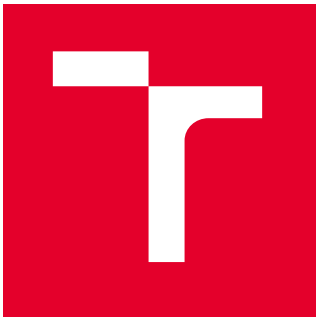


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A IMPLEMENTACE MODULU PRO ZÁTĚŽOVÉ TESTOVÁNÍ HTTPS POŽADAVKY V NÁSTROJI APACHE JMETER

DESIGN AND IMPLEMENTATION OF A MODULE FOR LOAD TESTING OF HTTPS REQUESTS IN APACHE JMETER

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Šimon Čížek

AUTHOR

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

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Šimon Čížek ***ID:*** 230792

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

NÁZEV TÉMATU:

Návrh a implementace modulu pro zátěžové testování HTTPS požadavky v nástroji Apache JMeter

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je návrh a implementace modulu pro zátěžové testování v nástroji Apache JMeter. Student se seznámí s nástrojem pro zátěžové testování a již existujícími moduly rozšiřujícími možnosti testování. Student zde navrhne nový či modifikuje existující modul pro testování webových stránek na záplavu HTTPS požadavků z více zdrojových IP adres. V teoretické části práce student provede rešerši možností, jak realizovat záplavové útoky z více zdrojových IP adres a jednotlivé možnosti vhodně vyhodnotí. V praktické části práce student implementuje patřičnou funkcionalitu v rozšiřujícím modulu Apache JMeteru. Dále student vytvoří vhodné scénáře pro zátěžové testování a demonstruje je na vybraném příkladu. Výsledky z navržených scénářů student vhodně vizualizuje a popíše.

DOPORUČENÁ LITERATURA:

[1] RODRIGUES, Antonio Gomes, Bruno DEMION a Philippe MOUAWAD. Master Apache JMeter - From Load Testing to DevOps: Master performance testing with JMeter. Packt Publishing Ltd, 2019.

[2] ATAR, Afsana. Mastering JMeter 5.0. Packt Publishing, 2020.

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

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

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

ABSTRAKT

Bakalářská práce se zabývá DDoS útoky přes HTTPS protokol a jejich simulací z jednoho síťového rozhraní. Na základě rozboru možností pro odesílání požadavků z falešných IPv4 a IPv6 adres byl naprogramován zásuvný modul do softwaru Apache JMeter. Následně byly otestovány jeho dopady prostřednictvím vytvořeného scénáře na postavený server se softwarovými webovými servery Nginx a Apache2. V závěru jsou uvedeny výsledky vygenerované zátěže na obou serverech.

KLÍČOVÁ SLOVA

JMeter, DDoS, testování, útok, HTTPS, webový server

ABSTRACT

The Bachelor Thesis focuses on DDoS attacks over the HTTPS protocol and their simulations from a single network interface. Based on the analysis of options for sending requests from spoofed IPv4 and IPv6, a plugin module was developed for Apache JMeter software. Subsequently, module impacts were tested using a created scenario on Nginx and Apache2 test servers. The conclusion presents the results of the generated load on both servers.

KEYWORDS

JMeter, DDoS, testing, attack, HTTPS, web server

ČÍŽEK, Šimon. *Návrh a implementace modulu pro zátěžové testování HTTPS požadavky v nástroji Apache JMeter*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 58 s. Bakalářská práce. Vedoucí práce: Ing. Mgr. Pavel Šeda, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Šimon Čížek
VUT ID autora:	230792
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Návrh a implementace modulu pro zátěžové testování HTTPS požadavky v nástroji 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 rodině za neutuchající podporu, vedoucímu bakalářské práce panu Ing. Mgr. Pavlu Šedovi, Ph.D. za odborné vedení, trpělivost a ponechání volnosti v samostatné činnosti a dále Bc. Jakubu Jedličkovi za užitečné rady.

Obsah

Seznam obrázků

Úvod

DDoS útoky se stávají součástí všedního života, neboť jejich četnost a objem každým rokem narůstá – jen v roce 2022 byl nárůst oproti předchozímu roku o 109 % [1, 2]. Jejich největší výhodou je, že jsou poměrně nenáročné na vytvoření, obtížně filtrovatelné, neboť na první pohled působí jako legitimní komunikace a v případě úspěšného útoku mohou být škody fatální. Například už v roce 2000 teprve patnáctiletý Michael Calce zaútočil DDoS útokem na webové servery Amazonu, CNN, eBay, Yahoo! a Dellu. Celková škoda přesáhla miliardu dolarů [3].

Není žádný důvod předpokládat, že v budoucnu se těchto útoků zbavíme, naopak s příchodem IoT (Internet of Things) můžeme být stále častěji svědky situací, kdy naše lednička bude zcela bez vědomí majitele útočit například na Pentagon. Z těchto důvodů je příprava na tyto útoky aktuálnější než kdy jindy. Tato práce má za cíl vytvořit simulaci takového útoku z jednoho stroje přes HTTPS protokol, aby připomínal masivní záplavový útok z velkého počtu zařízení.

V této bakalářské práci je sestaven a nakonfigurován testovací webový server pro DDoS útoky, jsou rozebrány protokoly HTTP, HTTPS a jejich využití pro DDoS. Seznámíme se též s protokoly IPv4 a IPv6 a jejich využitím pro simulaci útoku z více zařízení. Tyto znalosti jsou uplatněny v rozšíření a zdokonalení zásuvného modulu do softwaru Apache JMeter, který je schopen DDoS útok přes HTTPS protokol simulovat. V poslední části jsou vytvořeny testovací scénáře a následně je jeden vybraný otestován na webovém serveru s ukázkou výsledků zatížení serveru.

