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

VÝVOJ A TESTOVÁNÍ MODULU PRO HTTP/2 V PROGRAMU APACHE JMETER

DEVELOPMENT AND TESTING OF A MODULE FOR HTTP/2 IN THE APACHE JMETER PROGRAM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Roman Szymutko

AUTHOR

VEDOUCÍ PRÁCE — RNDr. Ing. Pavel Šeda, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Roman Szymutko ***ID:*** 240979

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Vývoj a testování modulu pro HTTP/2 v programu Apache 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 implementujte 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: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: RNDr. Ing. Pavel Šeda, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Práce se zabývá vývojem nových modulů pro software Apache JMeter. Moduly implementují protokol HTTP/2, který není podporován všemi aplikacemi a servery. Mezi tyto aplikace, které nepodporují HTTP/2, patří i Apache JMeter. Vyvinuté moduly umožňují testovat původně chybějící verzi HTTP/2 a slouží k simulaci zátěže na severu, který se stal obětí útoku odepření služby. Konkrétně se jedná o spouštění útoků HTTP/2 flood a Slow Read. V teoretické části této práce jsou popsány některé typy útoků odepření služby. Jsou zde popsány jak klasické útoky, tak i útoky pomalé, jejichž výhodou je malé využití kapacity sítě. Praktická část popisuje podrobněji hlavní komponenty a to dva vytvořené moduly i webový server použitý k testování funkčnosti řešení.

KLÍČOVÁ SLOVA

Apache JMeter, HTTP/2, DoS, Slow DoS, Java

ABSTRACT

This thesis focuses on the development of new modules for the software Apache JMeter. These modules implement an HTTP/2 protocol, which is not yet globally adopted. This means that it is not supported on all applications and servers, including Apache JMeter. These new modules add options for testing the new version of HTTP and are meant for simulating traffic on a server targeted by a Denial of Service attack. These modules allow an execution of an HTTP/2 flood attack and a Slow Read attack. The theoretical chapter of this thesis describes technical details about HTTP, JMeter principles, its alternatives, and also classical DoS and slow DoS attacks. The main advantage of Slow DoS attacks is low usage of network capacity for their function. Lastly in this text, there are also described two main components in depth: the newly developed modules and the web server used for testing the modules' functionality.

KEYWORDS

Apache JMeter, HTTP/2, DoS, Slow DoS, Java

SZYMUTKO, Roman. *Vývoj a testování modulu pro HTTP/2 v programu Apache JMeter*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024, 53 s. Bakalářská práce. Vedoucí práce: RNDr. Ing. Pavel Šeda, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Roman Szymutko
VUT ID autora:	240979
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Vývoj a testování modulu pro HTTP/2 v programu Apache 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 RNDr. Ing. Pavlu Šedovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato práce se zabývá vývojem modulů pro testování webových serverů využívajících protokol HTTP/2. Práce je rozdělena do dvou kapitol. První z nich se zabývá teoretickými poznatky důležitými k řešení této práce. Kapitola je dále rozdělena na tři další části.

První část první kapitoly se zabývá softwarem Apache JMeter, pro který je v této práci vyvíjen modul. Jsou zde popsány jeho důležité součásti, které jsou v řešení použity. Nakonec jsou zde uvažovány alternativy k JMeteru.

Druhá část první kapitoly se zabývá protokolem HTTP. Ta je rozdělena na čtyři další části, které řeší jednotlivé verze protokolu HTTP – od protokolu verze 0 až po verzi 3. Nejdelší z těchto částí je kapitola o HTTP/2, protože tato verze protokolu je vyžadována v zadání. Je zde popsána i motivace a vývoj protokolu verze 2, dále pak přístup ke komunikaci, rozdíl protokolu oproti starým verzím, použité protokoly dalších vrstev nutné k činnosti HTTP/2. Na konci této části jsou nastíněny některé slabiny protokolu HTTP/2, které mohou při zneužití přerůst v odepření služby. Navíc je zde pak uvedena i verze HTTP/3 a její podobnosti a rozdíly s verzí 2.

Odepření služby je popsáno ve třetí části teoretické kapitoly. Jsou zde probrány nejprve klasické útoky, používané původně na starší verzi HTTP protokolu, které ke své činnosti vyžadují velký provoz. Je zde dále nastíněna problematika distribuovaných útoků a využití tzv. botnetů k jejich realizaci. Dále se zde řeší pomalé DoS útoky, které jsou zákeřné tím, že negenerují velký provoz a snadno se tak skrývají. Útoky v této části již uvažují i verzi HTTP/2.

V druhé kapitole jsou napsány poznatky z praktického řešení zadání. Jsou zde podrobněji popsány vyvinuté moduly i testovací web server spuštěný za pomoci softwaru Docker. Vyvinuté moduly jsou třídami v jazyce Java, které dědí vlastnosti z objektů vyvinutých vývojáři Apache JMeteru. U těchto tříd jsou rozepsány implementované metody, jejich vstupy, výstupy a také důvod jejich vzniku a existence.

První modul umožňuje testování komunikace na protokolu HTTP/2. Dále je možné na tomto modulu spouštět záplavový HTTP/2 útok. Druhý modul slouží ke spouštění útoku pomalého čtení.

Testovací server v roli oběti útoku používá ke své činnosti software Nginx. Ten je nakonfigurován jako HTTP/2 server. Pro správnou činnost HTTP/2 musí server podporovat zabezpečenou a šifrovanou verzi HTTPS. Z tohoto důvodu je testovací server vybaven i certifikátem, který je manuálně přidán mezi důvěryhodné.

V poslední částí jsou shrnuty výsledky testování spojení, záplavového útoku i útoku pomalého čtení na webový server spuštěný v kontejneru softwaru Docker.

1 Důležité poznatky

Tato část práce je věnována poznatkům, které jsou považovány za důležité pro řešení práce. Mezi tyto poznatky patří informace o softwaru, pro který byl modul vyvíjen a pak také technické znalosti o protokolu HTTP, jenž slouží k přenosu obsahu webových stránek.

1.1 Apache JMeter

Apache JMeter [1] je software s otevřeným zdrojovým kódem pro testování primárně klient-server aplikací, měření jejich výkonu a chování. První verzi vyvinul Stefano Mazzocchi z Apache Software Foundation [2]. Jeho práce byla dále rozvíjena a byly implementovány další možnosti testování. Verze použitá v této práci má označení 5.6.3. Celý program byl vyvinut v programovacím jazyce Java [3], díky čemuž je možné vytvářet vlastní rozšíření a moduly do tohoto softwaru v tomto jazyce. Pro zjednodušení vytváření vlastních modulů byly společností Apache Foundation připraveny abstraktní třídy a rozhraní, jejichž zdědění, respektive implementace, umožňuje vytváření vlastních tříd s vlastními testovacími procedurami.

1.1.1 Test Plan

Základním prvkem v Apache JMeteru je testovací plán. V něm se do testovací procedury přidají jednotlivé komponenty a moduly. Jedná se o jakýsi základní kámen každého testování. Testovací plán lze uložit jako soubor s koncovkou `.jmx`, což je soubor typu Java Management Extension používaný pro ukládání služeb programovacího jazyku Java [4]. To umožňuje při příštím spuštění programu Apache JMeter načíst stejnou konfiguraci a není potřeba vytvářet ji znovu. V režimu grafického rozhraní jsou jednotlivá vstupní textová pole předvyplněna posledními použitými hodnotami načtenými z konfigurace.

1.1.2 Thread Group

Thread Group je součást testovacího plánu, která simuluje uživatelskou skupinu dotazující se na testovaný server. V základním provedení definuje např. počet cyklů testu, počet vláken, počet uživatelů a zda budou noví uživatelé přibývat s časem. Ve složitějších variantách pro některé typy DDoS útoků, obsahuje tato položka mnohem více nastavitelných polí, kupříkladu životnost vlákna a jeho další časové vlastnosti.

1.1.3 Sampler

Sampler je další důležitou součástí testovacího plánu. Jeho hlavní funkcí je zasílání požadavků vybraného protokolu na daný server. Před každým testem je nutno určit, co a jak bude testováno, např. jaký protokol aplikační vrstvy má být využít, kde se nachází testovaný server apod. Do sampleru jsou hodnoty vkládány skrze GUI (Graphical User Interface – Grafické uživatelské rozhraní), které využívá rozhraní Java Swing a jednotlivé komponenty jsou potomky tohoto rozhraní.

Pro vývoj nového sampleru jsou přímo od Apache Software Foundation připraveny abstraktní třídy, jako např. `AbstractSampler`, který je součástí balíčku `org.apache.jmeter.samplers`. Tento balíček obsahuje i další třídy, které jsou v této práci využity, mimo jiné i třída `SampleResult`. Objekt třídy `SampleResult` je využit jako návratová hodnota výsledku sampleru.

1.1.4 Listener

Listener je poslední důležitou částí testovací procedury, která je v této práci využívána a slouží ke zpracování přijatých dat. Tato data jsou organizovaná do již dříve zmíněného objektu třídy `SampleResult`, neboli výsledku. Do výsledku spadají změřené hodnoty, např. doba testu, zpoždění apod., a pak také samotná data paketů přijatých v rámci testovací procedury. Listener však neumí z přijatých dat sestavit skutečný vzhled webové stránky, protože JMeter nepodporuje grafické zobrazení načtené stránky jako webový prohlížeč [1]. Uživateli pouze zobrazí obsah html dokumentu v textové formě.

1.1.5 Alternativy k Apache JMeter

Apache JMeter není jediným softwarem pro zátěžové testování serverů na trhu. Jednou z alternativ je LoadRunner [5] vyvinutý společností Micro Focus. V tomto softwaru je podporováno mnoho dalších nástrojů třetích stran. Další výhodou je využití na platformách Microsoft Windows i Linux. Nevýhodou jsou vysoké nároky na operační paměť a poměrně vysoká cena [5].

Open-source alternativou je nástroj Gatling [5]. Podporuje testování na protokolu HTTP a také je multiplatformní – byl totiž vyvinut v programovacím jazyce Java. Mezi výhody patří to, že je zdarma a již ve vývhozím stavu poskytuje dostatek užitečných moduly i bez nutnosti vytváření dalších nástrojů. Nevýhodou je však absence uživatelského rozhraní [5].

Locust [5] je také software s otevřeným zdrojovým kódem. Tento nástroj však využívá jazyk Python pro definici testovacího plánu. Mezi jeho výhody patří dobrý

výkon v různých prostředích [5]. Za nevýhodu lze považovat fakt, že k jeho použití je nutno znát alespoň základy jazyku Python.

Nástroj Spirent Avalanche je schopen generovat provoz až 100 Gb/s a umožňuje využití nastupujícího protokolu HTTP/3 [6]. Tomuto protokolu je věnována část 1.2.4. Spirent Avalanche je zpoplatněn.

1.2 HTTP

HTTP je zkratka pro Hyper Text Transfer Protocol. Tento protokol aplikační vrstvy je určen k přenášení hypertextových dokumentů, především ve formátu HTML. V tomto formátu jsou vytvářeny webové stránky. Dále také umožňuje přenos např. multimediálních MPEG nebo WAV souborů, Java appletů apod.

HTTP funguje na principu komunikace klient-server. V jednoduchosti lze princip shrnout takto: Klient, např. stolní počítač, který chce zobrazit stránku uloženou na webovém serveru zasílá požadavek na daný server. Tento server klientovi odpovídá a zasílá mu vyžádaná data. Požadavek se označuje jako `Request` a odpověď jako `Response`. Existuje několik typů HTTP požadavků (viz tabulka 1.1). Těmto typům požadavků se říká metody [7].

Tab. 1.1: HTTP Metody [7].

HTTP Metoda	Popis
`GET`	Žádost o zaslání dat ze serveru ke klientovi
`PUT`	Žádost o úpravu existujícího záznamu
`DELETE`	Žádost o smazání dat ze serveru
`POST`	Žádost o uložení dat klienta na server
`HEAD`	Žádost o zaslání HTTP hlaviček z odpovědi
`OPTIONS`	Žádost o zaslání informací o možnostech komunikace
`TRACE`	Žádost o zaslání informací o cestě
`CONNECT`	Žádost o vytvoření tunelu (např. SSL)
`PATCH`	Obdoba metody PUT pro částečnou úpravu

HTTP využívá ve svých odpovědích kódy, spadající do jedné z pěti základních kategorií. Rozlišovacím znakem, který určuje do které kategorie kód spadá, je první číslice (viz tabulka 1.2). Další dvě cifry nemají speciální pravidlo určující jejich zařazení.

HTTP do verze 2 pro svou činnost používá transportní protokol TCP. Jedná se o spolehlivý protokol – zasílá potvrzovací rámce pro kontrolu a při výpadku je možnost data zaslat znovu – a spojovaný – před komunikací musí být mezi klientem

a serverem uzavřeno spojení (viz obrázek 1.1). Komunikace probíhá na straně serveru standardně na portu 80 pro klasické, nezašifrované HTTP přenášené ve formátu prostého textu. Pro šifrovanou variantu, HTTPS, se využívá port 443.

Tab. 1.2: HTTP Kódy [7].

Kód	Anglické označení	Popis
`1XX`	Informational	Žádost přijata a je v procesu řešení
`2XX`	Success	Žádost vyřízena úspěšně
`3XX`	Redirection	Přesměrování
`4XX`	Client Error	Chyba na straně klienta
`5XX`	Server Error	Chyba na straně serveru

1.2.1 HTTP/0

První verze, která vyšla v roce 1991, se většinou označuje jako **HTTP/0.9**. V této verzi chyběly některé důležité věci jako HTTP hlavičky, jiné než ASCII kódování a obsahovala pouze jednu metodu a to `GET`. Touto metodou dokázala přenést pouze jednoduché HTML dokumenty [8].

1.2.2 HTTP/1

Další verze, **HTTP/1.0**, byla nasazena v mnohem širším měřítku. V této verzi byly přidány hlavičky, číslování verzí, další metody a podpora přenosu multimediálních dokumentů. V této verzi byly vytvářeny formuláře a graficky lépe vypadající stránky. Přesná specifikace této verze však nikdy nevznikla a jednalo se pouze o soubor nejlepších praktik při využívání tohoto protokolu [8].

