odpovědi.
- Accept-Encoding – hlavička definující formát komprese. [12]

5.7 Streamy a multiplexování

Stream je nezávislá a obousměrná sekvence rámců vyměňovaných mezi koncovými uživateli, tedy mezi klientem a serverem, kteří komunikují přes HTTP/2 spojení.

Základní charakteristiky streamu:

- Jedno HTTP/2 spojení může obsahovat souběžně více otevřených streamů s dalšími koncovými body.
- Stream může být založen a využíván jednostranně nebo sdílen s ostatními koncovými body.
- Stream může být ukončen jedním z koncových bodů.
- Pořadí zasílaných rámců je důležité, neboť je příjemce zpracovává v pořadí ve kterém je přijímá, například pořadí rámců `HEADERS` a `DATA` je sémanticky významný.
- Streamy jsou identifikovány celočíselnou hodnotou. Identifikátory streamů jsou přiděleny koncovými zařízeními, které spojení začaly. [12]

5.8 Definice rámců

Specifikace definuje čísla typů rámců, jednotlivé rámce jsou definovány unikátním 8bitovým typem kódu. Každý typ rámce slouží k jinému účelu, jak bylo již dříve popisováno v rámci této práce, nyní jednotlivé rámce budou blíže popsány.

Každý rámec je jiný a může způsobit změnu stavu spojení dvou koncových bodů, z tohoto důvodu je nutné, aby koncové body (klient a server) měly sdílený pohled na to, jak bude daná komunikace ovlivněna zasláním konkrétních rámců. [12]

5.8.1 Rámec DATA

Jeden nebo více rámců DATA jsou využívány k přenosu HTTP dotazu nebo odpovědi daného obsahu zprávy. Tento rámec může také obsahovat výplň, která může regulovat velikost dané zprávy, která je obsahem tohoto rámce a výplň patří také mezi bezpečnostní prvek. [12]

```
DATA Frame {
    Length (24),
    Type (8) = 0x00,

    Unused Flags (4),
    PADDED Flag (1),
    Unused Flags (2),
    END_STREAM Flag (1),

    Reserved (1),
    Stream Identifier (31),

    [Pad Length (8)],
    Data (..),
    Padding (..2040),
}
```

Části rámce jako délka (length), typ (type) apod. byly dříve popsány. Nyní bude zobrazena pouze dosud nepopsaná část. [12]

- Pad Length: 8bitové pole obsahující délku výplně rámce v jednotkách oktetů, toto pole je zobrazeno pouze v případě, že příznak `PADDED` je nastaven.
- Data: aplikační data
- Padding: musí být nastavena nulová hodnota, příjemce ověřuje tento příznak a pokud není nulový, tak daný rámec může vyhodnotit jako chybu spojení. [12]

5.8.2 Rámec HEADERS

Je využíván k otevření streamu a nese fragment pole. Tento rámec také může být zaslán na stream ve stavech „idle“, „reserved (local)“, „open“ nebo „half-closed (remote)“.

```
    HEADERS Frame {
        Length (24),
        Type (8) = 0x01,

        Unused Flags (2),
        PRIORITY Flag (1),
        Unused Flag (1),
        PADDED Flag (1),
        END_HEADERS Flag (1),
        Unused Flag (1),
        END_STREAM Flag (1),

        Reserved (1),
        Stream Identifier (31),

        [Pad Length (8)],
        [Exclusive (1)],
        [Stream Dependency (31)],
        [Weight (8)],
        Field Block Fragment (..),
        Padding (..2040),
}
```

- Exclusive: je zobrazován pouze v případě, že příznak `PRIORITY` je nastaven.
- Stream Dependency: identifikátor, který je zobrazován ve stejném případě jako Exclusive.
- Weight: stejný případ jako Exclusive. [12]

5.8.3 Rámec SETTINGS

Tento rámec slouží k zaslání konfiguračních parametrů, které si předem stanoví dva komunikující uzly, jako jsou preference a omezení. Rámec je také využit k potvrzení tohoto nastavení odesláním od příjemce, kterému dorazil původní rámec.

Nastavení není vyjednáno ve chvíli, kdy např. klient zašle nějaký požadavek v rámci `SETTINGS`, zatímco server jako odpověď ve svém rámci zasílá jinou hodnotu.

Rámec musí být vždy zaslán oběma koncovými body.

Nově zaslaný rámec `SETTINGS` mění původní konfigurační parametry v rámci komunikace.

Tento rámec ovlivňuje stav připojení, špatně zaslaný rámec nebo nekompletní rámec `SETTINGS` je označen jako chyba spojení `PROTOCOL_ERROR`.

Pokud rámec obsahuje odlišnou délku než násobek 6 oktetů, tak je označen za chybu spojení `FRAME_SIZE_ERROR`.

```
    SETTINGS Frame {
        Length (24),
        Type (8) = 0x04,

        Unused Flags (7),
        ACK Flag (1),

        Reserved (1),
        Stream Identifier (31) = 0,

        Setting (48) ...,
}


        Setting {
        Identifier (16),
        Value (32),
}
```

`Definované nastavení`

- **SETTINGS_HEADER_TABLE_SIZE**: informuje koncový bod o maximálně velikosti kompresní tabulky.
- **SETTINGS_ENABLE_PUSH**: jde o možnost povolit nebo zakázat server push.
- **SETTINGS_MAX_CONCURRENT_STREAMS**: udává maximální počet souběžných streamů, které odesílat dovolí.
- **SETTINGS_INITIAL_WINDOW_SIZE**: udává počáteční velikost okna odesílatele pro stream.
- **SETTINGS_MAX_FRAME_SIZE**: největší velikost payloadu rámce, který je odesílatel schopen akceptovat.
- **SETTINGS_MAX_HEADER_LIST_SIZE**: informuje o maximální velikosti pole, které odesílatel může přijmout.

Ve chvíli, kdy je zasláno jiné nastavení, tak příjemce toto nastavení ignoruje. [12]

5.8.4 Rámec PUSH_PROMISE

Tento rámec se využívá pro upozornění koncového bodu o tom, že odesílatel má v úmyslu zahájit s ním spojení. [12]

```
    PUSH_PROMISE Frame {
        Length (24),
        Type (8) = 0x05,

        Unused Flags (4),
        PADDED Flag (1),
        END_HEADERS Flag (1),
        Unused Flags (2),

        Reserved (1),
        Stream Identifier (31),

        [Pad Length (8)],
        Reserved (1),
        Promised Stream ID (31),
        Field Block Fragment (..),
        Padding (..2040),
}
```

5.8.5 Rámec GOAWAY

Tento rámec se využívá k zahájení vypnutí spojení, nebo signalizování chybového stavu. Umožňuje koncovému bodu zastavit přijímání nových streamů, zatímco dokončuje zpracování předešlých založených streamů. Tento rámec má také využití u administrátorů, kdy se může zaslat před údržbou serveru. [12]

```
    GOAWAY Frame {
        Length (24),
        Type (8) = 0x07,

        Unused Flags (8),

        Reserved (1),
        Stream Identifier (31) = 0,

        Reserved (1),
        Last-Stream-ID (31),
        Error Code (32),
        Additional Debug Data (..),
}
```

5.8.6 Rámec WINDOW_UPDATE

Rámec slouží k flow control. Tento rámec byl dříve blíže popsán v rámci této práce. [12]

```
    WINDOW_UPDATE Frame {
        Length (24) = 0x04,
        Type (8) = 0x08,

        Unused Flags (8),

        Reserved (1),
        Stream Identifier (31),

        Reserved (1),
        Window Size Increment (31),
}
```

5.8.7 Rámec CONTINUATION

Je používán k pokračování sekvence fragmentů bloků pole. Může být zaslán libovolný počet těchto paketů, dokud je předchozí rámec ve stejném streamu. [12]

```
    CONTINUATION Frame {
        Length (24),
        Type (8) = 0x09,

        Unused Flags (5),
        END_HEADERS Flag (1),
        Unused Flags (2),

        Reserved (1),
        Stream Identifier (31),

        Field Block Fragment (..),
}
```

5.9 Chybové kódy

Využívají se v rámcích `RST_STREAM` a `GOAWAY` na sdělení důvodu proč došlo k chybě streamu nebo spojení.

Definování následujících chybových kódů:

- **NO_ERROR** – přidružený stav není výsledkem chyby.
- **PROTOCOL_ERROR** – koncový bod zjistil nespecifickou chybu protokolu.
- **INTERNAL_ERROR** – koncový bod zaznamenal neočekávanou vnitřní chybu.
- **FLOW_CONTROL_ERROR** – koncový bod zjistil, že druhý koncový bod porušil flow control.
- **SETTINGS_TIMEOUT** – koncový bod odeslal rámec `SETTINGS`, ale nedostal odpověď v definovaném čase.
- **STREAM_CLOSED** – obdržení rámce po „half-closed“ stavu streamu.
- **FRAME_SIZE_ERROR** – obdržení rámce s neplatnou velikostí.
- **REFUSED_STREAM** – odmítnutí streamu.
- **CANCEL** – indikuje, že bude ukončovat stream.
- **COMPRESSION_ERROR** – koncový bod není schopen udržet kontext komprese sekce pole pro připojení.
- **CONNECT_ERROR** – spojení navázané v reakci na požadavek `CONNECT` bylo resetováno nebo uzavřeno zvláštním způsobem.
- **ENHANCE_YOUR_CALM** – jeden z koncových bodů zjistil, že partner může přetěžovat spojení.
- **INADEQUATE_SECURITY** – přenos obsahuje charakteristické vlastnosti, které nesplňují minimální požadavky zabezpečení.
- **HTTP_1_1_REQUIRED** – koncový bod chce využívat HTTP/1.1 místo HTTP/2. [12]

6 HPACK: Komprese hlavičky pro HTTP/2

V HTTP/1.1 u pole záhlaví neprobíhala komprese. Se zvětšujícím se nárůstem uživatelů přistupujících k webovým serverům kvůli zbytečnému pole záhlaví, se spotřebovávala šířka pásma, která zvyšovala znatelně latenci.

SPDY řešil tuto redundanci kompresí polí záhlaví pomocí formátu `DEFLATE`, který se ukázal jako efektivní. Nicméně toto řešení nebylo zabezpečené a bylo prolomené.