1 DDoS

DDoS (Distributed Denial Of Service) je typ útoku na webové služby, jehož účelem je danou službu zpomalit či zcela znepřístupnit legitimním uživatelům. Bývá též využíván jako zastírací manévr, kdy účelem hlavního útoku je například získání dat. Detekční mechanismy a lidské zdroje jsou v daný okamžik zaneprázdněny masivním provozem DDoS útoku a ostatní podezřelý provoz nemusí detekovat. Je považován za jeden z nejnebezpečnějších útoků, který může mít fatální následky. První DDoS útoky se začaly objevovat v roce 1996, ale jejich potenciál začínal být odhalován až v červnu 1998, kdy tomuto útoku čelilo několik velkých organizací. V srpnu 1999 podlehla na tehdejší dobu masivnímu útoku Minnesotská univerzita (University of Minnesota), jehož důsledkem bylo vyřazení celé IT infrastruktury na několik dní [4].

V současnosti jsou DDoS útoky velmi rozšířené, týdně jsou detekovány řádově desítky tisíc útoků tohoto typu a jejich velikost (poslední data ukazují, že masivnější útoky pravidelně přesahují 100 Gb hranici) a kvantita, i v souvislosti se současnou geopolitickou situací, stále roste – například na banku Sberbank bylo za první dva měsíce třetího kvartálu roku 2022 detekováno 450 DDoS útoků, což je stejný počet jako za předchozích pět let dohromady [5]. Četnosti útoků též napomáhá, že jejich pořízení nemusí být příliš finančně nákladné. Celodenní DDoS útok o objemu 20–50 tisíc spojení se prodává za cenu okolo 200 dolarů. V případě krátkého desetiminutového útoku o rychlosti 125 Gb/s si útočníci účtují okolo 20 dolarů.[1]

DDoS útok obvykle probíhá z takzvaného botnetu[2] připojováním mnoha zařízení na server, kde zaplavují síť náhodnými daty, udržují otevřená spojení, zatěžují výpočetní kapacity apod. [6] Požadavky mohou dosáhnout rychlosti až do řádu terabitů za sekundu. Největší známý útok byl k 8. listopadu 2022 na servery Googlu, a to 1. června 2022, kdy dosáhl ve vrcholu 46 000 000 požadavků za sekundu [7]. DDoS útoky dělíme na záplavové (též volumetrické), protokolové a aplikační.

První zmíněný typ využívá k útoku vygenerování velkého provozu k zaplnění šířky pásma mezi obětí a internetem nebo zahlcení sítě oběti. Nejrozsáhlejší útoky v současnosti dosahují rychlosti několika terabitů za sekundu, což odpovídá asi 9 000 obvyklých internetových připojení. Útok může být veden skrze UDP (User Datagram Protocol), kde dochází k zahlcování systému díky žádostem o informace od aplikace naslouchající na daném portu. Server každý takový požadavek kontroluje nebo odesílá odpověď, dochází k vyčerpání šířky pásma a služba se stává nedostupnou [8].

Protokolový DDoS útok zneužívá stavbu komunikačních protokolů

[1] Vzhledem k povaze není zdroj uveden.

[2] Síť počítačů infikovaných softwarem, který je možno centrálně řídit. Typicky slouží k rozesílání spamu, DDoS útokům apod. V ideálním případě by jednotlivé stroje měly být co nejvíce geograficky vzdáleny, aby nešly jednoduše odstřihnout na základě IP adres.

(např. TCP/IP) k vyčerpání zdrojů napadeného zařízení. Jako příklad může být uveden SYN flood útok, který se vyznačuje odesíláním velkého množství požadavků na oběť, nicméně odpovědi zahazuje, z tohoto důvodu tedy nedochází při třícestném handshaku k dokončení a serveru postupně ubývají volná spojení až k úplnému vyčerpání, kdy není možno navázat další. Rychlost těchto útoků z důvodů odesílání požadavků (paketů) měříme v paketech za sekundu (PPS – packets per second). Útoky v současnosti mohou dosahovat až stovky milionů paketů za sekundu [8].

Poslední typ – útok na aplikační vrstvě, je využíván pro veřejně přístupné aplikace, kde útočník posílá na oběť velký objem falešného provozu. Příkladem tohoto typu je HTTP flood, který zaplavuje systém za normálních okolností standardními požadavky HTTP POST a HTTP GET. I při dostatečné šířce pásma dochází k výpočetnímu zahlcení. Tento typ útoku dosahuje objemu až desítek milionů požadavků za sekundu (RPS – requests per second) [8]. V následující části bude podrobněji popsán HTTPS flood (DDoS útok přes protokol HTTPS), nejdříve je ovšem třeba porozumět protokolu HTTPS.

1.1 HTTPS protokol

HTTPS (Hypertext Transfer Protocol Secure) je protokol aplikační vrstvy referenčního modelu ISO/OSI, jenž je v počítačových sítích používán pro přenos dat mezi počítači. Pro určení konkrétních souborů požadovaných k přenesení je využíván identifikátor URL (Uniform Resource Locator). Tento textový řetězec v sobě obsahuje protokol, jímž se bude k souborům přistupovat, adresu serveru a cestu k požadovanému souboru. Standardně vystupuje na portu 443.