Do verze 1.0 byly rychle přidávány další funkce, které byly potřebné k rozvoji internetu v 90. letech minulého století. Mezi tyto funkce patří i `keep-alive` vlastnost, která umožňuje zaslání více žádostí a odpovědí skrze jedno TCP spojení. Toto vylepšení bylo označeno jako **HTTP1/0+**, i když se nejedná o žádnou oficiální verzi [8].

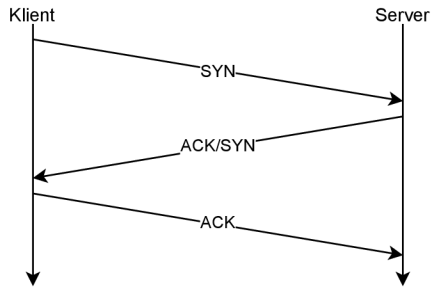
Ve verzi **HTTP/1.1** byla vylepšena optimalizace protokolu. Dále byly opraveny návrhové chyby a byly přidány možnosti vytváření webů s ještě složitějšími prvky. Díky tomu se brzy stala nejpoužívanější verzí a vytlačila předchozí verze HTTP [8].

Protokol HTTP/1 však stále trpěl nedostatky. Vzhledem k růstu rychlostí v sítích přestala být tak důležitým parametrem u přenosu pomocí HTTP propustnost a začala jím být latence. Latence je prodleva v komunikaci, která je u protokolu

HTTP/1 způsobena čekáním na dokončení odpovědi reagující na předcházející žádost [8]. Vyšší latence vychází i z faktu, že TCP spojení je vždy potvrzováno.

Jako jedno z možných řešení problému s latencí vznikl tzv. pipelining, který dovoloval zaslat další žádost před tím, než byla obsloužena předchozí. Tím se ze sériového zpracování požadavků stalo paralelní. Toto řešení však s sebou přineslo další problémy – složitou implementaci, náchylnost k rozbití a málo webových serverů pipelining podporovalo [9].

Nejjednodušším řešením problematiky synchronního sériového zpracování je otevření několika spojení pro jednoho klienta. Tímto jednoduchým způsobem je umožněno paralelní zpracování požadavků z jednoho zdroje. Nejběžněji se používá šesti spojení pro jedno zařízení. Nevýhodou však je, že každé spojení musí splnit náležitosti TCP spojení – tzv. handshake (viz obrázek 1.1) a také potvrzování přijatých dat. Kvůli tomu jsou i v tomto scénáři vynaloženy vyšší nároky na paměť a procesor serveru [9].



Obr. 1.1: Princip navázání TCP komunikace pomocí Three-way handshake.

1.2.3 HTTP/2

Verze HTTP/2 byla poprvé popsána v roce 2015 [10]. Při vývoji této verze byly použity poznatky z experimentálního protokolu SPDY vyvíjeného společností Google. Při vývoji SPDY byla prioritizována rychlost prohlížení webu. Na tento fakt odkazuje i název protokolu – speedy [11]. SPDY zasílá o 40% méně paketů než HTTP a lépe využívá protokol TCP, jelikož použije méně spojení než HTTP a také nečeká na automatické zaslání, ale sám jej umí vyvolat [12]. SPDY umí zaslat více dat skrze jedno TCP spojení díky multiplexování požadavků a odpovědí, které jsou nazývané streamy – datové toky. Dále umožňuje prioritizaci důležitých zdrojů, jakými jsou CSS a JavaScript soubory při načítání stránky [13].

HTTP/2 používá také žádosti a odpovědi, stejně jako předcházející verze. Změnou však jsou rámce (frames), které buď tvoří samotné zprávy a nebo jsou určeny

k vykonání dalších funkcí tohoto protokolu. Rámec je schopen přenášet data o velikosti mezi 2^{14} (16 354) a $2^{24} - 1$ (16 777 215) oktety [13], tedy zhruba 16 kB až 16 MB. Celková velikost rámce je pak velikost přenášených dat v součtu s hlavičkou rámce, která má velikost 9 oktetů [13]. V hlavičce rámce musí být specifikována velikost dat [14]. Hlavička rámce dále obsahuje řídící znaky, které se označují jako flags.

Rámce dále podporují kompresi polí (fields) [14]. Ty fungují už od starších verzí HTTP a jedná se o data v hlavičce organizovaná na jednotlivé řádky označené jmény a nesoucí informace pro klienta, např. datum. Velikost polí se za dobu existence HTTP výrazně zvětšila a přenášela se v nich redundantní data. HTTP/2 proto umí komprimovat velikosti polí a slučuje při tom jednotlivé řádky do bloků [14]. Navíc HTTP/2 přenáší pole, stejně jako celou zprávu, ve formě binárních dat, na rozdíl od předchozích verzí, které byly přenášeny v textové podobě (ASCII) [14]. Místo snadného čtení pro člověka jsou data snadno čitelná pro stroj, což snižuje dobu zpracování dat a tak snižuje latenci.

Rámce tvoří datové toky (streams), které jsou multiplexovány do jednoho spojení. Jednotlivé datové toky mají celočíselný identifikátor o velikosti 31 bitů [14]. Identifikátory jsou udělovány datovým tokům tou koncovou stranou, která přenosem dat skrze daný tok zahájila komunikaci [14].

Datové toky mají několik stavů, ve kterých se mohou nacházet (viz tabulka 1.3). Přechod mezi jednotlivými stavy určují signalizační rámce (viz tabulka 1.4).

Tab. 1.3: HTTP/2 stavy [14].

Stav toku	**Popis**
`idle`	Počáteční zahálející stav
`reserved (local)`	Rezervován pomocí zaslání `PUSH_PROMISE`
`reserved (remote)`	Rezervován na druhé straně komunikace
`open`	Otevřen a může být používán oběma stranami
`half-closed (local)`	Umožňuje zasílat pouze řídící rámce
`half-closed (remote)`	Druhá strana nevysílá a čeká jen řídící rámce
`closed`	Konečný stav po zaslání ukončovacích rámců

Komunikace mezi serverem a klientem vždy začíná tak, že klient zašle HTTP požadavek skrze nový datový tok s novým, zatím nepoužitým identifikátorem datového toku. Odpověď serveru je zaslána stejným datovým tokem, kterým přišla žádost. Každá HTTP/2 zpráva, ať už se jedná o žádost nebo odpověď, se skládá z těchto částí [14]:

1. jeden `HEADERS` rámec nesoucí hlavičku
2. případné `CONTINUATION` rámce nesoucí pokračování, je-li to nutné (0 nebo více)

3. případné `DATA` rámce, je-li to nutné (0 nebo více)
4. případný `HEADERS` rámec nesoucí pole `Trailer` (0 nebo 1)
5. případné `CONTINUATION` rámce nesoucí pokračování, je li to nutné (0 nebo více)

Tab. 1.4: HTTP/2 rámce [14].

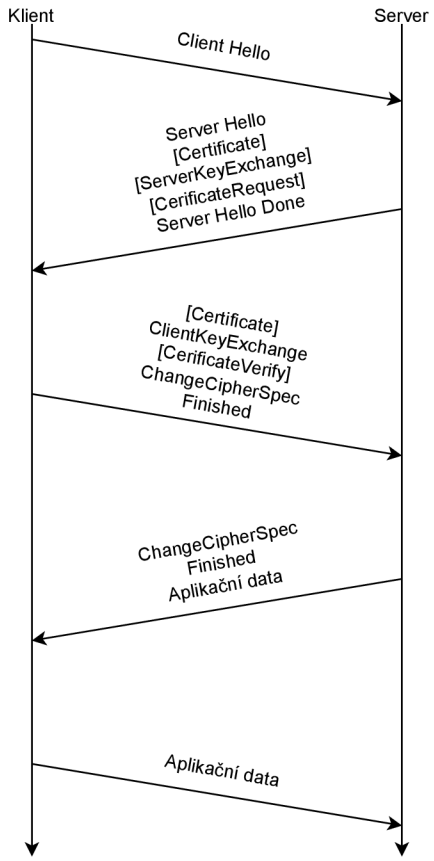
Typ	Popis
`DATA`	Přenos žádostí a odpovědí
`HEADERS`	Otevření toku a přenos hlaviček, fragmentů bloků polí
`PRIORITY`	Určení priority pro plynulejší komunikaci
`RST_STREAM`	Okamžité ukončení toku
`SETTINGS`	Přenos nastavení komunikace
`PUSH_PROMISE`	Oznámení protistraně o úmyslu započetí toku
`PING`	Měření času cesty, kontrola funkčnosti v `idle` stavu
`GOAWAY`	Ukončení spojení, zastavení přijmu nových toků
`WINDOW_UPDATE`	Kontrola množství tekoucích dat
`CONTINUATION`	Dokončení fragmentu bloku polí

Tělo zprávy je přenášeno v `DATA` rámci. Server může odeslat odpověď ještě před tím, než dostane celou žádost, pokud odpověď nezávisí na všech částech žádosti. Poslední rámec z daného datového toku nese značku (flag) `END_STREAM`. Pokud je poslední rámec typu `HEADERS`, ale je potřeba doplnit jej ještě o pokračování `CONTINUATION`, pak nese ukončovací značku právě `HEADERS`, ale zařízení očekává a přijímá i `CONTINUATION` [14].

U HTTP/2 se počítá pouze s využitím zabezpečeného protokolu. Proto druhá verze využívá pro zabezpečení svého přenosu protokol TLS [14], neboli Transport Layer Security Protocol. Ten zaručuje integritu dat a jejich zabezpečení před odposlechem. TLS řeší pomocí svojí podvrstvy TLS Handshake Protocol mimo jiné i výměnu šifrovacích klíčů, které jsou následně použity pro symetrické šifrování přenášeného obsahu [15]. TLS handshake je znázorněn na obrázku 1.2.

HTTP/2 spojení může vyžadovat více hardwarových prostředků, než tomu bylo u starší verze protokolu. To může vyústit v náchylnost vůči DoS (Denial of Service) útokům. Nebezpečí skýtá nedostatečný dohled nad nevyřízenými příchozími rámci – pokud útočník vytvoří velké množství rámců čekajících ve frontě na odeslání, server může být přetížen [14].

Velké množství rámců je možné generovat tak, že v rámci `WINDOW_UPDATE` se bude po malých krocích inkrementovat proměnná, která má za úkol určovat průtok dat. Dále je možné zneužít `PING` rámec, protože na něj je koncové zařízení povinno reagovat. Každý `SETTINGS` rámec musí být potvrzen pomocí `ACK` zprávy, což



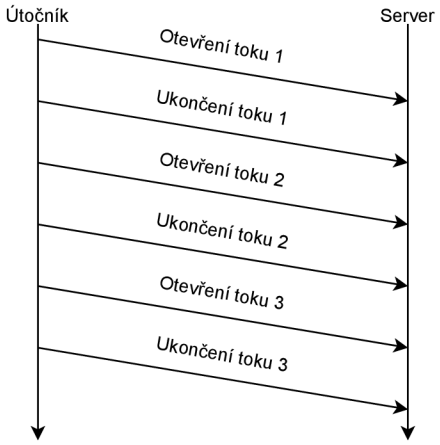
Obr. 1.2: Princip TLS handshake [15].

je možné opět využít pro útok. Pro vyvolání `RST_STREAM` rámců je možné posílat neplatné žádosti, protože server na ně bude reagovat právě tak, že se bude snažit ukončit spojení [14]. Server tak bude zahlcen, aniž by musel přijmout, nebo odeslat jakákoliv data.

Zranitelnost protokolu HTTP, která se nazývá Rapid Reset (označení CVE-2023-44487) [16], umožňuje s malým úsilím vyvolat útok DoS. Útočník může zneužít nízkou režii serveru nad sestavováním a ukončováním datových toků. Z této slabiny pak může vzniknout odepření služby, pokud útočník bude rychle sestavovat a ukončovat datové toky. Schéma tohoto útoku je znázorněno na obr. 1.3.

1.2.4 HTTP/3

HTTP/3 je posledním pokračovatelem rodiny protokolů HTTP. Principiálně funguje stejně jako protokol HTTP/2, tedy podporuje datové toky, jejich multiplexování a řízení jejich toku. Hlavním rozdílem je využití protokolu QUIC místo klasického protokolu TCP, který figuruje v protokolu HTTP/2 [17].



Obr. 1.3: Princip DoS útoku Rapid Reset [16].

QUIC byl vyvinut firmou Google LLC. a je založen na protokolu transportní vrstvy UDP. Ten je ze své podstaty rychlejší, než protokol TCP, protože je bezspojový – nevyužívá handshake k vytvoření spojení – a nespolehlivý – nevyžaduje potvrzování všech zaslaných paketů. QUIC využívá zabezpečení pomocí TLS [18].

1.3 Denial of Service

Denial of Service, v překladu odepření služby, je útok směřovaný na webové servery s úmyslem zahltit server zbytečnými požadavky. Oběť útoku pak není schopna odbavovat požadavky běžných uživatelů, čímž je poškozena zaměřená služba. Mezi typické služby, které se stávají oběťmi útoků patří email, webové stránky, bankovní účty a další [19].

Nejjednodušším způsobem, kterým je možné dosáhnout odepření služby, jsou tzv. flood útoky. Ty fungují tak, že oběť zahltí tak velkým počtem paketů, že je oběť nestihne zpracovávat a pro klasické uživatele se server jeví jako nedostupný.

Existuje podskupina DoS útoků, které se nazývají Slow DoS. Ty jsou specifické tím, že nevyužívají nadměrného provozu v síti. To je dělá špatně detekovatelnými a zároveň i jednodušími, protože k jejich provedení jsou potřebné značně menší útočníkovy zdroje, než v případě flood útoků. Slow DoS útoky zneužívají prodlevy mezi jednotlivými prvky komunikace. Útočník nutí server čekat na další část jeho požadavku co nejdéle je to možné, čímž zaměstnává jeho prostředky a může tak vznikat odepření služby pro běžné uživatele komunikující s daným serverem.