HPACK formát je záměrně jednoduchý a neflexibilní. Obě vlastnosti umožňují chybě implementace. [13]

Terminologie:

- **Pole záhlaví**: Pár jméno-hodnota.
- **Dynamická tabulka**: Spojuje uložená pole záhlaví s hodnotami indexu. Tabulka je dynamická a specifická pro kontext kódování nebo dekódování.
- **Statická tabulka**: Staticky spojuje pole záhlaví, které se často vyskytuje s hodnotami indexu. Tabulka je pouze pro čtení a sdílená mezi všemi kontexty kódování a dekódování.
- **Seznam záhlaví**: Seznam záhlaví je uspořádaná kolekce polí záhlaví, která jsou kódována společně a mohou obsahovat duplicitní pole záhlaví.
- **Reprezentace pole záhlaví**: Pole záhlaví může být reprezentováno v zakódované formě jako index.
- **Blok záhlaví**: Seznam reprezentací pole záhlaví, který po dekódování poskytne kompletní seznam záhlaví. [13]

6.1 Proces komprese

Specifikace nedefinuje konkrétní algoritmus pro kódování. Ale přesně definuje, jaké chování se od tohoto algoritmu očekává. [13]

6.1.1 HPACK statická tabulka

HPACK má statickou tabulku obsahující 61 společných názvů záhlaví (v některých případech i hodnoty). Pro představu jsou některé hodnoty v následující tabulce

Tab. 6.1: HPACK statická tabulka

Index	Název záhlaví	Hodnota záhlaví
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http

Tato tabulka se využívá pro dotazy a odpovědi a umožňuje HTTP zprávě efektivní komprese běžně používaných názvů. Například záhlaví

```
:method: GET
```

může být komprimováno jako reference na index 2, dle tab. 6.1. [14]

6.2 HPACK dynamická tabulka

Dynamická tabulka začíná na pozici 62 po statické tabulce až k maximální velikosti tabulky definované v nastavení hodnoty `SETTINGS_HEADER_TABLE_SIZE` v rámci `SETTINGS`. Dynamická tabulka je jedinečná pro každé TCP spojení. Proces vyhledávání v rámci dynamické tabulky je komplikovaný. [14]

6.3 HPACK typy záhlaví

Záhlaví může být nastaveno, aby se dalo přidat do dynamické tabulky, nebo ne. Existují čtyři typy záhlaví, které budou popsány v následujících podsekcích. [14]

6.3.1 Doslovná reprezentace pole záhlaví

Jedná se o přímé vyhledání v tabulce (dynamické nebo statické), takže se používá, když název záhlaví a hodnota existuje v tabulce. Tato hlavička se skládá z hodnoty indexu tabulky doplněné na minimálně 7 bitů. [14]

6.3.2 Doslovné pole záhlaví s přibývajícím indexováním

Využívá se, když hodnota záhlaví není dostupná v tabulce, ale může být přidána do dynamické tabulky pro pozdější použití. [14]

Tento typ obsahuje název záhlaví a hodnotu záhlaví. Pokud indexovaný název záhlaví je využívaný v tabulce (existuje v ní), bity obsahují 01, definují hodnotu indexu a následuje samotnou hodnotu záhlaví. [14]

6.3.3 Doslovné pole záhlaví bez indexování

Využívá se na položky, které se změní pro každý dotaz. Tyto typy obsahují název záhlaví, ale tento název a hodnota záhlaví nejsou vložené v dynamické tabulce. [14]

6.3.4 Doslovné pole záhlaví nikdy neindexované

Tento typ záhlaví se využívá pro citlivé informace (jména, hesla), informace které by neměly být vidět. [14]

6.4 Huffmanova kódovací tabulka

Huffmanovo kódování závisí na definované tabulce kódů, využívá se pro každý znak v textu. Pro HPACK je tato tabulka definovaná v specifikaci, takže klient i server znají hodnoty, které využít na kódování a dekódování názvu a hodnoty záhlaví. Tabulka 6.2 zobrazuje některé zakódované hodnoty. [14]

Tab. 6.2: HPACK Huffmanovo kódování

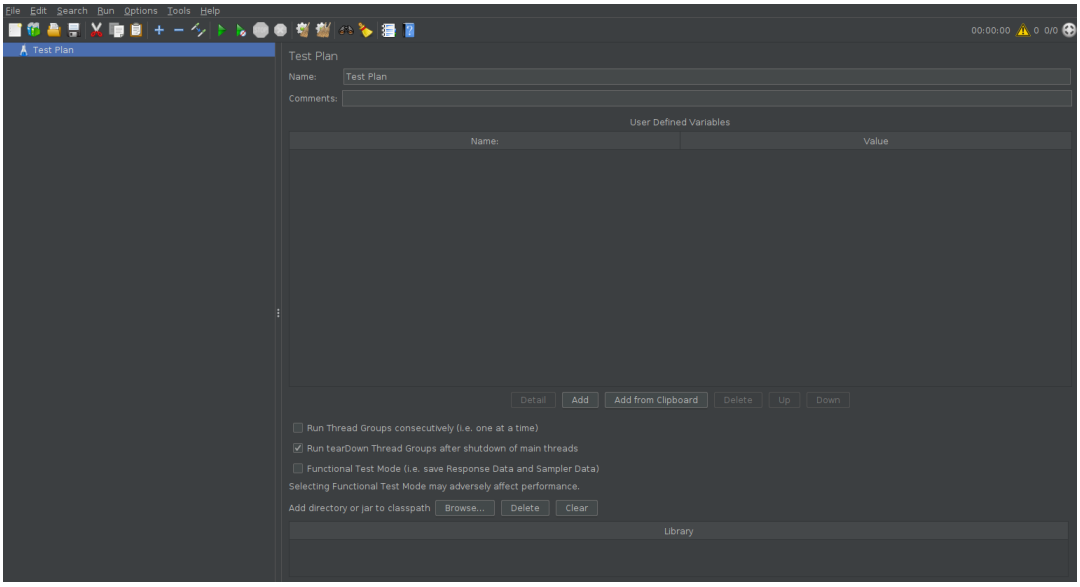
Symbol	ASCII kód	Huffmanův kód (binární)	Velikost (bity)
’ ’	(32)	|010100	[6]
’!’	(33)	11111110|00	[10]
’“	(34)	11111110|01	[10]
’#’	(35)	11111111|1010	[12]
’$’	(36)	||11111111|11001	[13]
’0’	(48)	|00000	[5]

7 Nástroj Apache JMeter

Jedná se o *open source* aplikaci, která je celkově založená na jazyce Java a je navržena pro zátěžové testování a měření výkonu. Originálně byla vytvořena pro testování webových aplikací, ale od doby vzniku se rozšířila k účelu dalších testovacích funkcí.

JMeter může být použit na testování výkonu statických i dynamických zdrojů a webových dynamických aplikací. Umožňuje simulování vysoké zátěže na server, skupinu serverů, síť nebo testování jiného objektu a analyzování celkového výkonu daného objektu při různém typu zátěže.

Umožňuje testovat zátěž a výkon spoustu odlišných zařízení, aplikací, serverů, protokolů a spoustu dalších, např. HTTP, HTTPS a TCP. [6]



Obr. 7.1: Úvodní obrazovka nástroje JMeter.

7.1 Spuštění nástroje JMeter

Následující řešení bude vysvětlováno v rámci operačního systému Linux.

Pro práci s tímto nástrojem je nutné udělat pár kroků k jeho úspěšnému spuštění. V první řadě je nutné naklonovat repozitář například z verzovacího systému GitHub pomocí příkazu:

Výpis 7.1: Příkaz pro naklonování repozitáře JMeter

```
git clone https://github.com/apache/jmeter
```

Následně je nutné spustit v naklonované složce JMeteru přes příkazový řádek příkaz:

Výpis 7.2: Sestavení nástroje JMeter.

```
./gradlew build
```

Tento příkaz způsobí celkové zkompilování projektu a vytvoří spustitelný soubor `jmeter.sh` ve složce `bin`.

Po těchto dvou základních příkazech 7.1 a 7.2 je možné spustit samotné GUI (Graphical User Interface) nástroje JMeter, jak lze vidět na obr. 7.1 příkazy:

Výpis 7.3: Spuštění nástroje JMeter.

```
cd /bin/
sudo sh jmeter.sh
```

V rámci příkazového řádku se prvním příkazem ve výpisu 7.3 lze přesunout do složky `bin`, kde je spustitelný soubor a následně jej spustit s právy superuživatele, z důvodu, když nástroj JMeter bude volat externí nástroje jako např. Python skript, tak by tuto operaci bez práv superuživatele nemohl provést, jelikož se jedná o zásah do systému.

Po zadání všech předešlých příkazů dojde ke spuštění uživatelského prostředí nástroje JMeter.

7.2 Struktura nástroje JMeter

Po naklonování JMeteru z oficiálního repozitáře na GitHubu bude složka po extrahování všech souborů obsahovat:

- backups – tato složka slouží k zálohování skriptů.
- bin – zde jsou uložené všechny binární soubory, dále jsou zde soubory potřebné pro spuštění nástroje, tato složka také obsahuje konfigurační soubory.
- doc – obsahuje obrázky, CSS (Cascading Style Sheets) soubory, např. logo nástroje.
- extras – jsou zde soubory XML (Extensible Markup Language), které jsou podporovány grafickým rozhraním nástroje.
- lib – obsahuje spustitelné jar soubory, podsložka `ext` obsahuje externí moduly.
- license – obsahuje všechny licence potřebné k provozování nástroje.
- xdocs – obsahuje např. tutoriály o nástroji a uživatelský manuál. [7]

7.3 Vývoj nového modulu pro nástroj JMeter

Nástroj JMeter je obecně přizpůsoben k tomu, aby do něj vývojáři měli snadné vyvíjet další moduly, obecně stačí v jednodušších případech vytvořit dvě třídy, a sice *Sampler* a *SamplerGui*. Třída **Sampler** se rozšiřuje abstraktní třídou AbstractSampler, zde dostaneme metody potřebné k provedení testu.

Další třída **SamplerGui** se rozšiřuje o abstraktní třídu AbstractSamplerGui a zde jsou také implementovány inicializační metody k vytvoření potřebného grafického rozhraní, toto grafické rozhraní se dále programuje přes Swing.

JMeter ve svém základu poskytuje velkou databázi modulů, na kterých se dá přímo stavět vytvářený modul, v případě této práce se bude jednat o modul HTTP request, který bude dále využit k vytvoření záplavových DoS útoků. Jelikož JMeter ve své nové verzi neposkytuje podporu HTTP/2 protokolu, je nutné využít externí modul od firmy BlazeMeter[1].

Možností při vývoji nového modulu je vytváření přímo v originálním zdrojovém kódu nástroje JMeter, tedy po vytvoření v našem případě tříd Sampler a SamplerGui se zavolá příkaz 7.2 a následně se ve složce `jmeter/build/libs` vytvoří `.jar` soubor s novým modulem pro nástroj JMeter a tento soubor z této složky zkopírujeme a přesuneme do složky `jmeter/lib/ext`, kde se po následném restartování nástroje tento nový modul zobrazí.

7.3.1 Instalace Apache serveru v aktuální verzi HTTP/2

Jelikož nejnovější verze Apache serveru, na který bude probíhat útok nepodporuje HTTP/2 protokol, je nutné doplňkovými příkazy podporu této verze HTTP protokolu umožnit.

V první řadě je nutné na virtualizovaný stroj nainstalovat aktuální verzi Apache serveru příkazem, také je dobré aktualizovat příkazy potřebné k instalaci:

Výpis 7.4: Instalace Apache serveru.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install apache2
```

Následně ověříme, zda je Apache server aktivní:

Výpis 7.5: Status Apache serveru.