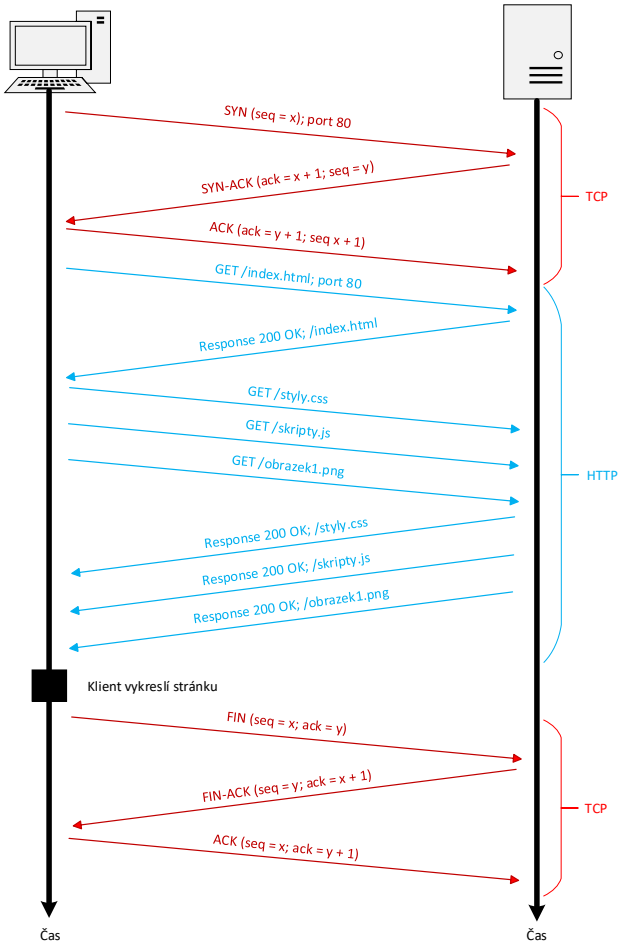
Protokol je nadstavbou protokolu HTTP, kde navíc využívá protokol SSL (Secure Sockets Layer), nebo TLS (Transport Layer Security). TLS je ve skutečnosti novější verze SSL, nicméně v literatuře se obvykle stále používají oba pojmy.[3] Oba tyto protokoly fungují na čtvrté vrstvě referenčního modelu ISO/OSI. Mohou být podobným způsobem využívány i u jiných protokolů – FTP, SMTP, IMAP a mohou být implementovány na jakýkoli další protokol, systém použití je vždy stejný. V současnosti se setkáme s TLS protokoly verze 1.2 a 1.3 [9].

Díky SSL/TLS HTTPS splňuje bezpečnostní triádu (CIA) – důvěrnost (Confidentiality), integritu (Integrity) a dostupnost (Availability). Důvěrnost je zajištěna obvykle symetrickou šifrou, kterou protokol pro přenos vyžaduje a inicializační komunikací, která bude vysvětlena později v tomto textu. Šifry jsou voleny dle výpočetního výkonu koncových zařízení a požadavků na bezpečnost přenášených dat.

[3] SSL název zůstal i v konfiguračních souborech mnoha serverů, ačkoli skutečná komunikace je TLS.

Integritu řeší hash algoritmus, který zajišťuje, že případná změna dat bude rozpoznána [10].

Samotný HTTP protokol funguje na modelu klient–server. Klient nejdříve zašle požadavek a server pošle odpověď. Požadavky jsou několika typů, např.: GET – pošli data, PUT – uprav data, DELETE – smaž data apod. Samotné připojení na webový server prostřednictvím HTTP/2[4] protokolu umožňující multiplexování[5] je schematicky znázorněno na obrázku 1.1 a 1.3.



Obr. 1.1: Diagram komunikace přes HTTP

Komunikace přes HTTPS je o něco složitější, poněvadž do komunikace navíc vstupuje důvěryhodná autorita, která zajistí, že kontaktovaný server je skutečně

[4]Označení verze protokolu.

[5]Proces, jenž umožňuje více datových toků shrnout do jednoho. V uvedeném příkladu klient najednou zažádá o slyly.css, scripty.js a obrazek1.png.

ten, za nějž se vydává. V prvním kroku klient (kdy zařízení rovnou vyžaduje zabezpečené stránky) po navázání TCP spojení od serveru vyžádá zabezpečené stránky, server odpoví zasláním svého veřejného klíče a certifikátu, které jsou podepsány soukromým klíčem certifikační autoritou. V prohlížeči jsou již veřejné klíče certifikačních autorit nainstalovány, tudíž může dojít k ověření a certifikát může být prohlášen za důvěryhodný. Tím je zajištěna autenticita serveru. Klientské zařízení vygeneruje klíč k symetrickému šifrovacímu algoritmu, nazývanému *shared secret*.

Tento klíč pošle zašifrovaný veřejným klíčem serveru, který si ho svým soukromým klíčem dešifruje. Další komunikace už probíhá šifrovaná symetrickou šifrou a klíčem, kterou vygeneroval klient a zaslal serveru. Pokud server podporuje HTTP i HTTPS, dokáže vrátit šifrovanou i nešifrovanou verzi webu, avšak pro nezabezpečenou verzi je třeba protokol HTTP vynutit napsáním do URL řádku – HTTPS protokol je v prohlížečích preferován. Weby vyžadující vyšší úroveň zabezpečení využívají HSTS (HTTP Strict Transport Security), kdy vynucují spojení přes zabezpečený HTTPS protokol [11].

Komunikace pak vypadá následovně (předpokládejme, že jde o první připojení na danou webovou stránku): proběhne třícestný TCP handshake, klient vyžádá stránky HTTP dotazem a server odpoví kódem 301, případně 307 a do hlavičky zapíše HSTS. Kód 301 říká prohlížeči, že požadavek by měl být směrován na jinou adresu. Druhý zmíněný kód je obvykle využíván pro přesměrování v případě, kdy je stránka dočasně přesunuta, nicméně vývojář Googlu uvádí, že může být použit i v tomto případě [12, 13]. Přesměrování pouze kódy 301/307 je náchylné na útok typu man-in-the-middle, neboť první HTTP dotaz ještě není kryptograficky zabezpečen (jak je vidět na obrázcích 1.2 a 1.3) a útočník může komunikaci přesměrovat na svoji verzi webu.