Další podskupinou DoS útoků jsou tzv. DDoS – distribuované útoky odepření služby. To znamená, že útočníkem není pouze jedno zařízení, ale je jím celá skupina,

někdy geograficky náhodně rozložena. Útočník kromě větších prostředků získá vícero zařízeními i horší vystopovatelnost. Často bývají zdrojem těchto útoků tzv. botnety.

Botnet je skupina zařízení – tzv. botů, které jsou řízeny z Command & Control (C&C) Serveru, nad nímž má správu tzv. botmaster [20]. Botnety jsou v dnešní době velkým rizikem. Nová zařízení jsou do botnetu získávána často pomocí phishing útoků a umožňují útočníkovi provádět i jiné ilegální aktivity než DDoS útoky [21]. Botnety jsou často sestavovány bez povšimnutí majitelů zařízení. Uživatel však může zaznamenat pokles výkonu zařízení a poskytovatel internetu podezřelý provoz skrze jeho firewall.

1.3.1 HTTP Flood

Velmi jednoduchým typem útoku je HTTP Flood. Ten používá metody `GET` a `POST`, které ve velkém počtu útočník zasílá na server. Jelikož jsou ale tyto požadavky na síti velmi časté, je odhalení tohoto typu útoku složitější, než by se mohlo zdát [22]. Pokud nepochází všechny pakety pouze z jedné IP adresy, tak lze jen těžko odhalit, zda je o danou stránku takový zájem, nebo jestli se stala terčem útoku. Tento typ útoku je vhodný především distribuovaný mezi větší počet útočících strojů.

1.3.2 SYN Flood

Tento útok zneužívá three-way handshake protokolu TCP (viz obr 1.1). Útočník zasílá žádosti `SYN` obsahující neexistující IP adresu [22]. Handshake není nikdy dokončen a server je nucen spravovat polootevřená TCP spojení. Oběť tedy nasazuje své hardwarové prostředky i na takováto spojení a snaží se je dokončit [22], čímž útočník vyčerpá serverové prostředky a legitimní uživatelé se ke službě nedostanou.

1.3.3 ICMP útoky

V případě **ICMP Flood** se opět jedná o velmi jednoduchý typ útoku. Útočník generuje velké množství ICMP Echo žádostí – ty jsou známé jako ping – a zahlcuje jimi cílový server [23]. Požadavků musí být poměrně velké množství, aby zahltily hardwarové prostředky serveru.

Pomocí protokolu ICMP lze generovat odepření služby i tak, že jsou zasílány ICMP Echo žádosti na broadcastovou adresu v síti oběti. Pakety mají zfalšovanou (spoofed) IP adresu, aby všechny odpovědi na ping z dané sítě mířily na IP adresu oběti a tím zahltily její hardwarové prostředky. Tento typ útoku se jmenuje **Smurf attack** [24]. Název pochází ze jména souboru zdrojového kódu – `smurf.c`.

Dalším druhem útoku s využitím protokolu ICMP je tzv. **Ping of death**. Útočník deformuje odchozí pakety tak, aby byly větší, než povoluje protokol IP. Klasický ICMP ping má velikost 56 bytů a maximální povolená velikost IPv4 paketu je 65 535 bytů. Příliš velké pakety jsou pak fragmentovány a posílány na zařízení oběti, kde se znovu sestavují. Zde pak dochází k problému, že znovu sestavené pakety jsou příliš velké, což vede k přetečení vyrovnávací paměti a tím je způsobeno odepření služby [25].

1.3.4 UDP flood

Při tomto útoku generuje útočník velké množství UDP datagramů obsahujících pakety s neplatnou IP adresou zdroje. Aby byl účinek ještě zvýšen, jsou zaměřovány různé otevřené porty serveru. Výsledkem je to, že se server snaží zasílat ICMP zprávy informující o nedostupnosti falešné IP adresy a tím se vyčerpají jeho prostředky a pro legitimní uživatele se služba jeví jako nedostupná [22].

1.3.5 DNS útoky

Pokud se terčem útoku stane přímo DNS server, může jej při dostatečném provozu vyřadit škodlivý uživatel tak, že zasílá dostatečný počet DNS požadavků, aby vyřadil oběť z provozu pro ostatní uživatele. Tento typ útoku se nazývá **DNS flood** [22].

Další možností, jak zneužít DNS dotaz je tzv. **DNS amplification**. Zde jsou na DNS server posílány dotazy s padělanou zdrojovou IP adresou, tak, aby DNS server odpovídal oběti. Útočník zasílá velké množství dotazů na DNS server a ten pak svými odpověďmi zahltí oběť [26].

1.3.6 Slowloris

Slowloris patří mezi tzv. Slow DoS útoky. V případě útoku Slowloris stačí na úspěšné ochromení serveru pouze jeden osobní počítač [27], což jej dělá velmi snadno implementovatelným. Jeho snadné nasazení je jedním z důvodů jeho účinnosti.

Tvůrcem je Robert „RSnake“ Hansen [27]. Název tohoto útoku odkazuje na jméno zvířete, česky zvaného outloň váhavý [28]. Druhové jméno tohoto živočicha přesně vystihuje fungování tohoto typu útoku.

Základním principem tohoto útoku je zasílání nedokončených HTTP požadavků na server [29]. K útoku je využíváno požadavků typu `GET` [30]. V této metodě pak chybí ukončovací sekvence `\r \n \r \n` [31]. Server drží pro každou žádost časovač, během jehož běhu očekává její dokončení. Když hrozí, že se spojení ukončí, pošle útočník další část žádosti (naplněnou i náhodnými znaky), kvůli čemuž je vynulován časovač a žádost tak nevyprší [29]. Tato komunikace mezi útočníkem a serverem

obsahuje malý objem dat, která nejsou odesílána ihned za sebou. Provoz na síti je tak velmi malý. Útok spočívá především v obsazení spojení, aby jej nemohli využít další uživatelé.

Proti útoku Slowloris se však lze bránit. Nejjednodušším opatřením je zajistit možnost připojení pro více uživatelů pomocí zvýšení hardwarových prostředků serveru [27]. Útočník tak bude vytěžovat pouze svoje spojení, ale připojení dalších uživatelů neovlivní. Aplikace, kterou server provozuje, musí mít správně ošetřeno využití prostředků spojeními s malým datovým objemem.

Další možností obrany je omezit počet spojení, která může navázat jeden uživatel [27]. Kvůli tomu nemůže útočník okupovat vícero spojení a server tak může obsloužit další uživatele. Útočník však může opatření obejít distribucí útoku mezi zařízení.

Logickým řešením je i kontrolovat nejnižší dovolenou rychlost zasílání po sobě jdoucích částí jednoho HTTP požadavku [27]. Tímto bezpečnostním krokem je zajištěno, že útočníkovi budou uzavírána spojení ze strany serveru, takže je nebude přetěžovat a zabraňovat dalším uživatelům v přístupu ke službě.

Následky útoku lze zmírnit i kontrolováním doby připojení každého uživatele [27]. Toto řešení, stejně jako předcházející, limituje útočníkův přístup k prostředkům serveru. Opatření lze částečně obejít distribucí útočících zařízení a opakováním útoků.

1.3.7 Slow POST

Slow Post, někdy taky nazývaný R. U. D. Y. (z anglického R U Dead Yet, tedy v češtině „Jsi už mrtvý?“), je útok, při kterém je škodlivým uživatelem zneužito natavení velikosti zaslaných dat [31]. Útočník nejprve nastaví velikost příchozího těla zprávy na co největší, čímž donutí server čekat na konec této zprávy [30].

Samotná zpráva je pak velmi malá a server očekává její dokončení. Zde se pak útočník chová obdobně, jako u útoku Slowloris a co nejvíce zdržuje zaslání další části požadavku. Rozdíl je však i v tom, že je využita metoda `POST`.

Slow POST útok je možné provést i s využitím protokolu HTTP/2. Útočník nutí oběť čekat na `DATA` rámec. Každý server pak má implementovaný čas, po který je ochoten čekat na slíbený rámec. Útočník zneužívá, obdobně jako i v jiných typech pomalých útoků, právě tuto časovou prodlevu. Pro úspěšné odepření služby je zapotřebí několik spojení [31].

Z podobnosti útoků Slow POST a Slowloris pak vyplývá i to, že obrana proti útoku bude velmi podobná. Server může opět kontrolovat dobu, po kterou se bude požadavek dokončovat. Pokud útočník nedokončí dotaz ve správném časovém intervalu, server s ním rozváže spojení. Pro útok využívající HTTP/2 je pak vhodným

bezpečnostním opatřením kontrola počtu otevřených datových toků pocházejících z jedné zdrojové IP adresy. Opatření lze obejít distribucí útoku.

1.3.8 Slow Read

Tento druh útoku popsal Sergey Shekyan z Qualys Security Labs [30]. Útočník v tomto případě zneužívá toho, že server je ochoten zasílat velké zprávy po malých částech. Ke zneužití se dá použít např. obrázek, který bude kvůli nastavení krátké datové jednotky odesílán klidně i po dobu několika dní [31].

Pomalého čtení lze dosáhnout tak, že se během navázání spojení paketem `SYN` při TCP three-way handshake vnutí serveru velikost okna pouze 28 bytů, místo nejčastějších 1448 bytů [30]. Jelikož je navázání spojení validní a na první pohled v pořádku, je poměrně složité, stejně jako u ostatních Slow DoS útoků, jej odhalit. Odmítnutí služby ale většinou nastane až při více navázaných spojeních [30], což je možné pak opět řešit tak, že jednomu uživateli bude vyhrazen omezený počet spojení se serverem.

Útok je proveditelný i s využitím protokolu HTTP/2. Zde je zneužito nastavení znaku `SETTING_INITIAL_WINDOW_SIZE` na hodnotu 0, čímž se serveru oznámí, že je uživatel momentálně zaneprázdněn. Server tak bude čekat na zvětšení velikosti okna pomocí `WINDOWS_UPDATE`, aby mohl zaslat vyžádaná data. Změna velikosti však není útočníkem zaslána [31]. Implementace útoku pomalého čtení je popsána v části 2.1.2.

1.3.9 Slow HEADERS

Slow HEADERS využívá protokolu HTTP/2 a je přímým nástupcem Slowloris útoku používaného proti HTTP/1 web serverům [32]. V tomto typu útoku je zneužito rámců `HEADERS` a `CONTINUATION`. Útočník serveru sdělí, že nedokončil hlavičku a bude tak zasílat `CONTINUATION` rámec s pokračováním. Vhodnými metodami jsou `GET` a `POST`. Pro každou z těchto variant jsou nastaveny jiné příznaky – u metody `GET` je zasláno `END_HEADERS: 0`, `END_STREAM: 1` a u metody `POST` `END_HEADERS: 0`, `END_STREAM: 1` [31].

V případě použití metody `GET` je zneužito funkcionality HTTP/2, která způsobí, že i když se signalizuje ukončení spojení (`END_STREAM: 1`), tak server před ukončením datového toku očekává minimálně ještě alespoň jeden `CONTINUATION` rámec (popsáno v části 1.2.3). Čekání na pokračování je ošetřeno časovačem, na jehož nastavenou hodnotu tento útok cílí [31].

Aby útočníkovi nebylo serverem ukončeno spojení, stejně jako je popsáno v části 1.3.6, posílá škodlivý uživatel pokračovací rámce s náhodnými daty [32], čímž opět resetuje odečet časového limitu.

1.3.10 H_2DoS

Tento typ útoku nespadá do kategorie Slow DoS. Zde je zneužito multiplexování datových toků a nastavení příliš malého okna [33]. Pro jeho činnost stačí navázat jediné TCP spojení. Pro zesílení účinku útoku může útočník otevřít více než jedno spojení a vyčerpat tak prostředky oběti ještě rychleji.

Útok lze shrnout v několika bodech [33]:

1. Navázání spojení TCP a TLS.
2. Nastavení maximálního počtu datových toků útočníkem na co největší číslo v toku 0.
3. Nastavení velikosti okna na co nejmenší útočníkem v toku 0.
4. Zaslání `GET` žádosti s hlavičkou rozdělenou na `HEADERS` a `CONTINUATION` rámce útočníkem v datovém toku 1.
5. Zasílání dalších žádostí po dalších datových tocích číslovaných lichými čísly.
6. Zaslání `WINDOWS_UPDATE` s malou inkrementací – zabránění ukončení spojení.

Jelikož útok spoléhá na odepření služby kvůli přehlcení serveru požadavky, potřebuje útočník také adekvátní hardwarové prostředky. Navíc otevření několika toků bude generovat na síti velký počet procházejících paketů, což je možné detekovat a následně lokalizovat útočníka.

2 Výsledky práce

Tato práce obsahuje dvě základní části řešení. První z nich je samotný vývoj modulů – samplerů – pro software Apache JMeter. Druhá část řeší jeho testování – práce se zde zabývá vytvářením vhodného webového serveru v roli oběti a poté spouštěním jednotlivých útoků na tento cíl.

Pro vývoj modulů bylo použito prostředí IntelliJ Idea od společnosti JetBrains. Sestavení projektu bylo zajištěno nástrojem Gradle, který je možné přidat do vývojového prostředí IntelliJ jako plug-in. Po jeho přidání se stává prostředí responzivním a skrze interaktivní tabulky informuje vývojáře o případných chybách, dědičnosti a vlastnostech použitých tříd.

V této práci je využit projekt, který byl součástí diplomové práce [34]. Všechny moduly však v tomto projektu využívaly protokol HTTP/1.

V rámci práce vznikly dva samplery. První z nich byl vytvořen pro testování HTTP/2 spojení, transportu dat skrze tento aplikační protokol a také testování HTTP/2 flood útoku. V druhém sampleru byl implementován útok typu Slow Read.

2.1 Samplery

Každý sampler potřebuje ke své funkci alespoň dvě třídy – jednou třídou bylo vytvořeno grafické rozhraní a v rámci druhé třídy byl implementován vnitřní algoritmus sampleru.