```
systemctl status apache2
```

[1]firma, která vyvíjí nové moduly pro Apache JMeter

Dále je nutné nakonfigurovat firewall k zvýšení zabezpečení a povolit, aby přijímal spojení *OpenSSH* a *Apache Full*, které umožňuje zasílat požadavky i zabezpečené verze HTTPS na portu 443 protokolu a také té nezabezpečené, tedy HTTP na portu 80 příkazy:

Výpis 7.6: Konfigurace firewallu.

```
sudo ufw allow OpenSSH
sudo ufw allow 'Apache␣Full'
sudo ufw enable
```

Dále je nutné aktualizovat a doinstalovat potřebné PHP (Hypertext Preprocessor) moduly, které podporují HTTP/2 komunikaci v rámci Apache serveru.

Výpis 7.7: Instalace potřebných PHP modulů.

```
sudo apt-get install php7.4-fpm
sudo a2dismod php7.4
sudo a2enconf php7.4-fpm
```

A spuštění přídavných modulů, které přidávají nastavení serveru k zpřístupnění tohoto protokolu.

Výpis 7.8: Konfigurační moduly pro HTTP/2.

```
sudo a2enmod proxy_fcgi
sudo a2dismod mpm_prefork
sudo a2enmod mpm_event
sudo a2enmod ssl
sudo a2enmod http2
sudo systemctl restart apache2
systemctl status apache2
```

Jako poslední je nutné do konfiguračního souboru `http2.conf` přidat protokoly, aby Apache server vytvořil prioritu HTTP/2 před HTTP/1.1 příkazem: [16]

Výpis 7.9: Cesta ke konfiguračnímu souboru.

```
cd /etc/apache2
grep -r Protocols
```

A přidá se zde řádek:

Výpis 7.10: Přidání daného řádku k textu.

```
Protocols h2 h2c http/1.1
```

7.3.2 Testování HTTP/2 spojení

První z možností jak ověřit, zda server podporuje HTTP/2 spojení je zadání příkazu:

Výpis 7.11: Ověření podpory HTTP/2 protokolu.

```
curl http://<ip adresa> --http2-prior-knowledge
```

kde `<ip adresa>` je IP adresa Apache serveru. Ve chvíli, kdy server vrátí odpověď na tento dotaz, tak podporuje HTTP/2 protokol.

Testování HTTP/2 spojení bude provedeno pomocí modulu HTTP/2 Request od firmy BlazeMeter, kde je nutné ke stažení tohoto modulu si stáhnout `Jmeter Plugins Manager` od stejnojmenné firmy. Tento stáhneme příkazem:

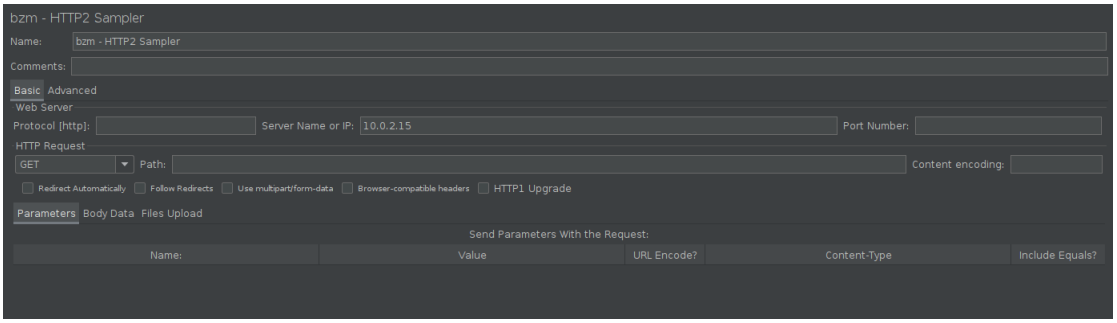
Výpis 7.12: Stažení Jmeter Plugins Manageru.

```
wget "https://jmeter-plugins.org/get/"
```

a následně je nutné umístit do složky `jmeter/lib/ext/`, jak bylo dříve vysvětleno, při vkládání nového modulu, pomocí příkazu:

Výpis 7.13: Kopírování staženého modulu do nástroje JMeter.