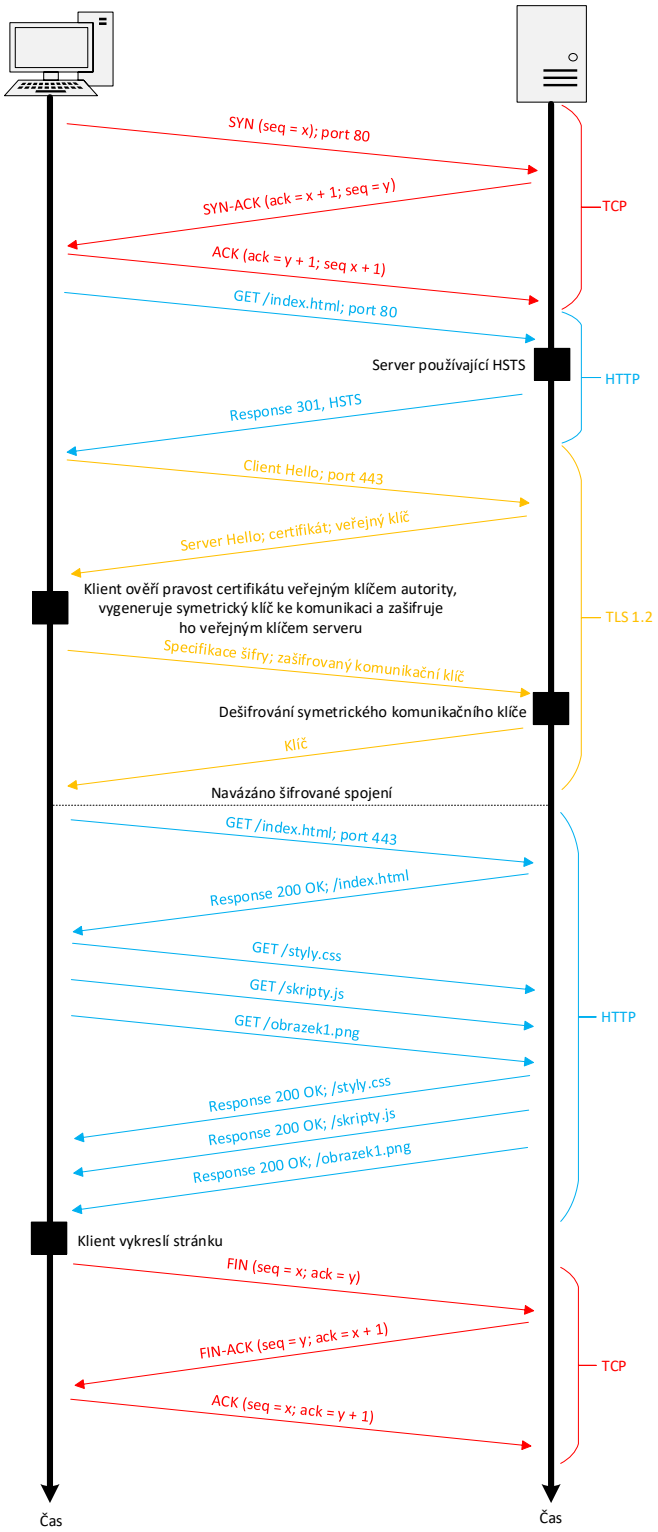
Z tohoto důvodu nastupuje HSTS, který informuje prohlížeč, že při všech dalších navštíveních webu se má připojovat ihned přes HTTPS protokol. Je dobré si uvědomit, že prvotní spojení je ovšem na útok man-in-the-middle stále náchylné, na rozdíl od všech dalších [14]. V případě, kdy server vyžaduje pouze zabezpečenou komunikaci, vypadá komunikace tak, jak je znázorněno na obrázcích 1.2 a 1.3.

1.2 HTTPS flood

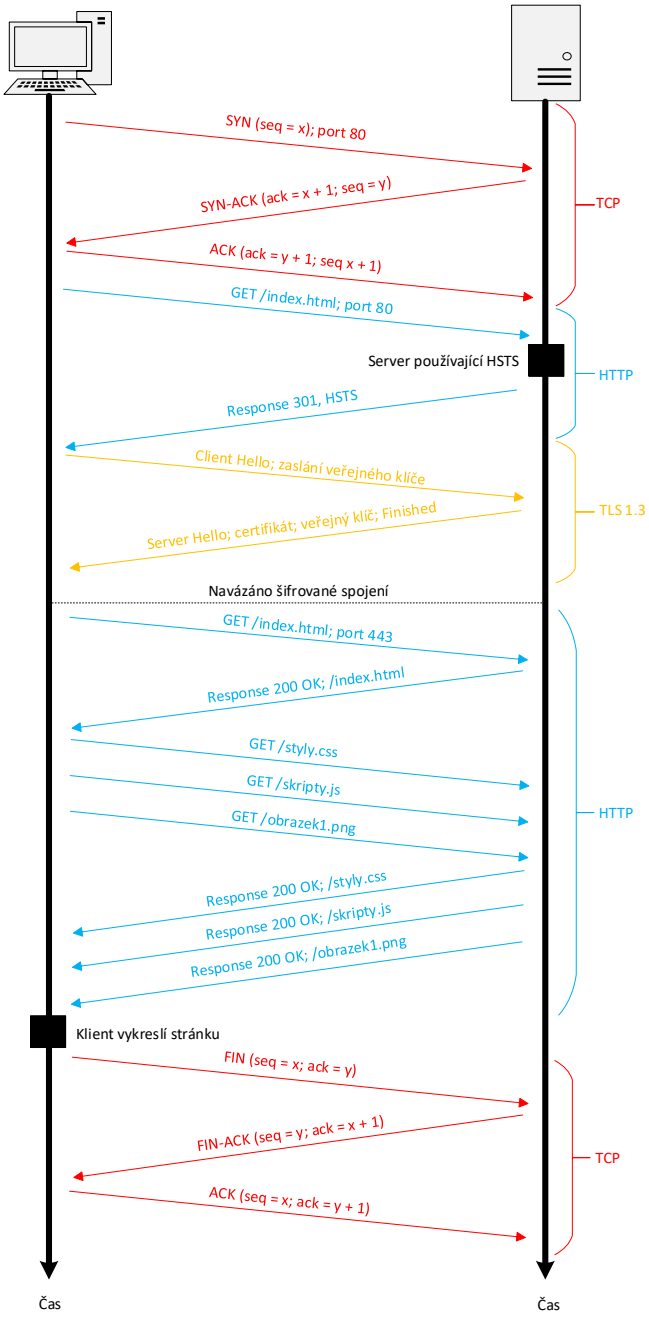
HTTPS flood patří do skupiny volumetrických útoků (viz kapitola 1). Funguje v principu podobně jako HTTP flood, oba jsou určeny k zahlcení webového serveru neustálým dotazováním z více klientských zařízení. Když je kapacita serveru zahlcena, server nemá prostředky k obsloužení dalších požadavků. HTTPS útok může ovšem zahltit i SSL/TLS démona kvůli vytížení výpočetních kapacit pro činnost asymetrické kryptografie. Co nastane dříve, záleží na dostupné šířce pásma serveru a výpočetní kapacitě.