Předávání nastavených hodnot mezi uživatelským rozhraním a vnitřní částí sampleru bylo realizováno skrze tzv. properties – vlastnosti. Ty mají dán jednotný formát – `public static final String NÁZEV = "Identifikátor"`. V grafickém rozhraní pak je vstupní pole provázáno s danou vlastností. Pokud je potřeba k vlastnosti přistoupit, volá se metoda, která byla zděděna ze tříd vyvinutých v Apache Software Foundation. Tato metoda se nazývá `getPropertyAsString()`, případně `getPropertyAsBoolean()`. Jejím vstupním parametrem je název vlastnosti. Metoda pak vrací nastavenou hodnotu vlastnosti jako datový typ `String`, respektive `Boolean`. Jelikož sampler tuto metodu dědí, tak se volá přes klíčové slovo `this`. Všechny vlastnosti jsou definovány v abstraktní třídě `HTTPSamplerBase`.

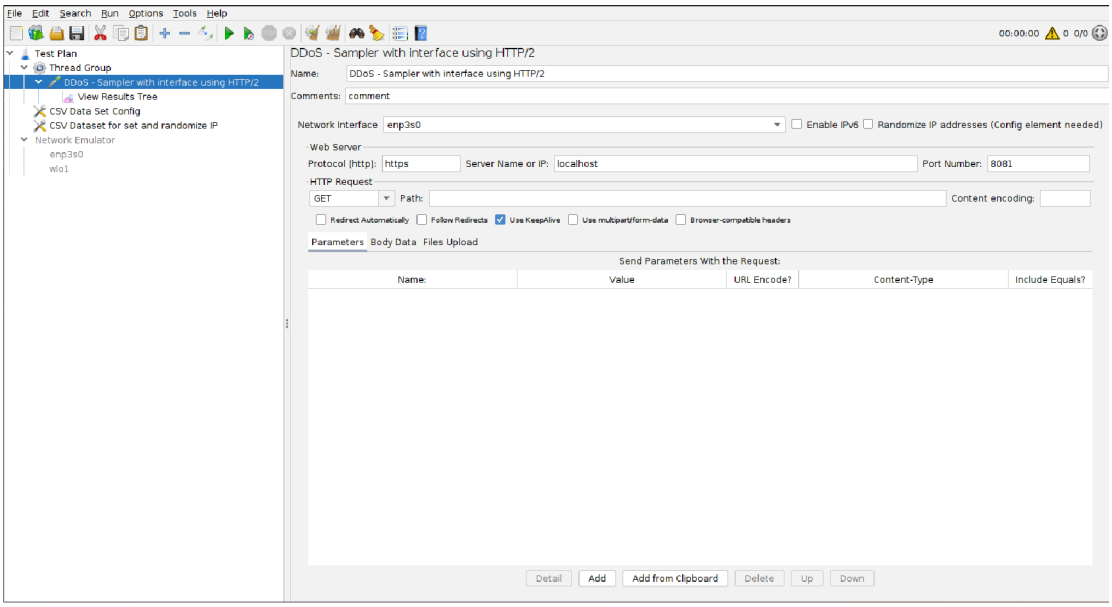
2.1.1 Testovací a záplavový modul

Úkolem tohoto sampleru bylo ověřit funkčnost protokolu HTTP/2 na platformě JMeter a možnost testování tohoto protokolu v jazyce Java. Tento sampler umí vytvářet HTTP požadavky a komunikovat pomocí nich se serverem, který je nakonfigurován na stejnou verzi tohoto protokolu.

InterfaceHttp2SamplerGui.java

Základní grafické rozhraní prvotního modulu tvoří `InterfaceHttp2SamplerGui` dědící z připravené abstraktní třídy `AbstractSamplerGui`. V ní bylo využito předdefinované třídy `UrlConfigGui`, která obsahuje všechny důležité položky, které v sampleru musí být pro správnou činnost definovány. Ve třídě, ve které bylo definováno grafické rozhraní, bylo určeno i zobrazené jméno modulu. Název modulu v menu Apache JMeter je „DDoS – Sampler with interface using HTTP/2“. GUI vychází z rozhraní „DDoS – Sampler with interface“ [34]. Grafické rozhraní je zachyceno na obr. 2.1.

Jelikož nebyl v rámci projektu k dispozici žádný vhodný sampler, který by umožňoval generovat provoz na protokolu HTTP/2, musela vzniknout úplně nová třída `Http2Sampler`, kam jsou předány hodnoty z grafického rozhraní a jsou na jejich základě vytvořeny požadavky.



Obr. 2.1: Grafické rozhraní testovacího modulu.

Http2Sampler.java

Třída vyvinutého modulu dědí z abstraktní třídy `AbstractSampler`. V této třídě je využita dědičnost dalších abstraktních tříd. Tato struktura pak poskytuje metodu `sample()`, která byla pomocí klíčového slova `@override` přetížena. V této přetížené metodě pak byla definována celková činnost celého vytvořeného modulu – jedná se tedy o jakousi hlavní metodu této třídy.

Objekt třídy `SampleResult` je návratovou hodnotou metody `sample()`, jak bylo zmíněno už v části 1.1.3. Tato třída obsahuje další důležité metody, které se skrze běh sampleru provolávají nad návratovým objektem. Návratový objekt je pojmenován `result`.

Během průchodu metodou je nejprve výsledku nastaveno označení pomocí metody `result.setSampleLabel()` a pak je zapsán čas spuštění sampleru pomocí `result.sampleStart()`. Tím je sejmut časový otisk pro měření doby vzorkování. Dále je vytvářen samotný HTTP požadavek.

K vytvoření klienta slouží třída `HttpClient`. Tato třída umožňuje používat verzi HTTP/2, pokud jí podporuje server. Verze byla specifikována ve vytvořené metodě `buildClient()`, která dále spravuje i politiku přesměrování. Pokud není možné použít verzi HTTP/2, protože ji server nepodporuje, je použito HTTP/1 a komunikace proběhne skrze verzi 1.

Požadavek je vytvářen pomocí objektu třídy `HttpRequest`. V něm je definován `Builder`, který spojuje URI dotazovaného serveru, metodu požadavku a tělo požadavku. Tělo požadavku vytváří objekt `BodyPublishers`, který jej sestavuje z datového typu `String` zadaného skrze GUI.

Součástí třídy `HttpRequest.Builder` jsou některé HTTP metody, jako např. `GET`. Jelikož ale nejsou v této třídě obsaženy úplně všechny existující HTTP metody, je zde možnost využít metodu s názvem `method()`. Ta má dva vstupní parametry: Typ metody zadaný v datovém formátu `String` a tělo zadané ve formátu `HttpRequest.BodyPublishers`. Takto lze vytvořit všechny existující metody, které HTTP podporuje. `BodyPublishers` umožňuje pomocí metody `ofString()` naplnit tělo zprávy ze zadaného textového řetězce ve formátu `String`.

Výsledkem je sestavená HTTP žádost, která je odeslána skrze dříve vytvořeného HTTP klienta. K tomu slouží metoda `send()`, která má dva parametry a to sestavenou žádost a také zpracování těla přijaté zprávy typu `HttpResponse.BodyHandlers`. Návratová hodnota je typu `HttpResponse` s generickým parametrem <`String`>.

Data z těla odpovědi jsou pak předána do návratové hodnoty `result` pomocí tzv. setterů. Nakonec je sejmut časový otisk, čímž je ukončena doba běhu sampleru a je nastaveno úspěšné ukončení běhu.

Některé volané metody mohou vyvolat výjimkou. V metodě `send()` může být vyvolána výjimka při přerušení nebo při chybě odesílání a přijímání. Z tohoto důvodu byla umístěna kritická část kódu do try-catch bloku. Text výjimky je předán do výsledku, aby bylo možné jej zkontrolovat v listeneru. Výjimku totiž může vyvolat i špatně zadaná URI, která má zadán špatný špatný formát. Zachycená chyba je zaznamenána do logu pomocí objektu třídy `Logger`.

V grafickém rozhraní existují mimo jiné i zatrhávací políčka (checkbox) `Follow Redirect` a `Redirect Automatically`. Tyto dvě položky se navzájem vylučují. Po-

kud je zaškrtnuto Redirect Automatically, je odpověď s kódem `3XX` ošetřena automaticky a uživatel nic nepozná. Pokud je však zaškrtnuté políčko `Follow Redirect`, musí sampler vytvořit výsledek jak pro odpověď s informací o přesměrování, tak i pro odpověď přesměrovaného dotazu.

Zde bylo potřeba vytvořit druhý `result`, který je přidán do konečného výsledku pomocí metody `addSubResult()`. K tomu slouží metoda `subSample()`, která vrací částečný výsledek. Ke své činnosti potřebuje objekt klienta, ze kterého byla předešlá žádost poslána a také odkaz na adresu, kde se požadovaná služba skutečně nachází.

Ze serveru je spolu s kódem `3XX` zaslána i adresa k přesměrování. Adresa je však v odpovědi obklopena HTML značkami. K nalezení identifikátoru ve zprávě je využit regulární výraz, neboli regex. Regex, z anglického Regular Expression, je specifické formátování řetězce pro hledání konkrétních shluků znaků. Tyto výrazy jsou aplikovány v metodě `findUriIn3xx`, jejímž vstupním parametrem je text zprávy v datovém formátu `String` a návratovou hodnotou je objekt třídy `URI`.

V této metodě jsou použity třídy `Pattern` a `Match`, které řeší celou logiku hledání. Stačí zadat regulární výraz a data, ve kterých má být skupina znaků, v tomto případě webová adresa, nalezena. Z důvodů optimalizace je výraz definován jako globální konstanta celé třídy – řetězec se nemusí kompilovat při každém hledávání znovu.

Na vytváření URI byla do sampleru přidána metoda `buildUri()`. Ta bere jednotlivé položky zadané v GUI a skládá je do jednoto textového řetězce. Kvůli šetření paměti byl využit objekt třídy `StringBuilder`. Při spojování řetězců pomocí znaménka „+“ je Java nešetrná k paměti a při každém zřetězení textu touto metodou se alokuje nová paměť, protože se generuje nový objekt typu `String`. Pomocí objektu `StringBuilder` se spojí protokol (https), adresa, cesta a port. Tento objekt je po sestavení řetězce konvertován do datového typu `URI` pomocí konstruktoru.

Metoda `translateCodeToText()` slouží k překladu číselných kódů serverových odpovědí na textové vyjádření. Toho se využívá při nastavení zprávy do výsledku. Celá metoda obsahuje pouze `switch`, který na základě vstupní celočíselné hodnoty přiřadí výstupu textový řetězec ve formátu `String`. Metoda obsahuje všechny existující kódy [40].

2.1.2 Modul pomalého čtení

Tento modul vznikl v pořadí jako druhý a umožňuje spouštět pomalý DoS útok, který se nazývá Slow Read, neboli pomalé čtení. Princip tohoto útoku je blíže popsán v části 1.3.8.

HTTP klient, který byl použit pro vývoj prvotního – záplavového – modulu v této práci, neumožňuje manipulaci s HTTP/2 rámci. Celé TCP spojení, dohodnutí verze a ustanovení zabezpečeného kanálu je totiž řešeno pouze uvnitř klienta

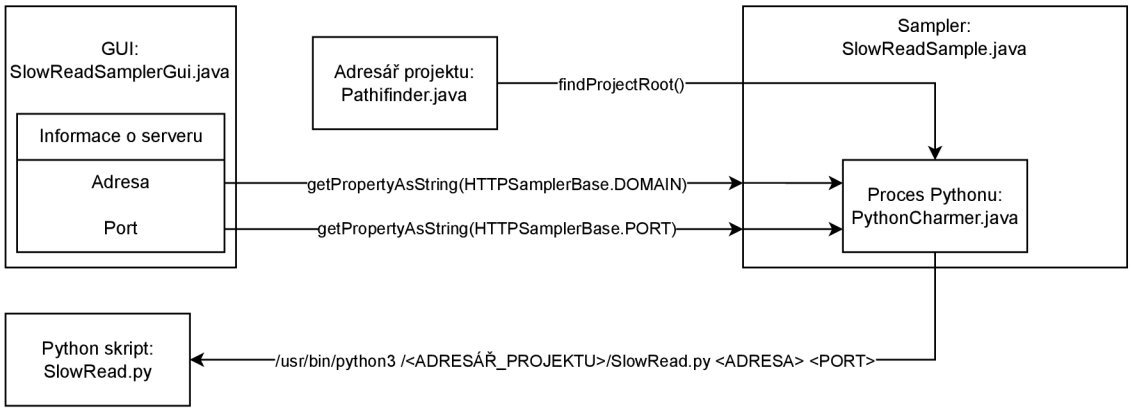
a při implementaci není možné dostat se skrze úrovně abstrakce k těmto vnitřním procesům.

Při hledání řešení připadalo v úvahu využití HTTP klienta z projektu Jetty od společnosti Webtide spadající pod Eclipse Foundation [35]. Jelikož je však projekt, do kterého byl modul vyvíjen, konstruován v Gradle, měl by se použít plugin jménem Gretty [36].

Problémem řešení Gretty/Jetty je to, že se jedná primárně o nástroje určené k vytváření serverů. Poskytování klientských tříd je až druhotné a jedná se o malý zlomek celého projektu.

Při hledání dalších možností byl objeven projekt [37], který se odkazuje na zdroj [32] již citovaný v rámci teoretické části práce. V rámci tohoto projektu autor implementuje čtyři typy pomalých útoků ve skriptovacím jazyce Python. Díky tomuto skriptu bylo zjištěno, že s pomocí jazyka Python je problém řešitelný. Projekt, jehož součástí je tato práce, navíc už obsahoval některé útoky implementované pomocí Python skriptů.

Princip volání útoku v sampleru pro pomalé čtení je znázorněn na obr. 2.2.



Obr. 2.2: Schéma sampleru pomalého čtení.

SlowRead.py

Tento skript se skládá ze tří hlavních částí. Tou první je vytvoření HTTP/2 spojení. Práci s rámci umožňuje knihovna hyperframe [38]. Na této knihovně pak staví knihovna h2, která definuje metody pro navázání potřebného spojení se serverem. Obě knihovny lze nainstalovat pomocí nástroje `pip`.

```
$ pip install h2
$ pip install hyperframe
```

Pro instalaci byl vytvořen skript, který nainstaluje Python interpret i s potřebnými balíčky v jednom kroku. Tento skript se jmenuje `python_install.sh`. Jeho funkčnost byla otestována na virtuálním stroji se systémem Ubuntu 22.04 LTS.