```
cp jmeter-plugins-manager-1.8.jar jmeter/lib/ext
```



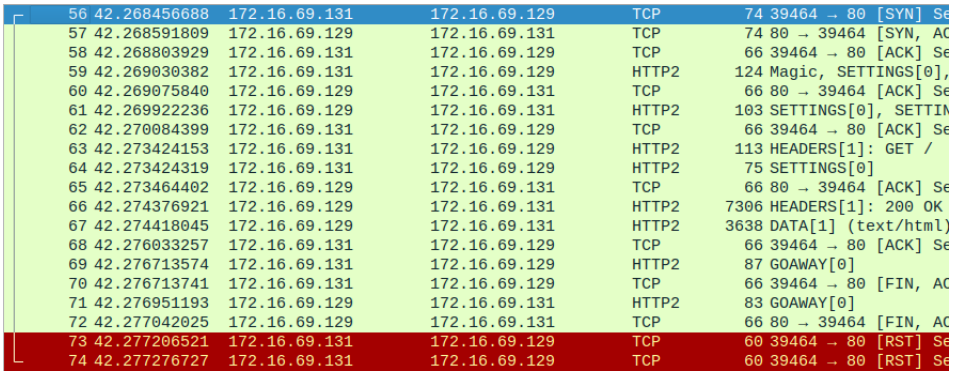
Obr. 7.2: HTTP/2 modul v nástroji JMeter.

Testování spojení lze vidět na obr. 7.3, kde v grafickém rozhraní stačí nastavit (minimálně) dva vstupní parametry, a sice

- *Server Name nebo* IP, kde zadáme IP adresu cílového serveru, na který chceme zaslat např. žádost GET v našem případě, je potřeba adresu zadat bez prefixů *http://* nebo *https://*, jak je uvedeno.
- Dále do pole *Protocol [*HTTP] je defaultně nastavené HTTP.
- *Port Number* je defaultně 80, což odpovídá nezabezpečené verzi HTTP protokolu.
- HTTP Request zde vybíráme podporované HTTP metody pro tuto verzi protokolu, tzn. GET, POST, PUT, PATCH, DELETE a OPTIONS.

- *Path* je volitelné textové pole, kde můžeme zadat např. složku na webu, ze které chceme data získávat, kam chce data vkládat apod.

Apache server byl otestován na dostupnost HTTP/2 modulem od firmy BlazeMeter. Test byl úspěšný, server na tento dotaz reagoval viz zachycení v programu Wireshark.



56	42.268456688	172.16.69.131	172.16.69.129	TCP	74	39464 → 80 [SYN] Se
57	42.268591809	172.16.69.129	172.16.69.131	TCP	74	80 → 39464 [SYN, AC
58	42.268803929	172.16.69.131	172.16.69.129	TCP	66	39464 → 80 [ACK] Se
59	42.269030382	172.16.69.131	172.16.69.129	HTTP2	124	Magic, SETTINGS[0],
60	42.269075840	172.16.69.129	172.16.69.131	TCP	66	80 → 39464 [ACK] Se
61	42.269922236	172.16.69.129	172.16.69.131	HTTP2	103	SETTINGS[0], SETTIN
62	42.270084399	172.16.69.131	172.16.69.129	TCP	66	39464 → 80 [ACK] Se
63	42.273424153	172.16.69.131	172.16.69.129	HTTP2	113	HEADERS[1]: GET /
64	42.273424319	172.16.69.131	172.16.69.129	HTTP2	75	SETTINGS[0]
65	42.273464402	172.16.69.129	172.16.69.131	TCP	66	80 → 39464 [ACK] Se
66	42.274376921	172.16.69.129	172.16.69.131	HTTP2	7306	HEADERS[1]: 200 OK
67	42.274418045	172.16.69.129	172.16.69.131	HTTP2	3638	DATA[1] (text/html)
68	42.276033257	172.16.69.131	172.16.69.129	TCP	66	39464 → 80 [ACK] Se
69	42.276713574	172.16.69.131	172.16.69.129	HTTP2	87	GOAWAY[0]
70	42.276713741	172.16.69.131	172.16.69.129	TCP	66	39464 → 80 [FIN, AC
71	42.276951193	172.16.69.129	172.16.69.131	HTTP2	83	GOAWAY[0]
72	42.277042025	172.16.69.129	172.16.69.131	TCP	66	80 → 39464 [FIN, AC
73	42.277206521	172.16.69.131	172.16.69.129	TCP	60	39464 → 80 [RST] Se
74	42.277276727	172.16.69.131	172.16.69.129	TCP	60	39464 → 80 [RST] Se

Obr. 7.3: Zachycení HTTP/2 modulu v nástroji Wireshark.

8 Tvorba pomalých DoS modulů v nástroji JMeter

Hlavní cíl bakalářské práce je tvorba jednotlivých pomalých útoků popsaných v teoretické části. Útoky jsou realizovány v pěti různých skriptech napsaných v programovacím jazyce Python a následně jsou volány pomocí programovacího jazyka Java v sampleru JMeteru. V následující části práce bude popsána tvorba jednotlivých skriptů i modulů, následně bude ověřena funkčnost všech vytvořených modulů. V rámci kapitoly Slow READ budou popsány postupy, které budou dále využívány u tvorby a testování dalších modulů.

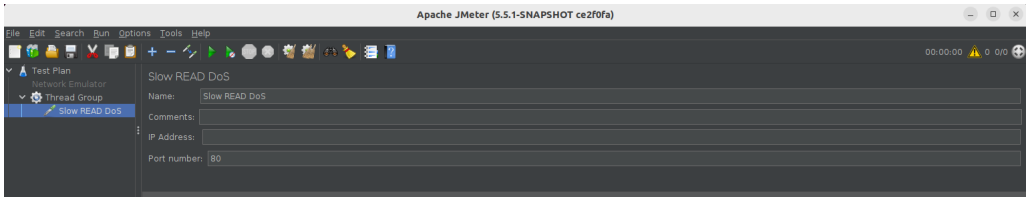
8.1 Tvorba modulu Slow READ

V první části je nutné vytvořit celou logiku v Python skriptu, který se postará o vytvoření žádosti, která následně bude odeslána na server.

Kód kopíruje schéma postupu DoS útoku popsaného v teoretické části útoku Slow READ viz kapitola 3.1. Pro další postup, tedy implementaci skriptu z Pythonu do sampleru JMeteru je nutné ověřit přímo funkčnost tohoto skriptu vůči Apache serveru. Skript má za úkol navázat pouze jedno spojení a je poté na typu serveru za jakou dobu tento neúplný HTTP/2 dotaz ukončí. Pro ověření funkčnosti skriptu musí být server nakonfigurován pro komunikaci s dotazy typu HTTP/2, jedná se tedy o starší než výchozí verzi, zda byl server úspěšně nakonfigurován lze ověřit jednak funkčností skriptu, který vůči tomuto serveru zašle dotaz a neukončí se a dalším způsobem ověření je vidět v podkapitole 7.3.2, bez této konfigurace Apache serveru nelze útok vykonat.

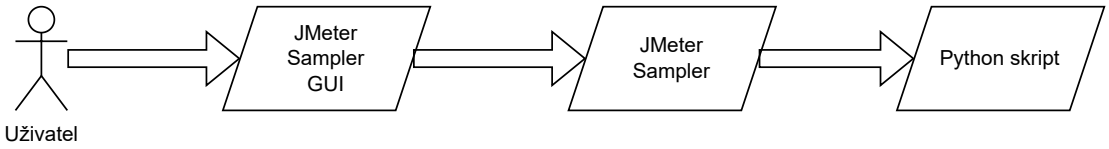
Následná implementace skriptu do JMeter sampleru už není složitá, tento skript má pouze dva parametry, jedním je adresa HTTP/2 serveru, na kterou se bude v rámci útoku skript dotazovat a dalším parametrem je port na kterém server běží.

V hlavním GUI je tedy pouze vytvořené pole pro zadání těchto dvou parametrů. Sampler si následně přebere zadané hodnoty od GUI a jeho úlohou je hodnoty přidat ke skriptu a skript následně spustit. Výsledný modul poté vypadá viz obrázek 8.1.



Obr. 8.1: HTTP/2 Slow READ modul v nástroji JMeter.

Komunikace pro předávání dat v rámci nástroje JMeter mezi jednotlivými částmi programu je zobrazena na obr. 8.2. Nejprve uživatel zadá hodnoty do grafického uživatelského rozhraní, následně je toto GUI zasílá do sampleru a ten je přiřadí na vstup Python skriptu.

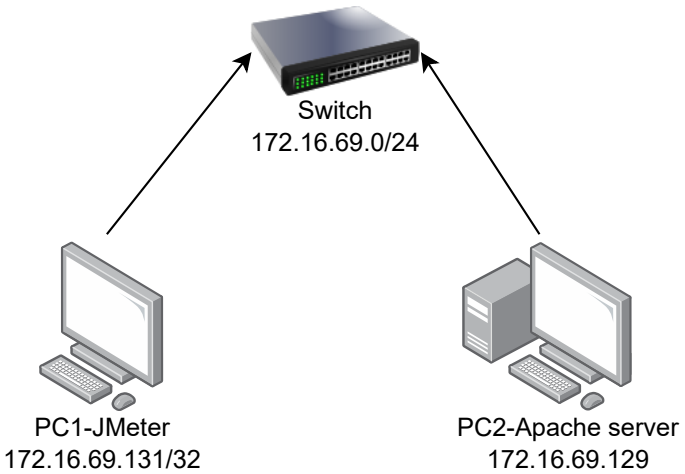


Obr. 8.2: Zobrazení komunikace částí JMeteru.

8.2 Otestování modulu Slow READ

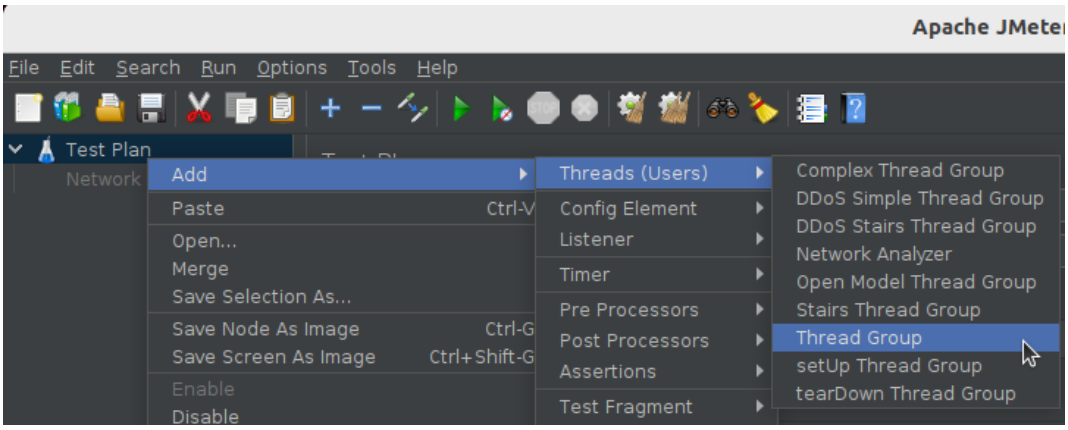
V testovací části byly vytvořena dvě virtuální prostřední propojené virtuálním přepínačem, kde na jednom je nainstalován Apache JMeter v nejnovější verzi a slouží k zaslání útoku a ve druhém virtuálním prostředí je nainstalován Apache server ve verzi 2.4.52. Verze Ubuntu je na obou stranách 22.04.2 LTS. Situace včetně IP adres je zobrazena na obr. 8.3.

Pro otestování modulu je nutné vložit zkompilovaný jar soubor přímo do originálního JMeteru, jak bylo popsáno v kapitole 7.3.2. Oproti předešlému popisu je zde ještě nutnost vložit vytvořený skript do cesty jmeter/lib/ext/ddos, neboť sampler je tímto způsobem naprogramován, přebírá z této cesty soubor, který zavolá na vstupu.

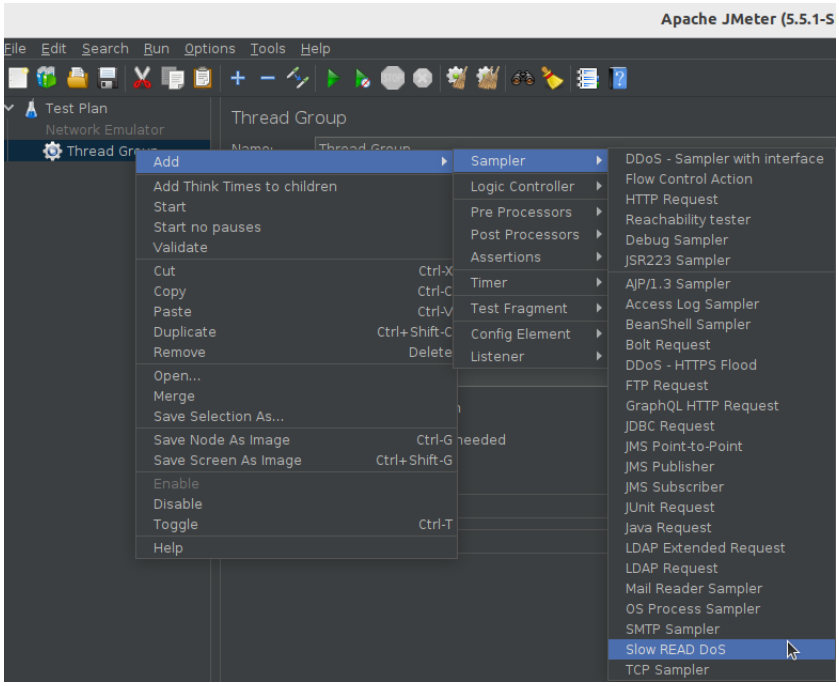


Obr. 8.3: Schéma zapojení praktické části.