Obvykle jsou v útocích obsaženy požadavky GET a POST, můžeme se ovšem setkat i s unikátnostmi, jako například útok přes PUT, DELETE a podobně [15]. Na rozdíl od jiných DDoS útoků nepoužívá různě poškozené pakety, ale požadavky jsou zcela legitimní, to výrazně ztěžuje detekci a možnost obrany proti tomuto útoku.

Útok je nejefektivnější, když každým přístupem dokáže alokovat co největší množství výpočetní kapacity. Za tímto účelem se používají právě POST dotazy, jelikož mohou obsahovat parametry, jejichž zpracování server zatíží více, než prosté odeslání vyžádaných dat. HTTP GET požadavky jsou ovšem jednodušší vytvořit, jsou univerzálnější a lze je lépe škálovat v botnetu [16].



Obr. 1.2: Diagram komunikace přes HTTPS s TLS 1.2

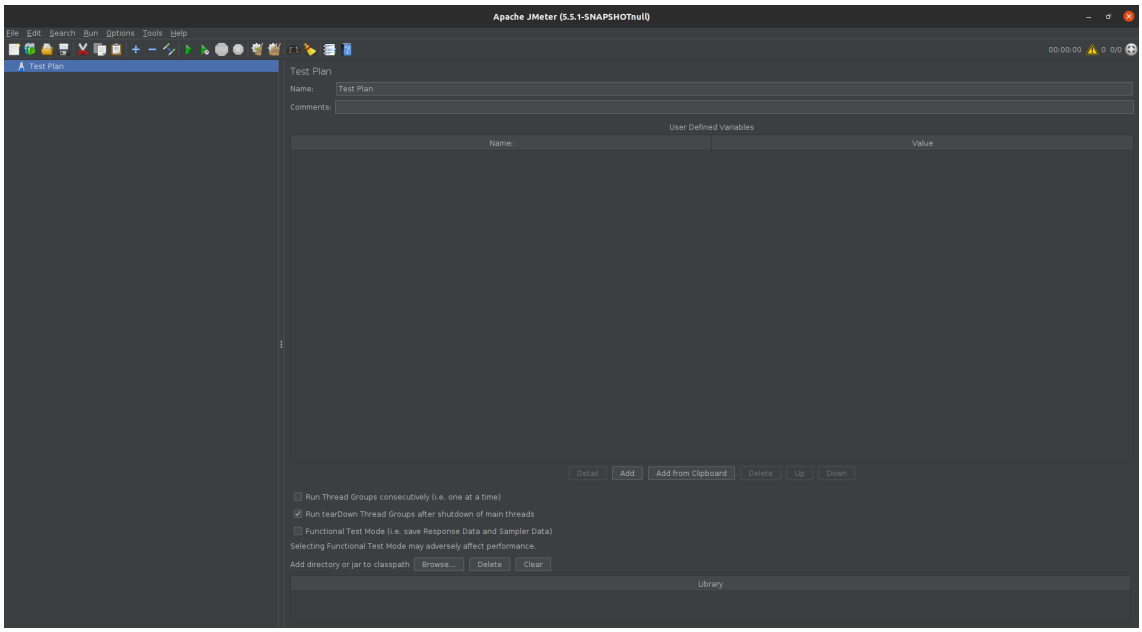


Obr. 1.3: Diagram komunikace přes HTTPS s TLS 1.3

2 Program JMeter

Apache JMeter™ je open source software vyvíjený americkou neziskovou společností The Apache Software Foundation, kde s jeho vývojem začal Stefano Mazzocchi [17]. Původně byl určen k testování výkonu Apache JServ (v současnosti známý jako Apache Tocat Project). Firmou Apache byl později rozšířen o GUI (Graphic User Interface) a další testovací prvky. Celý program je naprogramován v jazyce JAVA [18].

Slouží k testování výkonu webových aplikací, simulací zatížení serverů a sítí, SQL databází a podporuje protokoly JDBC, FTP, SOAP, LDAP, JMS a HTTP [19]. Umožňuje vytvoření specifických testů pro konkrétní aplikace, které simulují reálnou provozní zátěž. Pracuje na protokolové úrovni, neumožňuje tedy renderování webových stránek.



Obr. 2.1: Úvodní obrazovka JMeteru

JMeter po spuštění testu pošle požadavky na cílový server, kdy simuluje skupinu uživatelů, následně sesbírá data pro výpočet statistik a zobrazí výsledky prostřednictvím různých formátů [17].