Jelikož je HTTP/2 určen pro komunikaci v šifrovaném režimu, musí být vytvořen SSL kontext. Kontext jsou informace o nastavení zabezpečení přenášených dat formátované do objektu. V případě tohoto projektu je však kontext zanechán ve výchozím (default) nastavení. Kontrola certifikátu dotazovaného serveru je vyřazena. To umožní komunikovat i se serverem, který má pouze certifikát podepsaný sám sebou (self-signed), což je případ testovacího serveru. Data ze zdroje bez autentizace totiž nepředstavují žádné riziko. Z principu útoku totiž žádná data přijata nebudou.

K vytvoření kontextu slouží metoda `get_ssl_context_h2()`. V ní je navíc přímo definováno, že použitým protokolem je pouze HTTP/2.

Pro vytvoření TCP spojení slouží metoda `create_tcp_connection()`. Ta vytváří ze zadané IP adresy a portu soket, na kterém bude vytvořený klient komunikovat se serverem.

V metodě `negotiate_tls_ssl()` se vyjednávají protokoly mezi klientem a serverem pro danou komunikaci. Vytváří se zde TLS spojení (viz obr. 1.2) pro již ustanovené TCP spojení. Dále se zde vyjednává verze – HTTP/2. Pro vyjednání použité verze se používá protokol ALPN. Kdyby preferovaný protokol nebyl k dispozici, je použit protokol NPN. Pokud se ani to nepodaří, je vyvolána chyba. Bez protokolu HTTP/2 totiž není schopen modul fungovat.

Druhou částí je samotný útok, který je proveden podle části 1.3.8. Nejprve klient ustanoví spojení pomocí tříd, které byly popsány výše. Na této komunikaci je pak použit protokol aplikační vrstvy HTTP/2. Podle doporučení [14] se musí nejprve zaslat tzv. „Preface“. To je dohodnutý sled 24 bytů, který má v čitelném formátu tvar `"PRI *HTTP/2.0\r\n\r\nSM\r\n\r\n"`.

Po zaslání této předmluvy je zaslán rámec `SETTINGS`. Tento rámec je zneužit v rámci útoku. Pomocí knihovny `hyperframe` je nastavena velikost okna na 0. Poté je vytvořena žádost pomocí metody `GET`. Nakonec je nastaven signál pro ukončení spojení.

Po poslání všech těchto dat klient čeká na odpověď. To je pomyslná třetí část kódu. Jelikož v jazyce Python neexistuje smyčka `do-while`, je zde udělaná nekonečná smyčka `while` a ukončení smyčky je testováno v podmínce `if`. V této smyčce skript naslouchá a čeká na data zaslaná ze serveru. Jelikož ale nemá server jak data zaslat

skrze okno o velikosti 0 bytů, je pouze obsazeno spojení. Útok je ukončen ze strany serveru při rozvázání spojení.

Data, kterými server signalizuje své nastavení si lze zobrazit. Komunikace je sice bajtově orientovaná, nikoliv textově, ale knihovna `h2` umožňuje přeložit tento datový tok na text. Tento text je nakonec vypsán do standardního výstupu skriptu.

PythonCharmer.java

Tato třída slouží k obsluze skriptu. Skript samotný je možno spouštět z terminálu – cíl útoku je zadán jako argumenty a výsledek je vypsán do standardního výstupu. Jelikož je však tato práce součástí velkého projektu pro JMeter, musí být vyřešeno rozhraní mezi skriptem v Pythonu a samplerem JMeteru, který je napsán v jazyce Java.

Třída obsahuje dvě metody. Jedna útok volá a ta druhá ruší proces útoku. Metoda, která útok volá, se jmenuje `callPythonAttack()`. Jejími parametry jsou cesta k Python skriptu, IP adresa cíle a port, na kterém běží zaměřená služba oběti.

Python skript útoku je spouštěn jako proces za pomoci objektů tříd `Process` a `ProcessBuilder`. V konstruktoru třídy `ProcessBuilder` se zadá cesta k Python interpretru verze 3, cesta ke skriptu a jeho parametry – IP adresa a port oběti. Dále je nastaveno přesměrování chybového výstupu do standardního výstupu skriptu.

Proces je spouštěn pomocí volání metody `processBuilder.start()`. Instance procesu je předána do vlastnosti třídy `PythonCharmer` typu `Process`. Proces musí být vlastností celé třídy, aby bylo možné volat metodu na jeho zastavení.

Standardní výstup je přesměrován do bufferu. K tomuto účelu byl využit objekt třídy `BufferedReader`. Vše, co by skript vypsal do terminálu je zapsáno do této proměnné. Chybové hlášky jsou spolu s daty nasměrovány do jednoho výstupu. Data z bufferu jsou slučována pomocí objektu třídy `StringBuilder` do textu, který tato metoda vrací.

Aby bylo možné proces předčasně ukončit, je zde implementována metoda s názvem `destroyProcess()`. Uvnitř ní je volána metoda `destroy()` nad objektem právě běžícího procesem.

Pathfinder.java

Třída Pathfinder obsahuje jedinou statickou metodu `findProjectRoot()`. Ta slouží k nalezení absolutní cesty ke kořenovému adresáři celého projektu. Znalost umístění projektu je důležitá při hledání cesty k Python skriptu. Absolutní cesta adresáře, ve kterém běží proces celého JMeteru, je zjištěna pomocí systémového volání. K tomu byla použita metoda `System.getProperty("user.dir")`. Bylo zjištěno, že při spuštění JMeteru je aktuální adresář zanořen do `./jmeter-main/jmeter/bin`.

Zjištěná cesta je rozdělena pomocí metody `split("/")`. Takto vznikne seznam adresářů, vedoucí k aktuálnímu umístění. Jelikož je potřeba dostat se o dva adresáře výše, jsou poslední dvě položky ze seznamu odstraněny. Cesta je následně ve smyčce `for` sestavena zpět do textové podoby (formát `String`) za pomocí objektu třídy `StringBuilder`. Mezi jednotlivé adresáře je vloženo lomítko (`"/"`). Metoda vrací nalezenou absolutní cestu ke kořenovému adresáři projektu ve formátu `String`.

Tato zjištěná cesta je pak přidána k relativní cestě k Python skriptu v rámci projektu. Celá cesta je předána pomocí parametru do metody `callPythonAttack()`, která je součástí třídy `PythonCharmer`, viz 2.1.2. Metoda volání útoku je použita v sampleru 2.1.2.

SlowReadSampler.java

Třída sampleru dědí z abstraktní třídy `AbstractSampler`, stejně jako tomu bylo u modulu vyvinutého pro testování druhé verze protokolu. Tento sampler ale navíc ještě implementuje rozhraní `Interruptible`. Toto rozhraní umožňuje přerušovat běh jednoho vzorku. To je v tomto případě důležité, jelikož jeden vzorek běží rámcově několik minut.

Rozhraní `Interruptible` má deklarovánu metodu `interupt()`. Ta je v tomto sampleru přetížena pomocí klíčového slova `@Override` a implementuje možnost přerušení běžícího vzorku. V této metodě je volána metoda `destroyProcess()` ze třídy ovladače Python skriptu `PythonCharmer`.

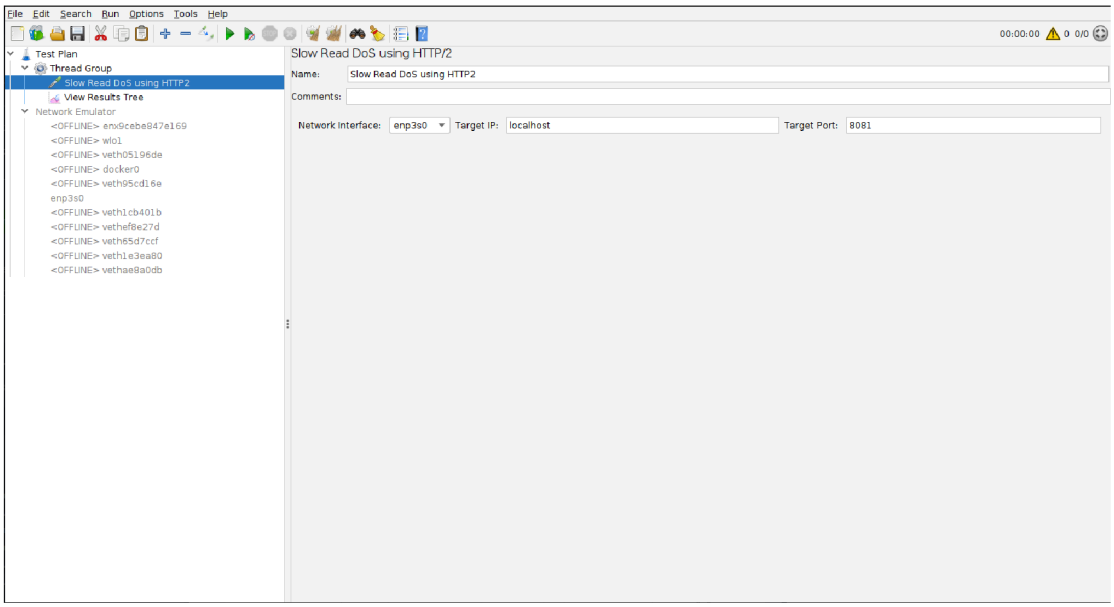
Aby bylo možné přerušit kontext daného procesu, je ve třídě `SlowReadSampler` vlastnost typu `PythonCharmer`. Díky tomu se v rámci tohoto objektu skript spustí a následně je možné ho voláním metody přerušit skrze stejný objekt – metoda pro přerušení `interrupt()` nemá vstupní parametr, takže je definice procesu jako vlastnosti celé třídy nejlepším řešením. Informace o přerušení je zapsána do logu.

Přerušení může uživatel vyvolat v grafickém rozhraní pomocí kliknutí na červený křížek v horní části panelu (viz obr. 2.3 – lišta nahoře). Vše důležité k provolání přerušovací metody z rozhraní `Interruptible` je zajištěno už v samotném JMeteru.

S možností přerušení je pak třeba počítat i při vytváření výsledku sampleru. K tomu slouží metoda `initResult()`. Ta vrací výchozí nastavení výsledku. Toto nastavení nese informace o tom, že proces nebyl korektně ukončen – zpráva „`Not finished`“.

Stejně jako tomu bylo u sampleru vyvinutého pro generování záplavy, je činnost sampleru vykonána v metodě `sample()`. Opět je jejím návratovým datovým typem `SampleResult`.

Spouštění samotného procesu je uskutečněno v bloku `try-catch-finally`. Blok `try` zkouší spustit skript útoku se všemi parametry. Blok `catch` zachycuje případnou



Obr. 2.3: Grafické rozhraní modulu pomalého čtení.

vyvolanou výjimku – spouštění procesu samotné může být chybné, dále se uvnitř spouštěcí metody volá `BufferedReader`, který opět může vyvolat při chybě výjimku. Všechny výjimky jsou typu `IOException`. Pokud nastane výjimka, je nastaven kód výsledku na 500 a do zprávy se přidá text popisující výjimku – kde nastala a proč. Výjimka je navíc zapsána do logu jako error. Blok `finally` pak slouží ke ukončení procesu v případě výjimky.

Bylo uvažováno řešení, které by využívalo `try-with-resources` blok. Ten funguje tak, že se za klíčové slovo `try` napíší kulaté závorky a do nich se napíše inicializace objektu, který implementuje rozhraní `AutoCloseable`. Toto rozhraní má v sobě metodu `close()`, která by měla řešit logiku zrušení procesu.

Zde však nastal problém. Do tohoto bloku lze napsat pouze lokální proměnnou, nikoliv vlastnost celé třídy. Jak už bylo dříve zmíněno, objekt třídy volající útok však musí být globální v rámci třídy, aby bylo možné nad tímto objektem volat přerušení. Implementace rozhraní `AutoCloseable` v rámci třídy `PythonCharmer` by tak nic neřešila. Metoda útoku navíc vrací datový typ `String`, který není `AutoCloseable`.

Další možností bylo umístění `try-with-resources` o stupeň níže, tedy do třídy `PythonCharmer`. Zde však nastává jiný problém – třídy `Process` a `ProcessBuilder` nejsou součástí rozhraní `Autocloseable`. Za předpokladu, že by součástí byly, tak by i nadále přetrvával problém s voláním globální proměnné třídy – `Process` musí být opět vlastností celé třídy, aby bylo možné jej ukončit.

V poslední části metody `sample` je provedeno naplnění objektu návratové hodnoty typu `SampleResult`. Zde je zkontrolován textový řetězec, který vrátila metoda

volání útoku `callPythonAttack`. Pokud je řetězec prázdný, je jasné, že nebyl proces dokončen. Proto v návratové hodnotě zůstanou výchozí data zapsaná metodou `initResult()` oznamující nedokončení procesu.

Pokud jsou navrácena nějaká data, je potřeba zkontrolovat, co se v nich nachází. Pokud navrácený řetězec obsahuje slova „`Traceback`“ nebo „`invalid`“, je jasné, že nastala chyba. Chybová hláška v Pythonu většinou začíná právě slovem „`Traceback`“. Pokud bylo zadáno něco špatně, například parametry udávající cíl útoku, je v chybové hlášce obsaženo slovo „`invalid`“. V případě, že nastal jeden ze dvou scénářů, je nastaven kód výsledku na hodnotu 500 a zpráva na „Error occurred“. V těle výsledku je pak možné si chybové hlášky přečíst. Výstup Python skriptu je totiž vždy přesměrován do výsledku, nehledě na to, zda byl úspěšný či nikoliv. Případná chyba je navíc zapsána do logu.

V případě úspěchu je kód nastaven na 200 a zpráva výsledku je nastaven na hlášku „OK“. Dále je zde nastavena proměnná `isOK` typu `boolean` na hodnotu `true`. Díky tomu bude při zobrazení výsledku viditelná zelená fajfka, nikoliv červený křížek, který je viditelný při neúspěchu či přerušení.