Po vložení jar souboru a skriptu lze provést test. Při otevření JMeteru se pravým tlačítkem klikne na Test Plan, následně Add, Threads (Users) a vybere se Thread Group viz 8.4. Tato slouží pro počet žádostí zaslaných na server, řeší tedy vytvoření DoS útoku ve smyslu, že Python skript generuje jeden dotaz a díky Thread Group je možné počet dotazů násobit číslem, které se zadá.



Obr. 8.4: Výběr Thread Group.



Obr. 8.5: Výběr konkrétního modulu.

Dalším krokem pro provedení testování je spuštění vybraného modulu na Thread Group, pravým tlačítkem a vybírá se Add, Sampler a Slow READ Sampler viz 8.5, obdobně pro další Samplery, které jsou určené k testování.

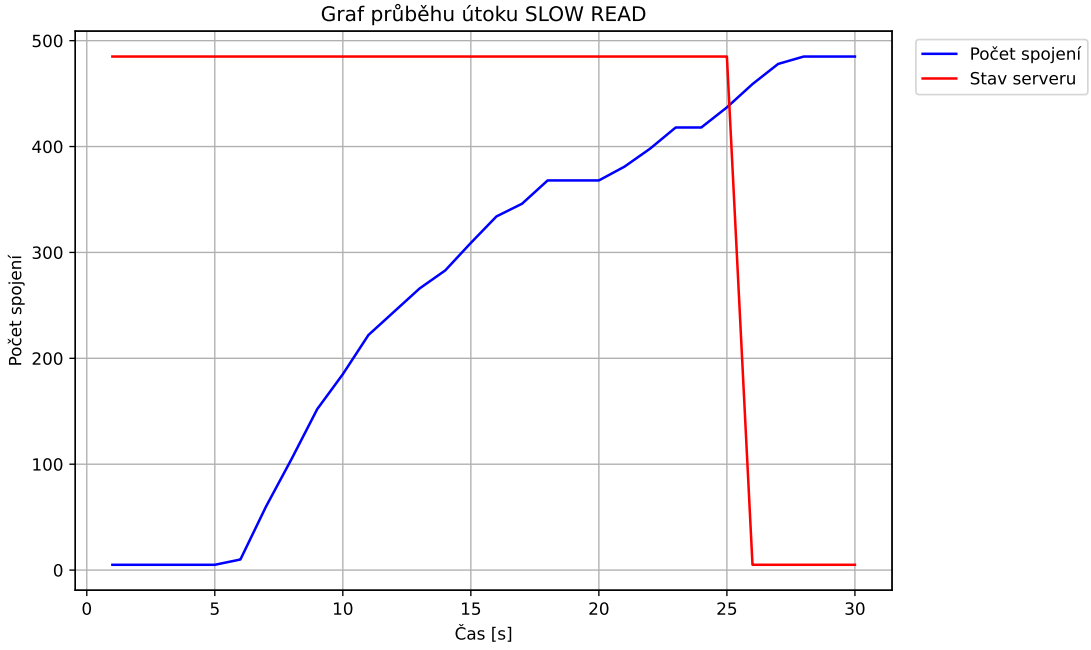
Je otevřen modul viz 8.1, kde je nutné zadat parametry v podobě IP adresy a portu, na kterém server běží, jak bylo dříve zmíněno. Pro naměření hodnot byl vytvořen bash skript, který do csv souboru zapisuje čas, počet otevřených spojení a status serveru (zda je aktivní nebo neaktivní), tedy hodnotami 0 a 1, kde 0 je neaktivní a 1 je aktivní.

Z celé této metriky je následně vytvořen graf přes skript v jazyce Python. Výstupem z měření je tabulka naměřených a vypočtených hodnot a graf, který zobrazuje jednotlivé naměřené hodnoty v závislosti na čase. Tabulka obsahuje několik málo naměřených hodnot, větší část bude viditelná přímo v grafu.

V naměřených hodnotách v tabulce lze vidět, že se někdy mezi 21 a 30 sekundami stal server nedostupným.

Tab. 8.1: Tabulka naměřených z útoku Slow READ při 500 žádostech.

Čas (s)	Počet spojení	Status serveru (0 nebo 1)
1	5	1
10	185	1
20	368	1
30	485	0



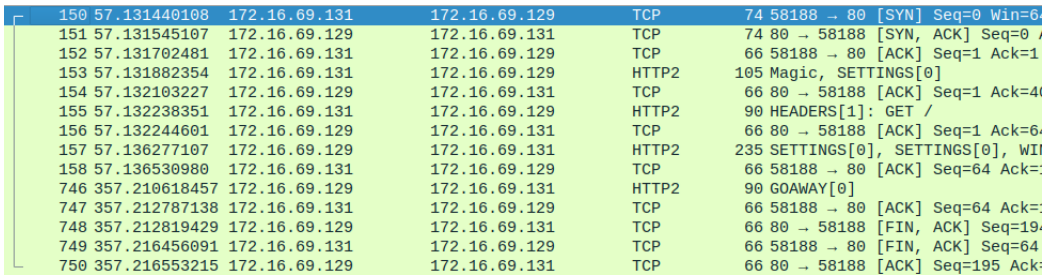
Obr. 8.6: Zobrazení průběhu útoku Slow READ.

Z grafu útoku je patrné, že přibližně kolem 450 otevřených spojení se server stal nedostupným. Měření probíhalo po sekundových intervalech, je tedy možné, že

k výpadku serveru došlo v dřívější době, než je uvedena. Server také spojení uzavírá až po 300 sekundách, lze tedy vidět průběh prvních 30 sekund.

Tedy je dokázáno, že útok Slow READ na Apache serveru ve verzi 2.4.52 je funkční a lze ním server učinit nedostupným.

V zachycené komunikaci v programu Wireshark lze vidět komunikace mezi klientem a serverem. Po 300 sekundách komunikace je evidentní uzavření spojení serverem, který k tomu využil rámec `GOAWAY`.

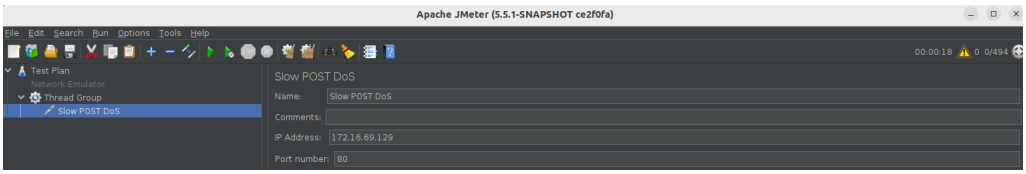


Obr. 8.7: Zachycení průběhu útoku Slow READ v programu Wireshark.

8.3 Tvorba modulu Slow POST

Struktura modulu je prakticky stejná a postup jako u modulu Slow READ. Skript se samozřejmě liší s ohledem na útok, je zde metoda, která zajišťuje zaslání tohoto neúplného požadavku na stranu serveru.

Grafické uživatelské rozhraní obsahuje dvě vstupní pole jako parametr pro skript viz 8.8.



Obr. 8.8: HTTP/2 Slow POST modul v nástroji JMeter.

Práce s modulem je stejná jako v popsaném modulu Slow READ.

8.4 Otestování modulu Slow POST

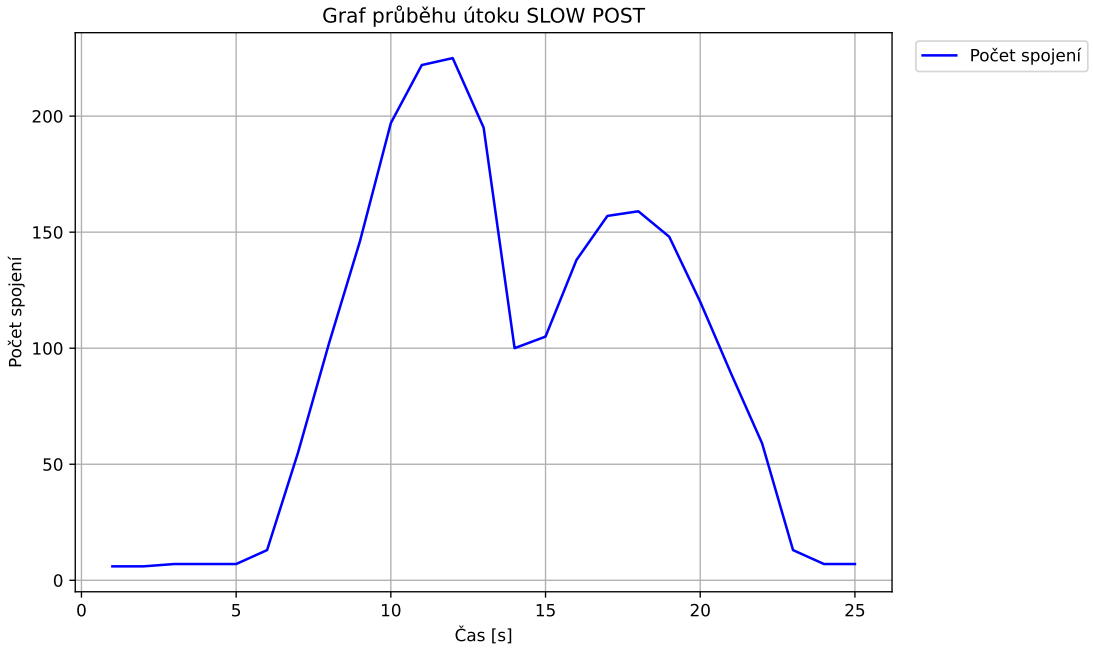
Je sestaveno testovací prostředí obdobně jako u předešlého útoku. Na straně klienta je spuštěn modul Slow POST v nástroji JMeter. Na server je zasláno 500 žádostí pomocí Thread Groupy. Při testování tohoto modulu bylo zjištěno, že Apache server

v této verzi je vůči tomuto útoku odolný, jinými slovy, při zvyšujícím se počtu spojení server vyhodnotí, že nejde o legitimní provoz a tomuto zamezí zavíráním spojení.

Tab. 8.2: Tabulka naměřených z útoku Slow POST při 500 žádostech.

Čas (s)	Počet spojení	Status serveru (0 nebo 1)
7	55	1
12	225	1
15	105	1
23	13	1

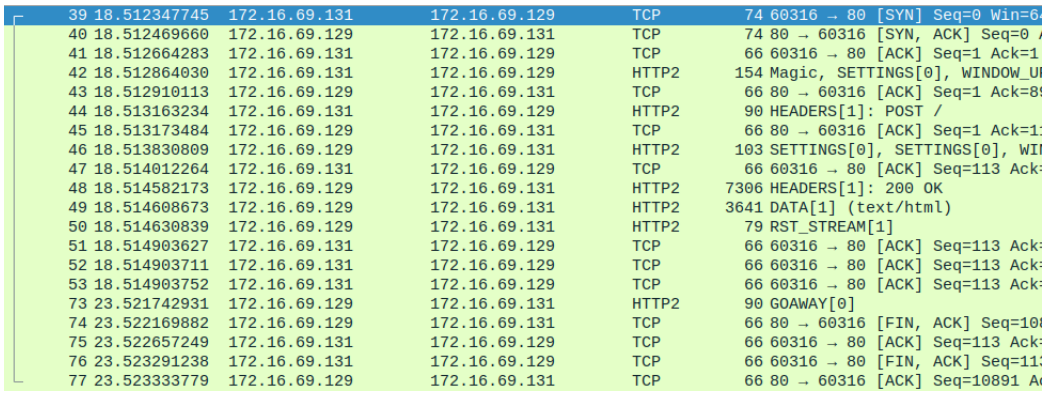
V tabulce 8.2 lze vidět, že server po zvyšujícím se počtu žádostí typu POST začal tyto spojení zavírat, maxima dosáhl ve dvanácté sekundě.



Obr. 8.9: Zobrazení průběhu útoku Slow POST.

Na grafu 8.9, který vykresluje více hodnot počtu spojení v závislosti na čase lze vidět, že klient navazoval spojení a kolem 12 sekundy došlo k zavírání otevřených spojení. Důvod proč došlo opět k vzrůstu spojení je, že server začal tyto probíhající spojení uzavírat ještě dříve, než se stihl útok dokončit.

Na zachycené komunikaci 8.10 lze vidět zaslaný jeden neúplný požadavek a server zde spojení ukončil rychle v řádu pěti sekund rámcem `GOAWAY`.

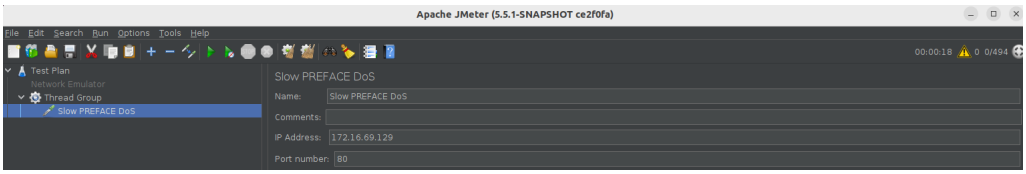


39	18.512347745	172.16.69.131	172.16.69.129	TCP	74	60316 → 80 [SYN] Seq=0 Win=6
40	18.512469660	172.16.69.129	172.16.69.131	TCP	74	80 → 60316 [SYN, ACK] Seq=0
41	18.512664283	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [ACK] Seq=1 Ack=1
42	18.512864030	172.16.69.131	172.16.69.129	HTTP2	154	Magic, SETTINGS[0], WINDOW_U
43	18.512910113	172.16.69.129	172.16.69.131	TCP	66	80 → 60316 [ACK] Seq=1 Ack=8
44	18.513163234	172.16.69.131	172.16.69.129	HTTP2	90	HEADERS[1]: POST /
45	18.513173484	172.16.69.129	172.16.69.131	TCP	66	80 → 60316 [ACK] Seq=1 Ack=1
46	18.513830809	172.16.69.129	172.16.69.131	HTTP2	103	SETTINGS[0], SETTINGS[0], WI
47	18.514012264	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [ACK] Seq=113 Ack
48	18.514582173	172.16.69.129	172.16.69.131	HTTP2	7306	HEADERS[1]: 200 OK
49	18.514608673	172.16.69.129	172.16.69.131	HTTP2	3641	DATA[1] (text/html)
50	18.514630839	172.16.69.129	172.16.69.131	HTTP2	79	RST_STREAM[1]
51	18.514903627	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [ACK] Seq=113 Ack
52	18.514903711	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [ACK] Seq=113 Ack
53	18.514903752	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [ACK] Seq=113 Ack
73	23.521742931	172.16.69.129	172.16.69.131	HTTP2	90	GOAWAY[0]
74	23.522169882	172.16.69.129	172.16.69.131	TCP	66	80 → 60316 [FIN, ACK] Seq=10
75	23.522657249	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [ACK] Seq=113 Ack
76	23.523291238	172.16.69.131	172.16.69.129	TCP	66	60316 → 80 [FIN, ACK] Seq=11
77	23.523333779	172.16.69.129	172.16.69.131	TCP	66	80 → 60316 [ACK] Seq=10891 A

Obr. 8.10: Zachycení průběhu útoku Slow POST v programu Wireshark.

8.5 Tvorba modulu Slow PREFACE