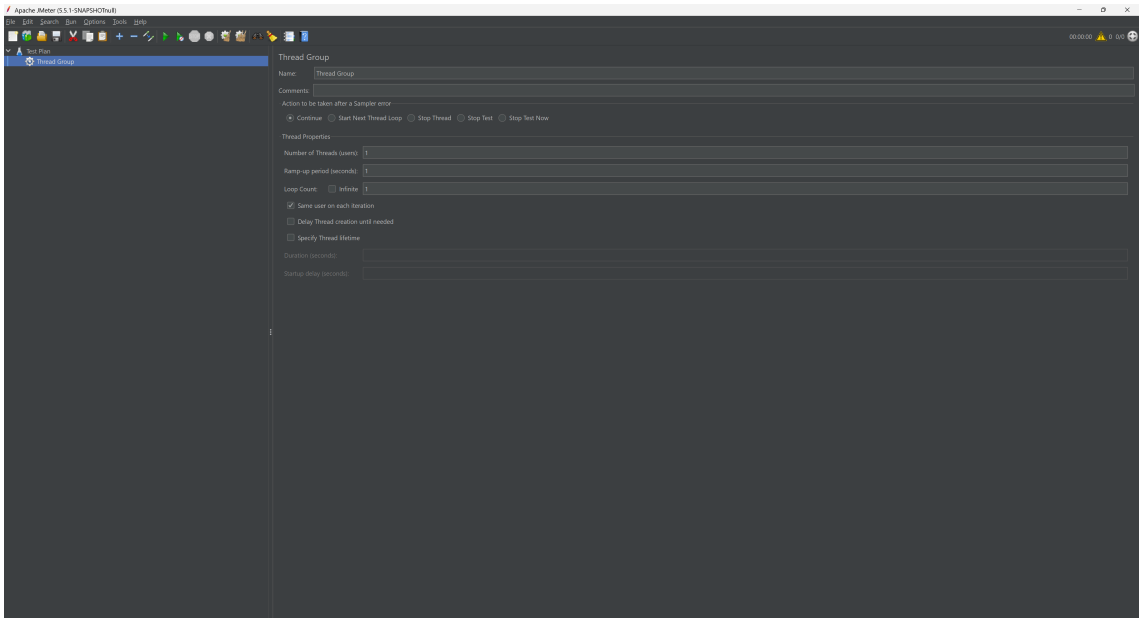
2.1 Typy testů v JMeteru

JMeter umožňuje následující typy testování:

- **Testování výkonu** probíhá obvykle vystavením serveru maximální očekávané zátěži, či jejím překonáním. Účelem tohoto testu je změření různých výkonnostních atributů systému – doba odezvy, spolehlivost, využití zdrojů, škálovatelnost a stabilita za různých podmínek. Provádí se obvykle před uvedením systému do ostrého provozu, jeho účelem je zjištění případných nedostatků.
  - **Simulace zátěže** se snaží napodobit zatížení při připojení mnoha reálných klientů – generuje různě velkou zátěž v průběhu času.
  - **Stres test** na druhou stranu vytváří po celou dobu maximální možnou zátěž.
- **API testing** slouží k testování návratových hodnot vracené API (Application Programming Interface) rozhraním.
- **Security testing** je využíván pro testování zranitelností, jako je spidering (prohledávání webů za účelem sběru dat a monitorování, kdy je cílem získání informací pro boj s konkurencí, či pro škodlivé účely [20]), fuzzing (technika testování softwaru, kdy jsou na vstup zadávány náhodné a neočekávané vstupy) a DDoS.

2.2 DDoS flood v JMeteru

Před vytvořením HTTPS záplavového útoku musíme nejdříve vytvořit vláknovou skupinu (*Thread Group*). Vlákno je prvním prvkem každého testu, všechny další testovací elementy se už musí nacházet ve vytvořeném vláknu.



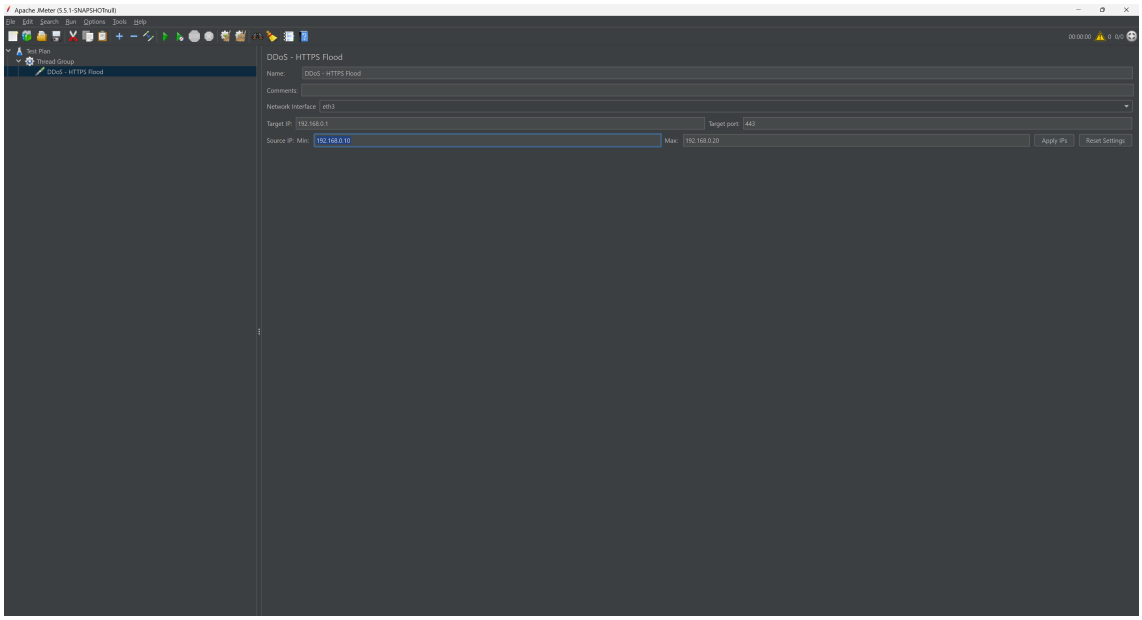
Obr. 2.2: Nastavení vlákna

Ve vláknu je možno nastavit:

- Počet vláken.
- Dobu náběhu – čas, který uplyne, než se test rozběhne na plný počet vláken. V případě 30 vláken a době náběhu 120 sekund jsou vlákna spouštěna po sobě se 4 s zpožděním (120/30 = 4). Je vhodné náběh volit dostatečně dlouhý, aby se zabránilo plnému vytížení hned v začátku testu a zároveň ne příliš krátký, aby některá vlákna neskončila dříve, než budou všechna zprovozněna.
- Počet spuštění – přednastavená hodnota je 1.