SlowReadSamplerGui.java

Grafické rozhraní vyvinutého modulu dědí z abstraktní třídy `AbstractSamplerGui`, stejně jako tomu bylo u modulu, který byl vyvinut v rámci testování HTTP/2. Jelikož je ale modul pro pomalé čtení funkčně značně odlišný od předchozího vyvinutého, byly zde učiněny změny i v rámci uživatelského grafického rozhraní.

U grafického rozhraní došlo ke značnému zjednodušení oproti testovacímu – záplavovému HTTP/2 modulu. To vyplývá ze skutečnosti, že pro vytvoření útoku Slow Read je potřeba pouze znát cílovou adresu a port služby. Vzhledem k tomuto faktu byly namísto objektu typu `UrlConfigGui` využity dvě textová pole typu `JTextField` – jedno pro IP adresu a druhé pro číslo portu.

Obr. 2.4: Vstupní pole modulu pro pomalé čtení.

2.2 Nginx

K testování slouží Nginx webserver, který má ve své konfiguraci povolenou pouze verzi HTTP/2. Jelikož využitý HTTP klient v sampleru umí komunikovat pomocí HTTP/2 i HTTP/1, jak bylo zmíněno v části 2.1.1, bylo potřeba ověřit, že sampler bude skutečně komunikovat pomocí HTTP/2. Nejjednodušším řešením by samozřejmě bylo použití nástroje Wireshark, ve kterém by se dala najít hlavička a v ní použitá verze, ale to nebylo možné provést, jelikož je protokol HTTP/2 jednak bitově orientovaný a navíc ještě šifrovaný na transportní vrstvě a není tak možné zobrazit si obsah jednotlivých TCP segmentů.

V konfiguraci Nginx web serveru je pevně nastaveno, že musí být použita pouze verze HTTP/2. Ověření verze je ještě uskutečněno pomocí metody `version()` volané nad objektem odpovědi serveru v sampleru. Výsledek této metody je vypsán do terminálového okna, ve kterém je spuštěn JMeter a také do logu.

Nginx je spouštěn jako kontejner ve službě Docker. Díky tomu jej lze spouštět příkazem `sudo docker start nginx`. Pro zastavení služby pak funguje příkaz `sudo docker stop nginx`. Obraz použitý k vytvoření kontejneru je z [39]. Tento Docker kontejner využívá pro činnost webserveru adresu `https://localhost:8081`.

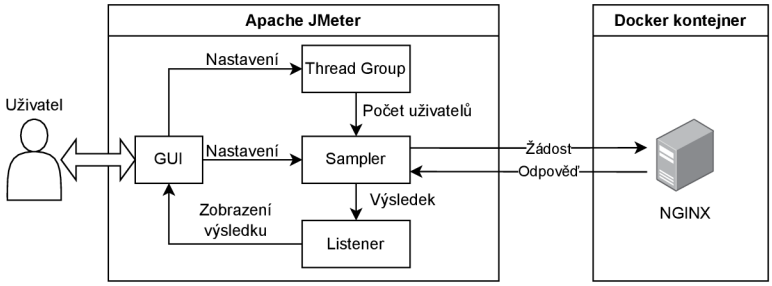
U tohoto testovacího prvku nastal problém při spouštění prvotního modulu `Http2Sampler`. Použitý HTTP klient považuje HTTPS bez důvěryhodného certifikátu za chybu a tak zde byla vyvolána výjimka, kvůli které nebyl program funkční. Aby se tomu předešlo, musel být vytvořen tzv. self-signed certifikát, neboli certifikát podepsaný sám sebou. Jenže takovýto certifikát stále nemá žádnou autoritu, a proto není důvěryhodný a výjimka přetrvává. Toto bylo vyřešeno tak, že byl podepsaný certifikát přidán do Java TrustStore. TrustStore je v podstatě databáze, která obsahuje důvěryhodné certifikáty a je zde nahlíženo při každém přístupu na web skrze program používající Javu.

K vytvoření certifikátu byl použit terminálový příkaz `openssl` za použití existujícího klíče s názvem `nginx-selfsigned.key`, který byl už přímo v image Dockeru, ze kterého je vytvořen kontejner [39]. Příkaz `openssl` se dotazuje uživatele na některé chybějící informace při vytváření certifikátu jako jsou doména a port, ke kterým má certifikát náležet. Některé informace jsou v tomto nasazení nepodstatné – např. země, vlastník apod.

```
$ openssl req -new -x509 -days 10000 -key nginx-selfsigned.key
-out selfsigned.crt
```

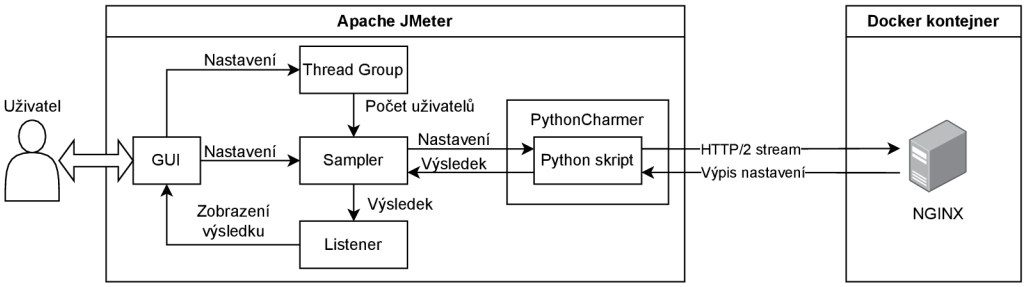
Pomocí příkazu `keytool` pak byl přidán právě podepsaný certifikát mezi důvěryhodné v rámci Java truststore. Při otevření v klasickém prohlížeči se i nadále objeví upozornění o certifikátu bez podpisu autority. To ale není pro tuto práci důležité.

```
$ keytool -import -trustcacerts -keystore
/usr/lib/jvm/java-17-openjdk-amd64/lib/security/cacerts
-storepass changeit -noprompt -alias mycert -file ./selfsigned.crt
```



Obr. 2.5: Schéma činnosti pracoviště pro testování HTTP/2.

Celé pracoviště testovacího modulu je znázorněno na obr. 2.5. Na obr. 2.6 lze vidět obdobné schéma pro sampler pomalého čtení. Pracoviště pomalého čtení se liší tím, že má ještě vřazenu obsluhu skriptu v jazyce Python a také vysílá a přijímá jiný typ dat.



Obr. 2.6: Schéma činnosti testovacího pracoviště pomalého útoku.

2.3 Poznámky k instalaci

V rámci projektu, do kterého spadá i tato práce, byly připraveny instalační skripty. Postup instalace vychází ze souboru `README.md` [34]. Jelikož je celý projekt nahrán na GitLab, jeho stažení je realizováno pomocí příkazu `git clone`.

```
$ git clone https://jmeter-access-token:<YOUR_ACCESS_TOKEN>
  @gitlab.utko.feec.vutbr.cz/ict-tester/jmeter-deployment.git
```

Tímto příkazem je vytvořen adresář, který bude obsahovat bash skripty pro instalaci. Tyto skripty automatizují přípravu, instalaci, sestavení a kompilaci celého projektu. Před prvním spouštěním je třeba provolat tři skripty.

```
$ sudo ./prepare_server.sh
$ ./build_and_deploy.sh -b
$ sudo ./start_jmeter.sh -p ~/jmeter-main
```

Výsledkem této sekvence by měl být spuštěný program Apache JMeter v grafickém režimu. Modul pro testování HTTP/2 záplavy je už v tomto stavu spustitelný. Modul pro pomalé čtení ale potřebuje doinstalovat ještě python3 se všemi důležitými balíčky. K tomuto byl vytvořen skript `python_instal.sh`.

```
$ sudo ./python_install.sh
```

V případě nasazení HTTP/2 záplavového modulu na webový server běžící na lokální smyčce je potřeba dořešit certifikaci daného serveru. Tento postup je popsán v části 2.2.

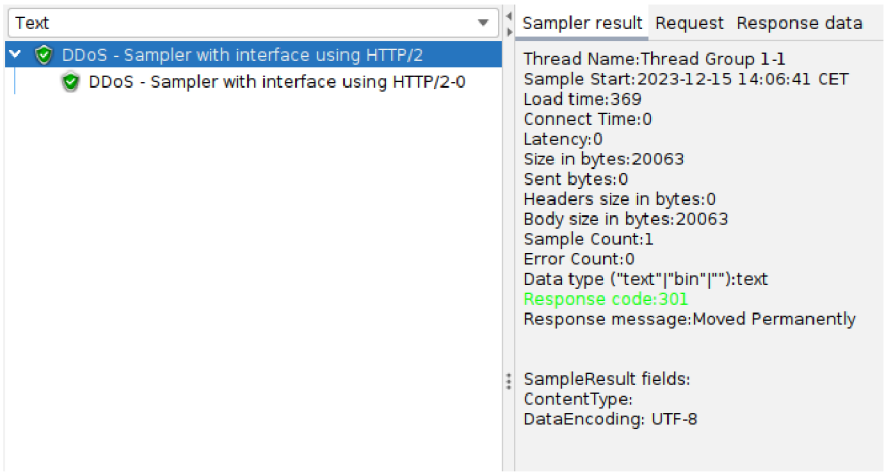
2.4 Výsledky testování

Vývoj i testování vyvinutých modulů bylo provedeno na operačním systému Ubuntu 22.04 LTS. Ten byl spuštěn na laptopu s procesorem Intel Core i5-9300H se čtyřmi jádry a osmi vlákny s maximální frekvencí 4,10 GHz. Operační paměť zařízení byla 24,0 GiB typu DDR4.

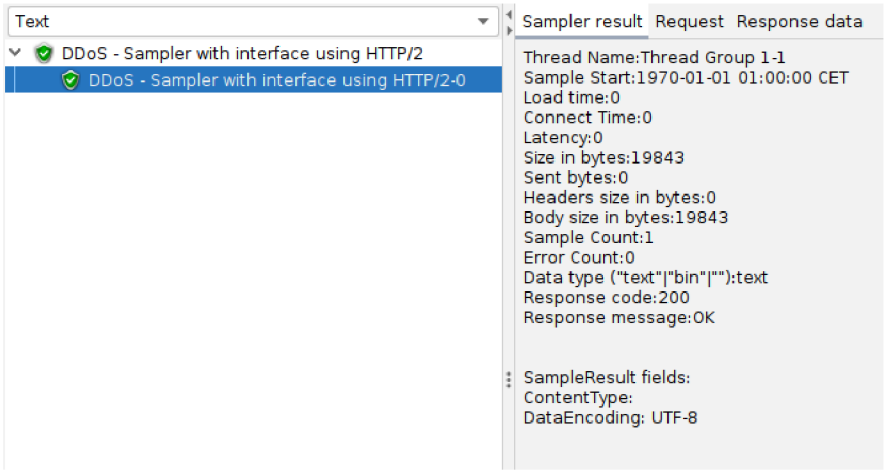
2.4.1 Testování HTTP/2 záplavového modulu

Nejprve byla ověřena správnost zasílaných požadavků. Bylo potřeba ověřit, že modul zasílá žádosti a server na požadavky reaguje. Během této fáze byla ověřena také správnost použité verze protokolu HTTP.

Dále bylo otestováno, zda funguje politika přesměrování – možnosti Follow Redirect a Redirect Automatically. Řešení těchto politik je popsáno v části 2.1.1. Jejich funkčnost je znázorněna na obr. 2.7 a 2.8 – nejprve je přijata zpráva s kódem 301 a následně je přijata přesměrovaná odpověď a kód podvýsledku je nastaven na hodnotu 200.



Obr. 2.7: Přesměrovaný výsledek politiky Follow Redirect.



Obr. 2.8: Podvýsledek přesměrovaného výsledku politiky Follow Redirect.

Další části testování bylo vyvolání HTTP-GET záplavy na testovacím serveru. Kvůli tomuto testování, které využívá velkého počtu dotazů směrem k testovacímu serveru, byly zmenšeny hardwarové prostředky kontejneru Nginx ve službě Docker, aby bylo testování proveditelné. K tomuto byl využit příkaz `sudo docker update -cpuset-cpus "1"nginx`. Tím došlo ke snížení počtu jader, které může webový server využívat, pouze na jediné jádro.

V konfiguraci útoku bylo definováno 100 virtuálních uživatelů v sekci Thread Group a v sampleru bylo zapnuto generování náhodných IP adres na virtuálním rozhraní `veth95110e3`.

Tab. 2.1: Využití zdrojů kontejneru Docker bez útoku.

ID	b5affc644262
Název	nginx
CPU %	0,00 %
Paměť použitá/dostupná	11,32 MiB / 23,32 GiB
Použití paměti	0,05 %
Síť I/O	29,8 MB / 43,2 MB
Block I/O	0 B / 172 kB
PIDS	2

Tab. 2.2: Využití zdrojů kontejneru Docker během HTTP flood útoku.