Dle teoretické znalosti z kap. 3.3 skript opět obsahuje hlavní metodu, která naváže spojení a drží jej otevřené.



Obr. 8.11: HTTP/2 Slow PREFACE modul v nástroji JMeter.

Modul obsahuje dvě vstupní pole, jedno pro zadaní IP adresy a druhé pro zadaní portu. Pro vykonání DoS útoku je nutné v Thread Group zadat množství zaslaných žádostí na server. Modul využívající externího generátoru skriptu v Pythonu generuje pouze jednu neúplnou žádost.

8.6 Otestování modulu Slow PREFACE

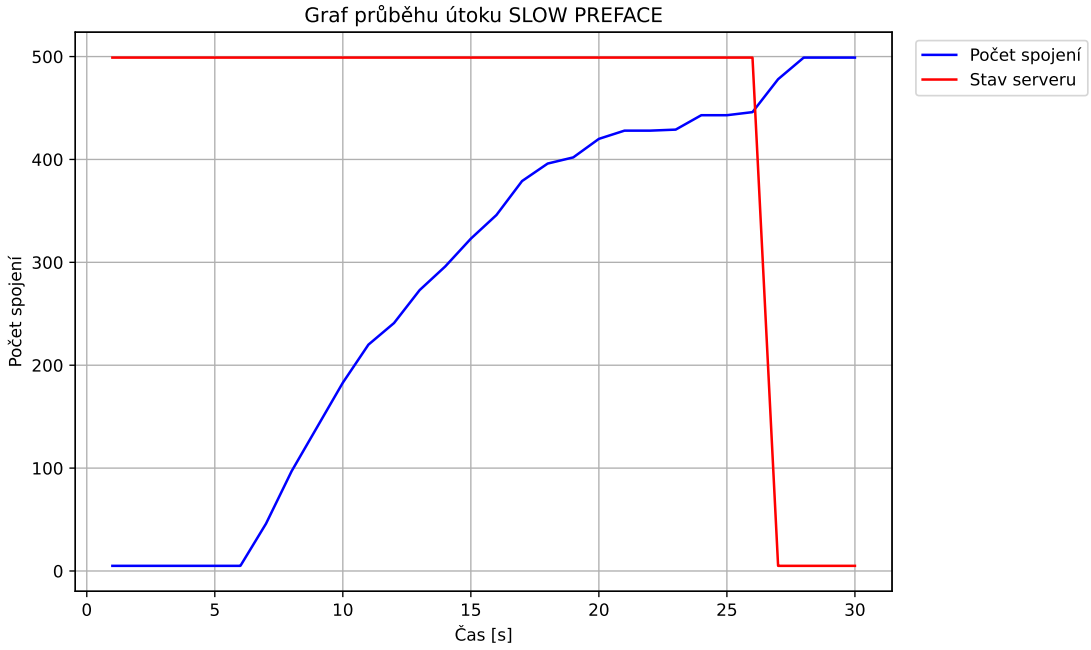
Průběh testu je ve stejném prostředí jako předešlé útoky. Na straně klienta je spuštěn modul Slow PREFACE pro nástroj JMeter. Je zasláno 500 žádostí na server, tabulka a graf ukazují chování serveru.

V tabulce 8.3 lze vidět, že server vytvořil spojení a následně zhruba ve 26 sekundách byla vyčerpaná šířka pásma a server se stal nedostupným. Pravděpodobně k tomuto došlo už v 25 sekundách, nicméně v rámci skriptu je doba na uložení nových dat 1 sekunda, je tedy možné, že tato prodleva způsobila, že je v grafu patrné spadení až po 25 sekundě. Dále se také může jednat o problém JMeteru, který zasílá žádosti o spojení postupně (i když s velmi malou prodlevou).

Tab. 8.3: Tabulka naměřených z útoku Slow PREFACE při 500 žádostech.

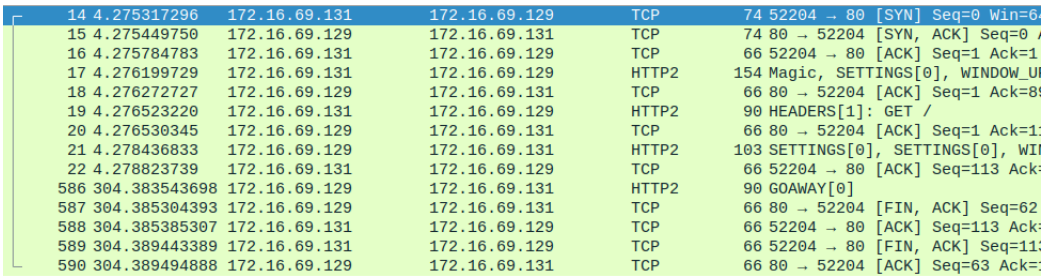
Čas (s)	Počet spojení	Status serveru (0 nebo 1)
10	183	1
20	420	1
25	443	1
30	499	0

Test tedy byl proveden úspěšně, server následně vždy když tímto způsobem otevírá spojení a nic se neděje po dobu 300 sekund, tak spojení uzavírá, jedná se o základní hodnotu nastavení Apache serveru.



Obr. 8.12: Zobrazení průběhu útoku Slow PREFACE.

Útokem Slow PREFACE lze tedy vytvořit nedostupnost serveru. Toto nastane někdy po 420 žádostech odeslaných na server.



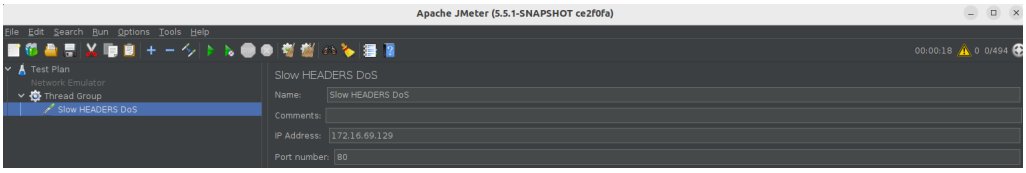
14	4.275317296	172.16.69.131	172.16.69.129	TCP	74	52204 → 80 [SYN] Seq=0 Win=6
15	4.275449750	172.16.69.129	172.16.69.131	TCP	74	80 → 52204 [SYN, ACK] Seq=0
16	4.275784783	172.16.69.131	172.16.69.129	TCP	66	52204 → 80 [ACK] Seq=1 Ack=1
17	4.276199729	172.16.69.131	172.16.69.129	HTTP2	154	Magic, SETTINGS[0], WINDOW_U
18	4.276272727	172.16.69.129	172.16.69.131	TCP	66	80 → 52204 [ACK] Seq=1 Ack=8
19	4.276523220	172.16.69.131	172.16.69.129	HTTP2	90	HEADERS[1]: GET /
20	4.276530345	172.16.69.129	172.16.69.131	TCP	66	80 → 52204 [ACK] Seq=1 Ack=1
21	4.278436833	172.16.69.129	172.16.69.131	HTTP2	103	SETTINGS[0], SETTINGS[0], WI
22	4.278823739	172.16.69.131	172.16.69.129	TCP	66	52204 → 80 [ACK] Seq=113 Ack
586	304.383543698	172.16.69.129	172.16.69.131	HTTP2	90	GOAWAY[0]
587	304.385304393	172.16.69.129	172.16.69.131	TCP	66	80 → 52204 [FIN, ACK] Seq=62
588	304.385385307	172.16.69.131	172.16.69.129	TCP	66	52204 → 80 [ACK] Seq=113 Ack
589	304.389443389	172.16.69.131	172.16.69.129	TCP	66	52204 → 80 [FIN, ACK] Seq=11
590	304.389494888	172.16.69.129	172.16.69.131	TCP	66	80 → 52204 [ACK] Seq=63 Ack=

Obr. 8.13: Zachycení průběhu útoku Slow PREFACE v programu Wireshark.

Na zachycení programem Wireshark lze vidět, že server spojení uzavřel po 300 sekundách a zaslal rámec `GOAWAY`.

8.7 Tvorba modulu Slow HEADERS

Pro tento modul byl vytvořen skript slowheaders.py, který obsahuje metodu pro vytvoření zranitelnosti Slow HEADERS.



Obr. 8.14: HTTP/2 Slow HEADERS modul v nástroji JMeter.

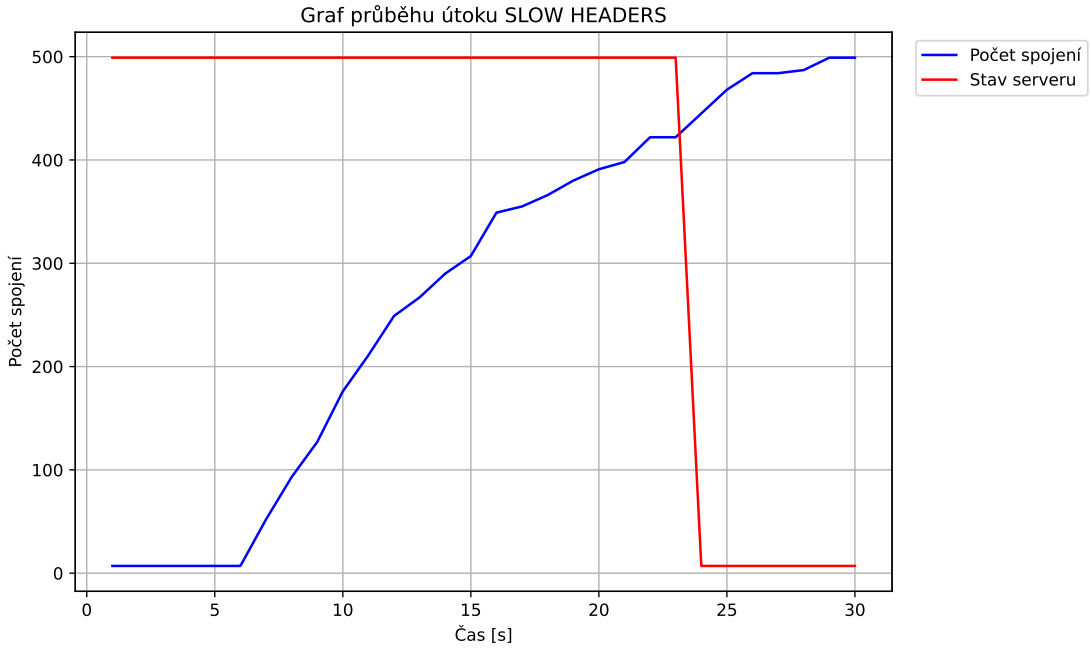
Modul obsahuje dva textové vstupy, kde jeden je IP adresa a druhý je port, přednastavenou hodnotou je 80. Opět modul způsobuje pouze vytvoření jednoho spojení a pro spuštění více spojení je nutné tento modul přidat spolu s Thread Group a nastavit zde počet takto vytvořených spojení.

8.8 Otestování modulu Slow HEADERS

Průběh testu probíhá ve stejném prostředí a za stejných podmínek jako předešlé testy, podmínky a prostředí jsou popsány v rámci kap. 8.2. Pro testování bylo nastaveno počet spojení na hodnotu 500 žádostí.

Tab. 8.4: Tabulka naměřených z útoku Slow HEADERS při 500 žádostech.

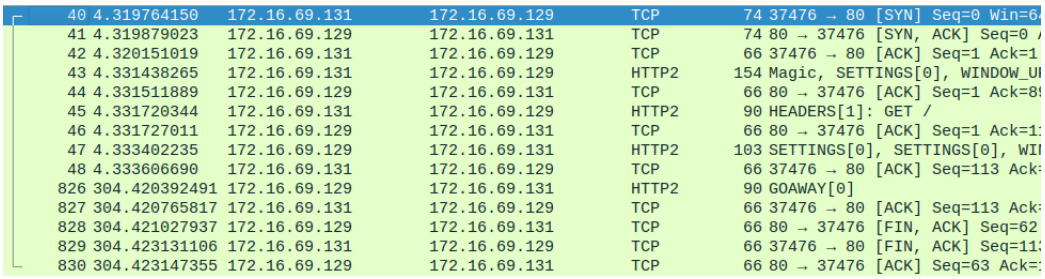
Čas (s)	Počet spojení	Status serveru (0 nebo 1)
10	176	1
20	391	1
25	468	0
30	499	0



Obr. 8.15: Zobrazení průběhu útoku Slow HEADERS.

Oproti předešlým testům lze vidět v grafu 8.15, že server spojení z počátku otevíral pomaleji a také v porovnání pád serveru nastal dříve.

Na zachycení lze opět pozorovat uzavření spojení po 300 sekundách od vytvoření komunikace a po těchto 300 sekundách server poslal rámec `GOAWAY`.

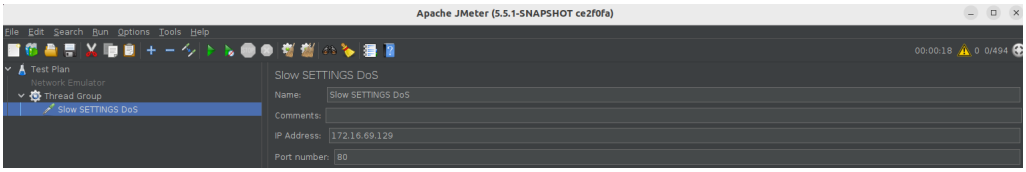


40	4.319764150	172.16.69.131	172.16.69.129	TCP	74	37476 → 80 [SYN] Seq=0 Win=6
41	4.319879023	172.16.69.129	172.16.69.131	TCP	74	80 → 37476 [SYN, ACK] Seq=0
42	4.320151019	172.16.69.131	172.16.69.129	TCP	66	37476 → 80 [ACK] Seq=1 Ack=1
43	4.331438265	172.16.69.131	172.16.69.129	HTTP2	154	Magic, SETTINGS[0], WINDOW_U
44	4.331511889	172.16.69.129	172.16.69.131	TCP	66	80 → 37476 [ACK] Seq=1 Ack=8
45	4.331720344	172.16.69.131	172.16.69.129	HTTP2	90	HEADERS[1]: GET /
46	4.331727011	172.16.69.129	172.16.69.131	TCP	66	80 → 37476 [ACK] Seq=1 Ack=1
47	4.333402235	172.16.69.129	172.16.69.131	HTTP2	103	SETTINGS[0], SETTINGS[0], WI
48	4.333606690	172.16.69.131	172.16.69.129	TCP	66	37476 → 80 [ACK] Seq=113 Ack
826	304.420392491	172.16.69.129	172.16.69.131	HTTP2	90	GOAWAY[0]
827	304.420765817	172.16.69.131	172.16.69.129	TCP	66	37476 → 80 [ACK] Seq=113 Ack
828	304.421027937	172.16.69.129	172.16.69.131	TCP	66	80 → 37476 [FIN, ACK] Seq=62
829	304.423131106	172.16.69.131	172.16.69.129	TCP	66	37476 → 80 [FIN, ACK] Seq=11
830	304.423147355	172.16.69.129	172.16.69.131	TCP	66	80 → 37476 [ACK] Seq=63 Ack=

Obr. 8.16: Zachycení průběhu útoku Slow HEADERS v programu Wireshark.

8.9 Tvorba modulu Slow SETTINGS

Pro vytvoření tohoto modulu je prvně nutné vytvořit skript, který bude uvozovat spojení. Tento skript slowsettings.py obsahuje metodu, která toto uvodí dle teoretické části 3.5. Následně je tohoto skriptu využito jako externího generátoru v nástroji JMeter.