Vláknová skupina umožňuje také další nastavení – trvání [s] (duration) a startovací zpoždění [s] (startup delay).

Na obrázku 2.3 je již zobrazen samotný modul HTTPS flood útoku. Umožňuje výběr síťového rozhraní, ze kterého má být generovaný provoz odesílán, IP adresu oběti, číslo portu (které je u HTTPS komunikace standardně na portu 443) a nakonec rozsah IP adres, ze kterých bude útok generován. Právě tuto funkcionalitu má za cíl bakalářská práce zlepšit z ohledu na výkon a doplnit o IPv6.



Obr. 2.3: HTTPS flood modul

3 Záplavový útok HTTP požadavků z více různých zdrojových adres

Pokud je cílem udělat DDoS útok pouze z jednoho stroje, potřebujeme zajistit, aby jím vygenerovaná komunikace byla z více IP adres. IP adresy se používají ve dvou verzích – IPv4 a IPv6.

3.1 IPv4

Internet Protocol version 4 (IPv4) je čtvrtá verze IP protokolu, adresa se skládá ze čtyř bajtů, což teoreticky umožňuje přiřadit unikátní adresu až $2^{4 \cdot 8} = 4\,294\,967\,296$ zařízení [21]. O přidělování těchto adres se stará celosvětová organizace IANA (Internet Assigned Numbers Authority), která je nadřazena pěti organizacím RIR (Regional Internet registry), které se starají o konkrétní území:

- AFRINIC (African Network Information Centre) – Afrika,
- ARIN (American Registry for Internet Numbers) – Antarktida, Kanada, menší část Karibiku, USA,
- APNIC (Asia Pacific Network Information Center) – východní Asie, Oceánie, jižní Asie, jihovýchodní Asie,
- LACNIC (Latin America and Caribbean Network Information Centre) – většina Karibiku, Latinská Amerika,
- RIPE NCC (Réseaux IP Européens Network Coordination Centre) – Evropa, Střední Asie, Rusko, západní Asie [22].

Obvykle je IP adresa klientským zařízením přiřazena DHCP serverem, který konkrétní zařízení identifikuje na základě MAC adresy síťové karty. Adresy může přidělovat dynamicky, či automaticky. Dynamicky alokovaná IP adresa se vyznačuje tím, že je přidělena na předem definovaný čas. Toto řešení je optimální v místech, kde dochází k časté změně zařízení – IP adresy se po uplynutí časové rezervace uvolní a mohou být přiděleny dalším zařízením. A pokud je zařízení i po uplynutí časového limitu připojeno, dojde k prodloužení rezervace adresy (adresa může být přiřazena nová, nebo zůstat původní – záleží na nastavení).

V případě automaticky alokované IP adresy nic jako časový limit není, IP adresa je přidělena na neomezeně dlouhou dobu. Jedinou cestou, jak změnit nebo uvolnit IP adresu, je manuální zasažení do nastavení. Další variantou je všechna klientská zařízení nastavit manuálně – tedy nepotřebujeme DHCP server a administrátor sítě musí zajistit, že adresy nebudou ve vzájemné kolizi.

Z důvodu počátečního suboptimálního hospodaření, počtu zařízení, který překonal původní očekávání a dalším faktorům, byl rozsah IP adres vyčerpán 31. ledna

2014 [23]. Z tohoto důvodu byl zaveden protokol IPv6 (viz kapitola 3.2), přechod na něj se ovšem ukázal jako komplikovaný a pomalý, proto vzniklo krátkodobé řešení standardem RFC 1918 [24].

3.1.1 RFC 1918 a NAT

Standart RFC 1918 určuje tři rozsahy IP adres, které nejsou směrovatelné z internetu a používají se pro lokální sítě. Tyto rozsahy jsou:

- 10.0.0.0 – 10.255.255.255 poskytující $\Rightarrow 2^{3 \cdot 8} = 16\,777\,216$ adres,
- 172.16.0.0 – 172.31.255.255 poskytující $\Rightarrow 2^{4+2 \cdot 8} = 1\,048\,576$ adres,
- 192.168.0.0 – 192.168.255.255 poskytující $\Rightarrow 2^{2 \cdot 8} = 65\,536$ adres [25].

Přechod (překlad) mezi těmito adresami zajišťuje NAT (Network Address Translation). Příchozímu paketu od klienta zamění IP hlavičku – změní IP adresu za svoji, kterou má ve svém rozsahu veřejných adres (obvykle má však k dispozici pouze jednu).[1] Tuto operaci si uloží do překladové tabulky, aby při odpovědi serveru znovu zaměnil IP adresu na původní a poslal ji zpět komunikujícímu zařízení [26].

3.2 IPv6

Pokud není uvedeno jinak, v celé této kapitole je jako zdroj použita elektronická kniha IPv6 [27]. Internet Protocol version 6 (IPv6) je šestá verze IP protokolu, skládá se z šestnácti bajtů, z tohoto důvodu se téměř vždy zapisuje v hexadecimální soustavě, kde jsou každé čtyři znaky (dva bajty) oddělené dvojtečkou.

V současnosti je protokol IPv6 používán pro maximálně čtvrtinu celkového světového provozu,[2] což je daleko za dřívějšími vizemi a plány. Pro zpřístupnění služeb po IPv6 je ovšem veden i politický tlak. Orgány vlády USA měly už do konce roku 2014 nasadit nativní IPv6 ve všech svých sítích, Evropská komise se též snaží o zavedení nové verze protokolu – v roce 2002 zafinancovala několik velkých projektů k používání IPv6, v květnu 2008 vydala akční plán, který požaduje, že při inovacích komunikační infrastruktury bude vyžadována podpora IPv6. Zastoupení nového směrovacího protokolu se i díky těmto tlakům každým rokem zvětšuje, a proto bude pro HTTPS Flood modul využito směrování jak pomocí protokolu IPv4, tak IPv6.