ID	b5affc644262
Název	nginx
CPU %	101,26 %
Paměť použitá/dostupná	37.66 MiB / 23,32 GiB
Použití paměti	0,16 %
Síť I/O	62.3 MB / 90,2 MB
Block I/O	0 B / 1.12 KB
PIDS	2

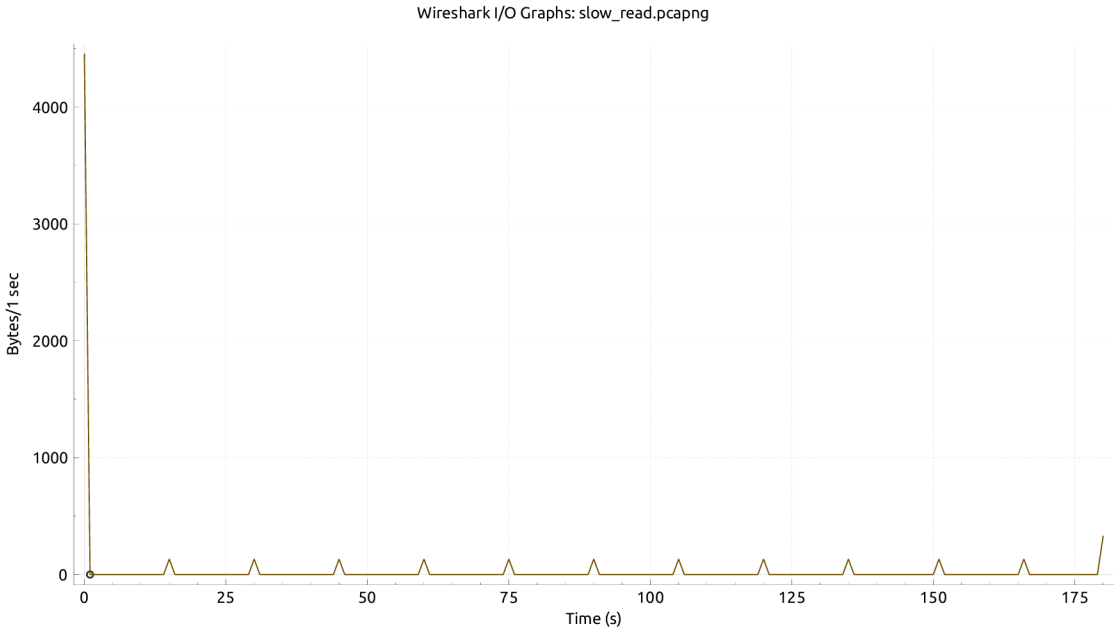
V tabulkách 2.1 a 2.2 je shrnuto hardwarové vytížení kontejneru s webovou službou před útokem a během útoku. K získání těchto dat byl použit terminálový příkaz `sudo docker stats nginx`. Jelikož však Docker vypočítává využití procesoru pomocí vzorce, který porovnává využití procesoru systémem a Dockerem během časového intervalu [41], není tento výpočet příliš přesný. Kvůli tomu Docker hlásí při

plném zatížení hodnoty vyšší než 100 %. Při povolení více jader se tato hodnota násobí ještě počtem jader, což znamená, že Docker může zobrazit využití např. 400 %, pokud by plně využil čtyři jádra.

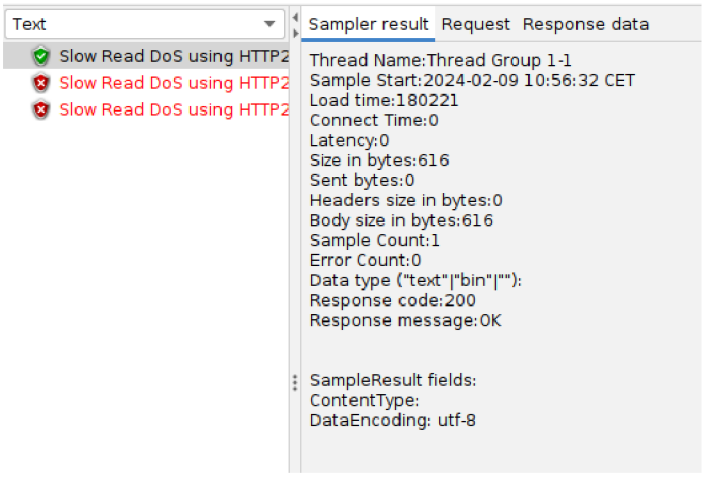
2.4.2 Testování Slow Read

Jako testovací oběť byl opět použit Docker kontejner, ve kterém byl spuštěn webový server Nginx. Nejprve bylo ponecháno nastavení z testování záplavy a bylo pouze testováno, zda modul funguje.

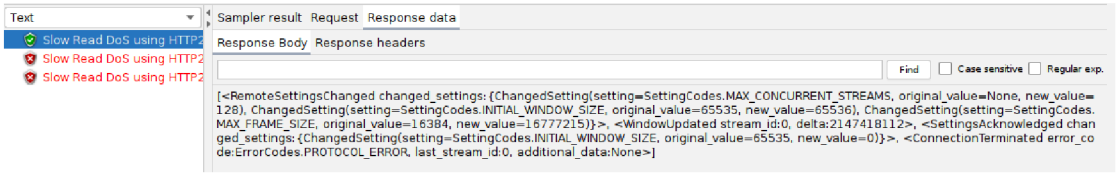
Při spuštění útoku bylo zjištěno, že každému vzorku trvá provedení 3 minuty. Průběh jednoho vzorku byl zachycen za pomocí softwaru Wireshark. I když je služba šifrovaná, lze z komunikace sestavit graf toku dat. Tento graf je zobrazen na obr. 2.9. Na průběhu lze vidět, že pouze na začátku komunikace je přenášeno větší množství dat. To je způsobeno tím, že se vytváří spojení mezi klientem a serverem. Zbytek komunikace je tvořeno pouze udržováním TCP spojení. Poté je spojení rozvázáno serverem a vzorek je možné považovat za úspěšný, jelikož bylo zatíženo spojení serveru na poměrně dlouhou dobu. Zpráva o úspěšném provedení vzorku je zobrazena na obr. 2.10. Obsah dat úspěšného vzorku je ukázán na obr. 2.11.



Obr. 2.9: Graf toku dat jednoho vzorku Slow Read.

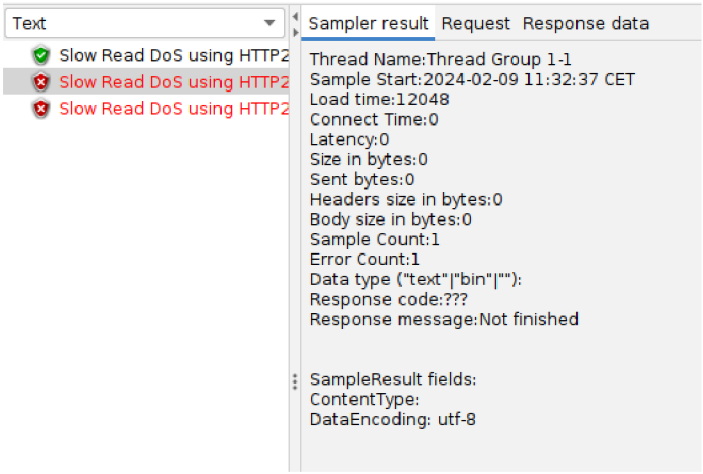


Obr. 2.10: Znázornění úspěšného výsledku.



Obr. 2.11: Obsah dat úspěšného vzorku.

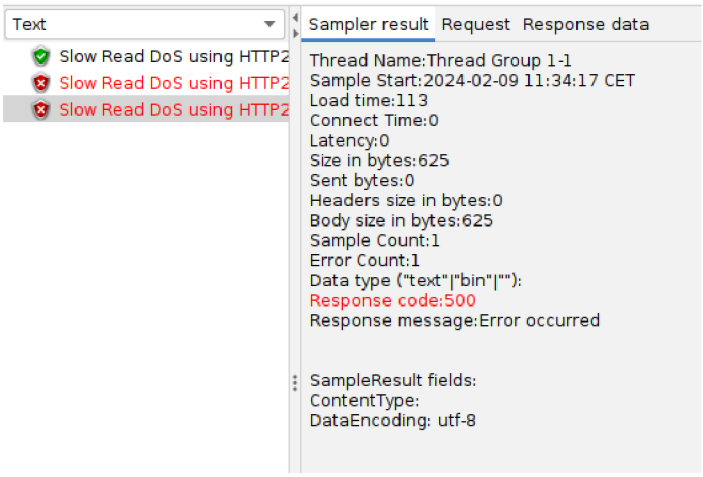
Dále bylo zjišťováno chování modulu při různých situacích. Byla otestována funkce přerušitelnosti procesu pomocí tlačítka. Zpráva o přerušení je znázorněna na obr. 2.12. Na obr. 2.13 je možno vidět, že obsah dat je dle očekávání skutečně prázdný.



Obr. 2.12: Znázornění přerušeného vzorku ve výsledku.

Obr. 2.13: Obsah dat přerušeného vzorku.

Jako další scénáře byly testovány špatně zadané parametry oběti nebo vypnutí kontejneru. Oba scénáře vedou prakticky k totožnému výsledku – cíl útoku je nedostupný, takže nelze uzavřít spojení. To vyvolá chybovou hlášku v Python skriptu. Tato hláška je zachycena v sampleru a vyhodnocena. Zpráva o neúspěchu je ukázána na obr. 2.14. Na obr. 2.15 lze vidět data tohoto výsledku obsahující slovo „Traceback“.



Obr. 2.14: Znázornění chybného vzorku ve výsledku.

Obr. 2.15: Obsah dat chybného vzorku.

Dále bylo vyzkoušeno vyvolání odepření služby. Nejprve bylo ponecháno nastavení kontejneru – jedno jádro procesoru pro kontejner. Jelikož je ale Nginx schopen

efektivně odbavovat požadavky, k odepření služby při použití jednoho jádra nedošlo. Bylo vyzkoušeno připojení i tisíce uživatelů, kdy každou sekundu přibývali noví, ale i tak se nepodařilo zamezit přístupu ke službě skrze webový prohlížeč Firefox. Test došel tak daleko, že počítači došla paměť a systém zamrzl. Bylo zaplněno všech 24 GiB operační paměti, ale služba byla stále dostupná. Běh systému Ubuntu spotřebuje zhruba 2 až 3 GiB operační paměti. Zbytek paměťového prostoru byl vyplněn procesy JMeteru.

Z tohoto důvodu byly procesorové prostředky kontejneru omezeny až na 0,1 jádro. Docker totiž umožňuje využívat zlomky jader pro různé kontejnery. K tomuto omezení byl použit příkaz `sudo docker update -cpus=0.1 nginx`. Omezení na zlomky procesorových jader je realizováno pomocí procesorového plánovače [42]. Ten omezí procesorový čas, který může proces kontejneru využít, což se bude jevit stejně, jako kdyby byla služba spuštěna na slabším procesoru. Vzhledem k tomu, že se takto upravuje procesový plánovač, je tento příkaz rozdílný oproti tomu, který byl využit pro omezení prostředků v části 2.4.1 – zde bylo totiž pevně nastaveno, že se pro účely procesu kontejneru smí využít jedno procesorové jádro.

Zde už došlo k odepření služby při použití 1000 uživatelů najednou. Nedošlo ani k přetečení paměti, i když bylo na generování útoku využito zhruba 10 GiB operační paměti. Před spuštěním útoku bylo dle sledování prostředků v systému využito 5,1 GiB a během útoku ukazoval sledovací nástroj, že bylo zaplněno 15,4 GiB.

Závěr

V rámci této práce byly moduly vyvinuty a následně otestovány na webovém serveru spouštěném pomocí služby Nginx v kontejneru programu Docker. Prvotní testovací modul práce zasílá požadavky pomocí protokolu HTTP/2 a také přijímá odpovědi, které lze zobrazit za pomocí listeneru – byl využit listener s názvem „View Results Tree“. Zde je možné podívat se na shrnutí výsledku, tělo i hlavičku zaslaného požadavku, dále také na přijatá data ve formě HTML kódu a tělo i hlavičku odpovědi.

Na tomto testovacím modulu je možné generovat útok typu HTTP flood pomocí nastavení mnoha opakování a uživatelů v části Thread Group, kteří využívají sampler „DDoS – Sampler with interface using HTTP/2“.

Při spuštění útoku HTTP flood se skokově zvýšilo použití procesu službou Nginx v kontejneru na 100 %. Útok neměl velký vliv na paměť oběti a plýtval pouze procesorem. Výsledky jsou shrnuty v tabulce 2.2. Nevýhoda tohoto útoku však je, že počítač, ze kterého byl útok spuštěn, využil všechny své procesorové prostředky na generování škodlivého provozu.

Modul vyvinutý ke spouštění útoku Slow Read byl také otestován na stejném pracovišti, jako HTTP/2 sampler, pouze s nižšími povolenými hardwarovými prostředky. Úprava byla realizována pomocí snížení procesorového času dané úloze přiděleného systémovým plánovačem.

Během testování bylo zjištěno, že výsledný modul udržuje spojení se serverem po dobu 3 minut. To je možné zkontrolovat pomocí času, který je zobrazen přímo v GUI JMeteru – v pravém horním rohu během činnosti vzorkování, nebo také v samotném výsledku vzorku.

Průběh pomalého čtení je znázorněn na obr. 2.9. Zde je opět vidět, že je spojení obsazeno po dobu 180 s. Větší počet přenesených dat je zřetelný pouze na začátku komunikace, zbytek komunikace je pouze periodické udržování spojení serveru s klientem.

Nakonec byl spuštěn i pokus o odepření služby testovacího serveru. Při použití 1000 uživatelů se útok zdařil a webová stránka nebyla dostupná. Je ovšem potřeba konstatovat, že tento typ útoku je poměrně náročný na paměť – při útoku bylo využito 10 GiB operační paměti.