Obr. 8.17: HTTP/2 Slow SETTINGS modul v nástroji JMeter.

Pro tento modul bylo vytvořeno GUI, které se chová stejně jako předešlé moduly. Opět se zadává IP adresa a port serveru, který je přednastavený na 80.

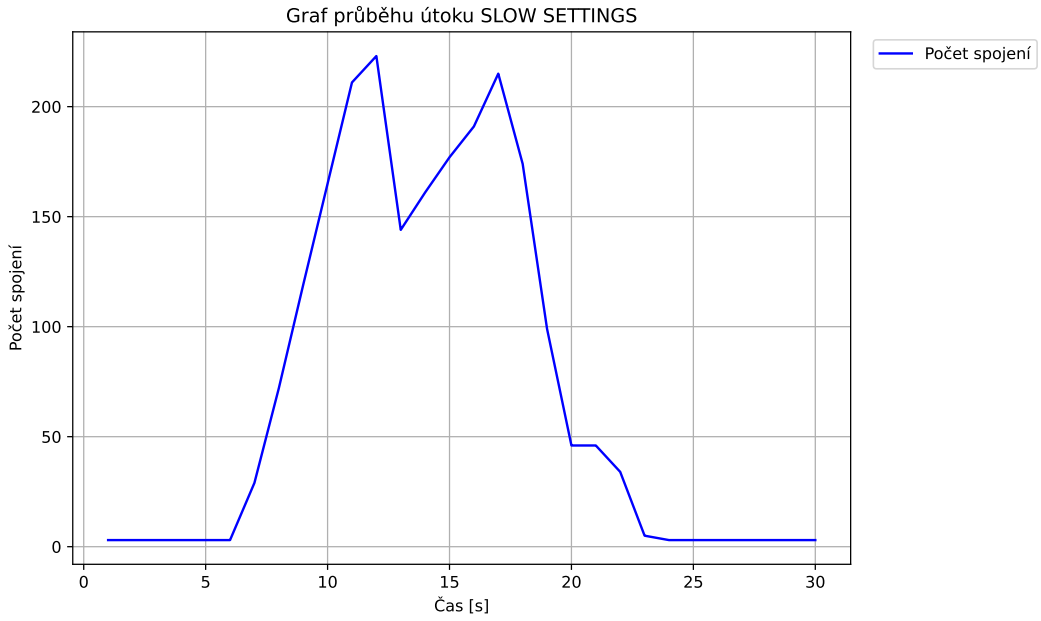
8.10 Otestování modulu Slow SETTINGS

Pro otestování modulu je využito nastavení sepsáno v kap. 8.2. Opět pro test bylo nastaveno 500 spojení.

Tab. 8.5: Tabulka naměřených z útoku Slow SETTINGS při 500 spojeních.

Čas (s)	Počet spojení	Status serveru (0 nebo 1)
10	165	1
12	223	1
18	174	1
19	99	1

Shození serveru opět neproběhlo úspěšně, neboť Apache server v této verzi je proti této zranitelnosti odolný. A ve chvíli, kdy se začne zvyšovat počet takto generovaného spojení viz 3.5, tak server tyto spojení začne automaticky uzavírat.



Obr. 8.18: Zobrazení průběhu útoku Slow SETTINGS.

Na grafu lze vidět zvýšení žádostí i po pádu a toto je opět způsobeno jako v případě Slow POST, tím, že ještě nebyly v době kdy server začal spojení zahazovat navázané všechna spojení. Tj. začala se navazovat znovu a znovu je server začal zavírat.

Dále v rámci zachycení paketů lze opět vidět úspěšné navázání HTTP/2 a následné ukončení rámcem `GOAWAY`.

1	0.000000000	172.16.69.131	172.16.69.129	TCP	74	58370 → 80 [SYN] Seq=0 Win=64
2	0.000127830	172.16.69.129	172.16.69.131	TCP	74	80 → 58370 [SYN, ACK] Seq=0 A
3	0.000410863	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=1 Ack=1
4	0.007623272	172.16.69.131	172.16.69.129	HTTP2	154	Magic, SETTINGS[0], WINDOW_UI
5	0.007726811	172.16.69.129	172.16.69.131	TCP	66	80 → 58370 [ACK] Seq=1 Ack=85
6	0.008035218	172.16.69.131	172.16.69.129	HTTP2	90	HEADERS[1]: GET /
7	0.008046301	172.16.69.129	172.16.69.131	TCP	66	80 → 58370 [ACK] Seq=1 Ack=11
8	0.009017981	172.16.69.129	172.16.69.131	HTTP2	103	SETTINGS[0], SETTINGS[0], WI
9	0.009260182	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=113 Ack=
10	0.009799207	172.16.69.129	172.16.69.131	HTTP2	2962	HEADERS[1]: 200 OK
11	0.009817998	172.16.69.129	172.16.69.131	TCP	2962	80 → 58370 [PSH, ACK] Seq=29
12	0.009829165	172.16.69.129	172.16.69.131	TCP	2962	80 → 58370 [PSH, ACK] Seq=58
13	0.009840498	172.16.69.129	172.16.69.131	HTTP2	2193	DATA[1] (text/html)
14	0.010226611	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=113 Ack=
15	0.010226653	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=113 Ack=
16	0.010226653	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=113 Ack=
17	0.010226695	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=113 Ack=
24	5.017009688	172.16.69.129	172.16.69.131	HTTP2	90	GOAWAY[0]
25	5.017415760	172.16.69.129	172.16.69.131	TCP	66	80 → 58370 [FIN, ACK] Seq=10
26	5.017948454	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [ACK] Seq=113 Ack=
27	5.018673226	172.16.69.131	172.16.69.129	TCP	66	58370 → 80 [FIN, ACK] Seq=11
28	5.018710808	172.16.69.129	172.16.69.131	TCP	66	80 → 58370 [ACK] Seq=10878 A

Obr. 8.19: Zachycení průběhu útoku Slow SETTINGS v programu Wireshark.

Závěr

Cílem práce bylo vytvořit moduly v nástroji Apache JMeter, které budou schopny testovat protokol HTTP/2. Dále nakonfigurovat Apache server, aby byl schopen tyto požadavky typu HTTP/2 zpracovávat.

V první kapitole byly popsány komunikační modely TCP/IP a ISO/OSI, jednotlivé vrstvy těchto modelů a komunikace mezi klientem a serverem v rámci „three-way“ handshake.

Další kapitola se zabývá popsáním útoků DoS a DDoS, popisuje hlavní rozdíly mezi nimi a definuje jednotlivé kategorie.

Třetí kapitola se věnuje podrobnému popisu pěti typů Slow DoS útoků a na obrázcích v této kapitole je vidět jejich průběh.

Následující kapitola popisuje v čase jednotlivé verze HTTP protokolu, porovnává je mezi sebou, zobrazuje jejich nedostatky a výhody oproti předešlým.

Další kapitola je vyhrazena pro protokol HTTP/2, kde popisuje rámce, které se v něm objevují, popisuje a celkové postupy při sestavování spojení, a také chybové kódy.

V šesté kapitole je popsána HPACK komprese hlavičky, jak probíhá, jaké postupy musí být dodrženy při vytváření tabulek pro kompresi.

Další kapitola se obecně zabývá nástrojem Apache JMeter, popisuje, co je nutné dodržet při tvorbě nového modulu, jak JMeter spustit a dále se zabývá konfigurací Apache serveru pro verzi HTTP/2 a následným otestováním funkčnosti.

V poslední kapitole je popsán vývoj modulů pro nástroj JMeter ze třetí kapitoly. Následně jsou moduly otestovány na nakonfigurovaném serveru a zobrazeny ve vizualizacích.

Závěrem bylo dosaženo úspěšného otestování všech modulů a konfigurace Apache HTTP/2 serveru v jeho verzi 2.4.52. Moduly Slow READ, Slow PREFACE a Slow HEADERS fungovali očekávaně, server se těmito třemi moduly podařilo znepřístupnit. Na další dva moduly Slow POST a Slow SETTINGS se ukázalo v průběhu testování, že server je vůči nim odolný a tyto spojení po pěti sekundách zahodí.

Literatura

[1] Imperva *OSI Model*[online]. [cit. 11. 10. 2022]. Dostupné z URL: <https://www.imperva.com/learn/application-security/osi-model/

[2] IJS *Referenční model ISO/OSI*[online]. [cit. 17. 10.2022] Dostupné z URL: <http://ijs.8u.cz/index.php/standardizace-v-pocitacovych-sitich/referencni-model-iso-osi

[3] Crypto-IT *TCP/IP Protocols*[online]. [cit. 15. 11. 2022] Dostupné z URL: <http://www.crypto-it.net/eng/theory/tcp-ip-protocols.html

[4] Techopedia *Three-Way Handshake*[online]. [cit. 23. 11. 2022] Dostupné z URL: <https://www.techopedia.com/definition/10339/three-way-handshake

[5] Mozilla *Evolution of HTTP*[online]. [cit. 27. 11. 2022] Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

[6] Apache *Apache JMeter*[online]. [cit. 01. 12. 2022] Dostupné z URL: <https://jmeter.apache.org/

[7] Perfmetrix *JMeter – Folder structure*[online]. [cit. 10. 12. 2022] Dostupné z URL: <https://www.perfmatrix.com/jmeter-folder-structure/

[8] Wallarm *What Is HTTP/2 And How Is It Different From HTTP/1?*[online]. [cit. 10. 12. 2022] Dostupné z URL: <https://www.wallarm.com/what/what-is-http-2-and-how-is-it-different-from-http-1

[9] W3 *Hypertext Transfer Protocol – HTTP/1.0*[online]. [cit. 11. 12. 2022] Dostupné z URL: <https://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html#Methods

[10] rfc-editor *Hypertext Transfer Protocol – HTTP/1.1*[online]. [cit. 11. 12. 2022] Dostupné z URL: <https://www.rfc-editor.org/rfc/rfc2616

[11] STACKPATH *SHIFTING FROM SPDY TO HTTP/2*[online]. [cit. 11. 12. 2022] Dostupné z URL: <https://www.stackpath.com/blog/spdy-to-http2

[12] HTTPWG *HTTP/2*[online]. [cit. 11. 12. 2022] Dostupné z URL: <https://httpwg.org/specs/rfc9113.html#starting

[13] HTTPWG *HPACK: Header Compression for HTTP/2*[online]. [cit. 11. 12. 2022] Dostupné z URL: <https://httpwg.org/specs/rfc7541.html#header.encoding

[14] POLLARD, B.: *HTTP/2 in Action*. Shelter Island, NY: Manning Publications Co., [2019]. 416 s. ISBN 9781617295164.

[15] man7 *trafgen(8) — Linux manual page*[online]. [cit. 11. 12. 2022] Dostupné z URL: <https://man7.org/linux/man-pages/man8/trafgen.8.html

[16] linode *Configure HTTP/2 on Apache*[online]. [cit. 12. 12. 2022] Dostupné z URL: <https://www.linode.com/docs/guides/how-to-configure-http-2-on-apache/

[17] sunnyvalley *DoS and DDoS Attacks: What are Their Differences?*[online]. [cit. 18. 02. 2023] Dostupné z URL: <https://www.sunnyvalley.io/docs/network-security-tutorials/dos-vs-ddos-attacks

[18] imperva *DDoS Attacks*[online]. [cit. 18. 02. 2023] Dostupné z URL: <https://www.imperva.com/learn/ddos/ddos-attacks/

[19] Eset *DDoS útok*[online]. [cit. 18. 02. 2023] Dostupné z URL: <https://www.eset.com/cz/ddos-utok/

[20] RedSpam *What Is A Low And Slow DDoS Attack? | DDoS Protection*[online]. [cit. 18. 02. 2023] Dostupné z URL: <https://www.redspam.com/blog/2021/aug/what-low-and-slow-ddos-attack-ddos-protection

[21] ScienceDirect *Slow rate denial of service attacks against HTTP/2 and detection*[online]. [cit. 18. 02. 2023] Dostupné z URL: <https://www.sciencedirect.com/science/article/pii/S0167404817301980?fr=RR-1&ref=cra_js_challenge

[22] Baeldung *HTTP: 1.0 vs. 1.1 vs 2.0 vs. 3.0*[online]. [cit. 19. 02. 2023] Dostupné z URL: <https://www.baeldung.com/cs/http-versions

[23] DebugBear *A Comprehensive Guide To HTTP/3 And QUIC*[online]. [cit. 19. 04. 2023] Dostupné z URL: <https://www.debugbear.com/blog/http3-quic-protocol-guide

[24] Rascasone *HTTPS V KOSTCE: CO TO JE, JAK FUNGUJE A JAK NA NĚJ PŘEJÍT*[online]. [cit. 19. 04. 2023] Dostupné z URL: <https://www.rascasone.com/cs/blog/co-je-https-http-ssl-tls

[25] Gigamon *What Is TLS 1.2, and Why Should You (Still) Care?*[online]. [cit. 19. 05. 2023] Dostupné z URL: <https://blog.gigamon.com/2021/07/14/what-is-tls-1-2-and-why-should-you-still-care/

[26] sslmarket *TLS 1.3 – Seznamte se s novým bezpečnostním standardem*[online]. [cit. 20. 05. 2023] Dostupné z URL: <https://www.sslmarket.cz/blog/tls-1-3-seznamte-se-novym-bezpecnostnim-standardem

[27] imperva *Ping of Death (POD)*[online]. [cit. 20. 05. 2023] Dostupné z URL: <https://www.imperva.com/learn/ddos/ping-of-death/

Seznam symbolů a zkratek

HTTP/2 Hyper Text Transfer Protocol ver. 2

ISO/OSI International Organization for Standardization/Open Systems Interconnection

TCP/IP Transmission Control Protocol over Internet Protocol

DoS Denial of Service

DDoS Distributed Denial of Service

HTTP Hyper Text Transfer Protocol

HPACK Header Compression for HTTP/2

DNS Domain Name Server

IP Internet Protocol

SYN Synchronize Sequence Number

ACK Acknowledge Sequence Number

botnet síť počítačů infikovaných speciálním softwarem, který umožňuje řízení této sítě

bps bits per second

UDP Flood User Datagram Protocol Flood

ICMP Flood Internet Control Message Protocol

pps packets per second

rps requests per second

ASCII American Standard Code for Information Interchange

HTTPS Hypertext Transfer Protocol Secure

HOL blocking Head-Of-Line blocking

TCP Transmission Control Protocol

SSL Secure Sockets Layer

URL Uniform Resource Locator

GUI	Graphical User Interface
CSS	Cascading Style Sheets
XML	Extensible Markup Language
PHP	Hypertext Preprocessor
QUIC	Quick UDP Internet Connections
TLS	Transport Layer Security
UDP	User Datagram Protocol
HTTPS	Hypertext Transfer Protocol Secure
RTT	Round Trip Time Resumption

Seznam příloh

A Obsah elektronické přílohy

Příloha obsahuje adresář `/Bash_script/`, který obsahuje skript pro naměření hodnot pro HTTP/2 moduly. Adresář `/Modules_jar/` obsahuje spustitelný soubor. Dále adresář `/Modules_src/` obsahuje zdrojové kódy Javy pro pět různých HTTP/2 pomalých útoků. A poslední adresář `/Python_scripts/` obsahuje skript pro generování grafu a zdrojové kódy Python skriptů, které slouží jako externí generátory pro vytvořené moduly.

- `/` ..... kořenový adresář přiloženého archivu
  - `Bash_script` ..... skript pro naměření hodnot
    - `script.sh`
  - `Modules_jar` ..... spustitelný jar soubor pro všechny vytvořené moduly
    - `Ddos.jar`
  - `Modules_src` ..... zdrojové kódy vytvořených modulů
    - `slowhttp2`
      - `SlowHttp2Gui.java`
      - `SlowHttp2Sampler.java`
    - `slowhttp2headers`
      - `SlowHttp2HeadersGui.java`
      - `SlowHttp2HeadersSampler.java`
    - `slowhttp2post`
      - `SlowHttp2PostGui.java`
      - `SlowHttp2PostSampler.java`
    - `slowhttp2preface`
      - `SlowHttp2PrefaceGui.java`
      - `SlowHttp2PrefaceSampler.java`
    - `slowhttp2settings`
      - `SlowHttp2SettingsGui.java`
      - `SlowHttp2SettingsSampler.java`
  - `Python_scripts` ..... zdrojové kódy python skriptů
    - `graph.py`
    - `slowheaders.py`
    - `slowpost.py`
    - `slowpreface.py`
    - `slowread.py`
    - `slowsettings.py`