Literatura

[1] Apache Software Foundation (1999 – 2023). *Apache JMeter* [Online] [cit. 2023-11-3] Dostupné z: <https://jmeter.apache.org/>

[2] HALILI E. H. *Apache JMeter* [Online]. První vydání. Packt Publishing Ltd, 2008, [cit. 2023-11-3]. ISBN 978-1-847192-95-0. Dostupné z: <https://ds.amu.edu.et/xmlui/bitstream/handle/123456789/12294/Packt.Apache.JMeter.A.Practical.Beginners.Guide.To.Automated.Testing.And.Performance.Measurement.For.Your.Websites.Jun.2008.ISBN.1847192955.pdf?sequence=1&isAllowed=y >

[3] Oracle Corporation. *Java* [Online]. [cit. 2024-4-8] Dostupné z: <https://www.java.com/en/>

[4] KREGER, H.; WARD, H.; WILLIAMSON, L. *Java and JMX: Building Manageable Systems* [Online]. První vydání. Addison-Wesley Professional, 2003, [cit. 2023-11-3]. ISBN 978-0672324086. Dostupné z: <https://books.google.cz/books?id=U3En1nWHKcsC&printsec=frontcover&hl=cs&source=gbs_atb#v=onepage&q&f=false >

[5] RAJKUMAR. *Best JMeter Alternatives in 2023* [Online]. Software Testing Material. Červenec 2023. Dostupné z: <https://www.softwaretestingmaterial.com/jmeter-alternatives/>

[6] Spirent Communications. *Spirent Avalanche Application Performance Testing Solution* [Online]. 2021. Dostupné z: <https://www.spirent.com/products/avalanche-security-testing>

[7] NIELSEN, H.; MOGUL, J.; MASINTER, L. M.; FIELDING, R. T.; GETTYS, J.; LEACH P. J.; BERNERS-LEE, T. *Hypertext Transfer Protocol – HTTP/1.1* [Online]. RFC Editor. Červen 1999. [cit. 9. 11. 2023] DOI: 10.17487/RFC2616. Dostupné z: <https://datatracker.ietf.org/doc/html/rfc2616>

[8] GOURLEY, D.; TOTYY B. *HTTP: The Definitive Guide* [Online]. První vydání. O'Reilly Media, Inc., 2002. [cit. 2023-11-3]. ISBN 978-1565925090. Dostupné z: <https://books.google.cz/books?hl=cs&lr=&id=3EybAgAAQBAJ&oi=fnd&pg=PT4&dq=http&ots=X67YUbhYYo&sig=m3Lm3yTRGmIZIsYM-ecFXpL0q4w&redir_esc=y#v=onepage&q&f=false>

[9] POLLARD, B. *HTTP/2 in Action* [Online]. Simon and Schuster, 2019. [cit. 2023-11-3]. ISBN 978-1638352334. Dostupné z: <https://books.google.cz/books?hl=cs&lr=&id=azozEAAAQBAJ&oi=fnd&pg=PT15&dq=http/2&ots=25KewxbG5e&sig=H8rtMsIT_5e_olunRW3LCnbksks&redir_esc=y#v=onepage&q&f=false>

[10] BELSHE, M.; PEON, R; THOMSON, M., E. *Hypertext Transfer Protocol Version 2 (HTTP/2)* [Online]. RFC Editor. Květen 2015. [cit. 2023-11-3]. ISSN: 2070-1721 Dostupné z: <https://datatracker.ietf.org/doc/html/rfc7540>

[11] HUYSEGEMS, R; VAN DER HOOFT, J; BOSTOEN, T; ALFACE, P., R.; PETRANGELI, S. et al. *HTTP/2-based methods to improve the live experience of adaptive streaming* In: *23e ACM International Conference on Multimedia 5MM2015, Proceedings*. Brisbane, Australia, 2015-10-26 – 2015-10-30 [Online]. Association for Computing Machinery, 2015. s. 541–550. [cit. 2023-11-03]. ISBN: 978-1-4503-3459-4 Dostupné z: <https://biblio.ugent.be/publication/7016610>

[12] The Chromium Projects. *SPDY: An experimental protocol for a faster web* [Online]. [cit. 2023-11-7] Dostupné z: <https://www.chromium.org/spdy/spdy-whitepaper/>

[13] VARVELLO, M.; SCHOMP, K.; NAYLOR, D.; BLACKBURN, J.; FINAMORE, A.; PAPAGIANNAKI, K. *Is the Web HTTP/2 Yet?* [Online]. Springer, Cham, 2016, [cit. 2023-11-3]. ISBN 978-3-319-30505-9. Dostupné z: <https://doi.org/10.1007/978-3-319-30505-9_17>

[14] THOMSON, M.; BENEFIELD, C. *Hypertext Transfer Protocol Version 2 (HTTP/2)* [Online]. RFC Editor. Červen 2022. [cit. 2023-11-7] ISSN: 2070-1721. Dostupné z: <https://datatracker.ietf.org/doc/html/rfc9113>

[15] DIERKS, T.; RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.2* [Online]. RFC Editor. Srpen 2008. [cit. 2023-11-9] DOI: 10.17487/RFC5246. Dostupné z: <https://datatracker.ietf.org/doc/html/rfc5246>

[16] Google.com *How it works: The novel HTTP/2 'Rapid Reset' DDoS attack* [Online]. Říjen 2023. [cit. 2023-11-11] Dostupné z: <https://cloud.google.com/blog/products/identity-security/how-it-works-the-novel-http2-rapid-reset-ddos-attack>

[17] BISHOP, M.; AKAMAI E. *HTTP/3* [Online]. RFC Editor. Červen 2022. [cit. 2024-2-5] DOI: 10.17487/RFC9114 Dostupné z: <https://datatracker.ietf.org/doc/html/rfc9114>

[18] ROSKIND, J. *Experimenting with QUIC* [Online]. Chromium Blog. Červen 2013. [cit. 2024-2-5] Dostupné z: <https://blog.chromium.org/2013/06/experimenting-with-quic.html>

[19] Cybersecurity & Infrastructure Security Agency. *Understanding Denial-of-Service Attacks* [Online]. Únor 2018. [cit. 2023-11-12] Dostupné z: <https://www.cisa.gov/news-events/news/understanding-denial-service-attacks>

[20] VORMAYR, G.; ZSEBY, T.; FABINI, J. *Botnet Communication Patterns*. In: IEEE Communications Surveys & Tutorials [Online]. 2017. s. 2768–2796. [cit. 2023-11-12] ISSN: 1553-877X Dostupné z: <https://ieeexplore.ieee.org/abstract/document/8026031>

[21] Národní úřad pro kybernetickou bezpečnost. *Varování o hrozbě Emotet-Trickbot-Ryuk* [Online]. Prosinec 2019. [cit. 2023-11-13] Dostupné z: <https://www.nukib.cz/cs/infoservis/hrozby/1478-varovani-o-hrozbe-emotet-trickbot-ryuk/>

[22] GRABOVSKY, S.; CIKA, P.; ZEMAN, V.; CLUPEK, V; SVEHLAK M.; KLIMES, J. *Denial of Service Attack Generator in Apache JMeter* [Online]. 2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). IEEE, 2018. s. 1–4. [cit. 2023-11-16] ISBN: 978-1-5386-9361-2. Dostupné z: <https://ieeexplore.ieee.org/document/8631212 >

[23] GUPTA, N.; JAIN, A.; SAINI, P.; GUPTA, V. *DDoS attack algorithm using ICMP flood*. In 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)[Online]. New Delhi, India, 2016. s. 4082–4084. [cit. 2023-12-6] Dostupné z: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7725026&isnumber=7724213>

[24] SANJEEV, K. *Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet* In: Second International Conference on Internet Monitoring and Protection (ICIMP 2007) [Online]. San Jose, CA, USA 2007. s. 25. [cit. 2023-11-16] DOI: 10.1109/ICIMP.2007.42. Dostupné z: <https://ieeexplore.ieee.org/abstract/document/4271771>

[25] ABDOLLAHI, A.; FAITHI, M. *An Intrusion Detection System on Ping of Death Attacks in IoT Networks*. In: Wireless Personal Communications [Online]. s. 2057–2070. January 2020. [cit. 2023-11-16] ISSN: 1572-834X. Dostupné z: <https://link.springer.com/article/10.1007/s11277-020-07139-y>

[26] MUTUTU, L.; SALEH, R.; MATRAWY, A. *Improved SDN responsiveness to UDP flood attacks*. In: 2015 IEEE Conference on Communications and Network Security (CNS) [Online]. Florencie, Itálie, 2015. s. 715–716. DOI: 10.1109/CNS.2015.7346900 Dostupné z: <https://ieeexplore.ieee.org/document/7346900>

[27] SHOREY, T.; SUBBIAH. D.; GOYAL, A.; SAKXENA, A.; MISHRA, A. K. *Performance Comparison and Analysis of Slowloris, GoldenEye and Xerxes DDoS Attack Tools*. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI) [Online]. Bangalore, Indie, 2018. s. 318–322. [cit. 2023-10-11] DOI: 10.1109/ICACCI.2018.8554590 Dostupné z: <https://ieeexplore.ieee.org/abstract/document/8554590>

[28] KRČMÁŘ, P. *Útok Slowloris aneb plíživé nebezpečí pro web servery* [Online]. root.cz. Květen 2011. [cit. 2023-10-11] Dostupné z: <https://www.root.cz/clanky/utok-slowloris-aneb-plizive-nebezpeci-pro-web-servery/>

[29] SABRI, S.; ISMAIL, N.; HAZZIM, A. *Slowloris DoS Attack Based Simulation*. In: IOP Conference Series: Materials Science and Engineering [Online]. Únor 2021. IOP Publishing, 2021. [cit. 2023-10-11] DOI: 10.1088/1757-899X/1062/1/012029 Dostupné z: <https://iopscience.iop.org/article/10.1088/1757-899X/1062/1/012029/meta>

[30] CAMBIASSO, E.; PAPALEO, G.; AIELLO, M. *Taxonomy of Slow DoS Attacks to Web Applications* In: THAMPI, S. M.; ZOMAYA, A. Y.; STRUFE, T.; ALCARAZ CALERO, J. M.; THOMAS, T. (eds) Recent Trends in Computer Networks and Distributed Systems Security [Online]. Springer Berlin Heidelberg. s. 195–204. [cit. 2023-10-11] ISBN: 978-3-642-34135-9. Dostupné z: <https://doi.org/10.1007/978-3-642-34135-9_20>

[31] SIKORA, M.; FUJDIAK, R.; KUCHAŘ, K.; HOLASOVÁ, E.; MIŠUREC, J. *Generator of Slow Denial-of-Service Cyber Attacks* [Online]. SENSORS. Roč. 21, č. 16, s. 1–27. MDPI, 2021. [cit. 2023-10-11] ISSN: 1424-8220. Dostupné z: <https://doi.org/10.3390/s21165473>

[32] TRIPATHI, N.; HUBBALLI, N. *Slow rate denial of service attacks against HTTP/2 and detection* [Online]. Computers & Security. Leden 2018. s. 255–272. [cit. 2023-11-11] Dostupné z: <https://doi.org/10.1016/j.cose.2017.09.009 >

[33] LING, X.; WU, C.; JI, S.; HAN, M. *H_2DoS: An Application-Layer DoS Attack Towards HTTP/2 Protocol.* In: LIN, X.; GHOBRANI, A.; REN, K.; ZHU, S.; ZHANG, A. (eds). Security and Privacy in Communication Networks [Online]. SecureComm 2017. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, Cham, 2018. s. 550–570. [cit. 2023-11-12] ISBN: 978-3-319-78813-5. Dostupné z: <https://doi.org/10.1007/978-3-319-78813-5_28>

[34] JEDLIČKA, J. *Integrace a automatizace nasazení aktualizovaných modulů pro zátěžové testování* [Online]. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2023. [cit. 2023-11-27] Dostupné z: <https://www.vut.cz/studenti/zav-prace?zp_id=151247>

[35] Webtide. *Class HTTP2Client* [Online]. [cit. 2024-2-13] Dostupné z: <https://eclipse.dev/jetty/javadoc/jetty-12/org/eclipse/jetty/http2/client/HTTP2Client.html>

[36] Gradle Inc. *Gradle Release Notes* [Online]. [cit. 2024-2-13] Dostupné z: <https://docs.gradle.org/3.2-rc-3/release-notes.html>

[37] DHIVER, B. *Slow Rate HTTP2 DoS PoC* [Online]. github.io. [cit. 2024-2-13] Dostupné z: <https://github.com/Dhiver/SlowRate-HTTP2-DoS/blob/master/README.md>

[38] BENFIELD, C. *hyperframe 6.0.1* [Online]. pypi.org. [cit. 2024-2-130] Dostupné z: <https://pypi.org/project/hyperframe/>

[39] POLINOWSKI, M. *NGINX HTTP/2 Docker Ingress* [Online]. github.io. [cit. 2023-11-20] Dostupné z: <https://mpolinowski.github.io/docs/DevOps/NGINX/2023-06-13-nginx-docker-ingress/2023-06-13/>

[40] Mozilla Foundation. *HTTP response status codes* [Online]. MDN Web Docs. [cit. 2023-11-20] Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

[41] VAN STIJN, S.; ALBERS, H. NEPHIN, D.; DEMEESTER, V.; DURRHEIMER, S. et al. *The Docker CLI* [Online]. Github – docker/cli [cit. 2023-11-25] Dostupné z: <https://github.com/docker/cli>

[42] Docker Inc. *Runtime options with Memory, CPUs, and GPUs* [Online]. Docker Docs. [cit. 2024-3-8] Dostupné z: <https://docs.docker.com/config/containers/resource_constraints/>

Seznam symbolů a zkratek

ACK	Acknowledgement
ASCII	American Standard Code for Information Interchange
CSS	Cascading Style Sheets
DNS	Domain Name System
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MPEG	Moving Picture Experts Group
\n	Newline
QUIC	Quick UDP Internet Connections
\r	Carriage return
SYN	Synchronize
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WAV	Waveform audio file format

A Obsah elektronické přílohy

Elektronická příloha neobsahuje celý projekt, protože by byla příliš velká. Jsou zde přiloženy pouze zdrojové kódy a skripty, které byly vytvořený přímo v rámci této práce.

Všechny zdrojové kódy jsou uloženy na GitLabu Ústavu Telekomunikací spolu s celým JMeter projektem. GitLab je dostupný na adrese `https://gitlab.utko.feec.vutbr.cz/`. Postup instalace pomocí nástroje `git` je popsán v části 2.3.

```
/ ..... Kořenový adresář přiloženého archivu
├── http2 ..... Adresář pro třídy modulů
│   ├── InterfaceHttpSamplerGui.java ..... Třída GUI testovacího modulu
│   ├── Http2Sampler.java ..... Třída implementující sampler testovací modul
│   ├── SlowReadSampler.java ..... Třída implementující sampler pomalého čtení
│   ├── SlowReadSamplerGui.java ..... Třída GUI pomalého čtení
│   └── tools ..... Adresář pomocných nástrojů
│       ├── Pathfinder.java ..... Nástroj pro hledání kořenového adresáře projektu
│       └── PythonCharmer.java ..... Nástroj pro volání Python skriptu
├── scripts ..... Adresář pro Python skripty
│   └── SlowRead.py ..... Python skript pomalého čtení
└── python_install.sh ..... Bash skript pro instalaci Pythonu vč. balíčků
```