Protokol díky svému rozsahu umožňuje přiřazení až $2^{16 \cdot 8}$, což odpovídá $340\,282\,366\,920\,938\,463\,463\,374\,607\,431\,768\,211\,456 \approx 3,4 \cdot 10^{38}$ unikátních adres, tedy

[1] NAT ve skutečnosti nemění pouze IP adresu, ale též změní číslo portu a z důvodu těchto změn musí přepočítat kontrolní součet v záhlaví (ten totiž zohledňuje i oba tyto parametry), jinak by došlo na cílové stanici k zahození paketu z důvodu nesouladu obsahu s kontrolním součtem.

[2] Zajímavá stránka pro zobrazení procentuálního zastoupení IPv6 v jednotlivých zemích je `https://stats.labs.apnic.net/ipv6`.

při současném počtu osmi miliard obyvatel připadá na jednoho přibližně $4,2 \cdot 10^{28}$ adres,[3] což je stále asi $9,9 \cdot 10^{28}$krát více, než je celkový rozsah IPv4.

Je definováno několik základních skupin adres. Skupinu je možno identifikovat na základě prvních dvou bajtů. Pro generování IPv6 adres pro DDoS útok je tento poznatek důležitý, neboť bez něj by mohlo dojít ke snadnému filtrování na základě zmíněných prvních dvou bajtů IPv6 adresy. Rozsahy adres a jejich význam jsou převzaty ze zdroje [22].

- 0000 až 00ff – nespecifikované, lokální smyčky (v IPv4 127.0.0.0 až 127.255.255.255) a IPv4 kompatibilní adresy,
- 0100 až 01ff – nepřiřazené,
- 0200 až 03ff – speciální adresy,
- 0400 až 1fff – nepřiřazené,
- 2000 až 3fff – agregovatelné globální unicastové adresy,
- 4000 až fbff – nepřiřazené, možné jako globální adresy,
- fc00 až fdff – lokální unikátní adresy (unicast),
- fe00 až fe7f – nepřiřazené,
- fe80 až febf – lokální linkové adresy (unicast),
- fec0 až feff – site local adresy, byly zrušeny,
- ff00 až ffff – skupinové adresy (multicast, zastupující i broadcastovou adresu, která u IPv6 není zavedena samostatně).

Z uvedeného vyplývá, že adresy vhodné pro generování se nachází v rozsazích 2000 až 3fff a 4000 až fbff. Prvních 48 bitů adresy přitom obvykle slouží jako globální směrovací prefix (slouží k identifikaci koncové sítě, je přidělen externě – prvních 16 bitů IANA, dalších 16 přiděluje RIR a nakonec zbývající 2 bajty ISP (Internet Service Provider). Následuje 16 bitů identifikátoru podsítě a nakonec 64 bitů pro identifikaci koncových rozhraní [28]. Při generování adres v HTTPS flood modulu můžeme tedy zvolit, z jakého kontinentu budou adresy pocházet.

Automatické přidělování zbytku IPv6 adresy je pak zajišťováno stavovou konfigurací – DHCPv6 serverem, který funguje principiálně podobně jako DHCPv4, hlavní odlišnost je ovšem v tom, že pro přidělení adresy nepoužívá MAC adresu. Místo toho zavádí DHCP Unique Indentifier (DUID), což je jednoznačný identifikátor zařízení, který se původně neměl změnit ani po výměně síťové karty. Nejpoužívanějším přidělovacím mechanismem[4] DUID je v současnosti generování samotným počítačem pro každé síťové rozhraní, které má být použito pro komunikaci DHCPv6. Algoritmus si následně má vygenerovaný kód uložit do stálé paměti, případně fungovat tak, aby

[3] Při přepočtu na celkový povrch země (včetně vodních ploch) je počet IPv6 adres na $1\,\mathrm{mm}^2$ téměř $7 \cdot 10^{17}$. Pro srovnání – adres IPv4 připadá na $1\,\mathrm{km}^2$ pouze 8.

[4] Autoři protokolu kromě již existujících typů (např. metoda EUI-64, která přenáší linkovou MAC adresu na síťovou) připouští další rozšiřování DUID.

vygenerovaný kód byl vždy stejný.

Druhá varianta přidělování je bezstavově (s touto variantou se počítá ve většinovém nasazení). Tento způsob je z hlediska přidělování zcela nový a vyskytuje se pouze v IPv6 (podrobněji je popsán v RFC 4862). Proces funguje v nejjednodušší podobě následovně: každý směrovač v definovaných intervalech posílá do sítí, k nimž je připojen, takzvané ohlášení směrovače. V této zprávě jsou obsaženy základní informace, zejména prefixy adres sítí a adresa výchozí brány. O tyto informace si též může počítač aktivně požádat.

Z ohlášení směrovačů se dozví, jaké adresy používá síť, k níž je připojen a k němu si připojí svých 64 bitů adresy rozhraní (ta může být vytvářena různými způsoby, některé algoritmy neobsahují prvek náhody – odvozují adresu například z MAC adresy – tedy je vždy stejná – apod.). Následně odesílá broadcast s dotazem, zda danou adresu již někdo nepoužívá. Pokud dostane kladnou odpověď, celý proces se opakuje. V této komunikaci si též vytvoří základ směrovací tabulky – seznam směrovačů, na které bude posílat pakety určené mimo síť.

Vzhledem k počtu možných adres je teoreticky použití NATu v IPv6 zbytečné, nicméně NAT svojí funkcí poskytuje i některé bezpečností benefity[5] (komunikace s lokálním zařízením je problematičtější, chrání soukromí – z venku není možné zjistit nic o struktuře vnitřní sítě ani odhalit, z jakého konkrétního zařízení byl provoz směrem ven vyvolán), a proto se i nyní v některých lokálních IPv6 sítích používá. Zachování tohoto způsobu ochrany je samozřejmě pro koncové uživatele žádoucí, ideálně ale bez NATu a s globálními adresami v koncové síti.

Podobný způsob chování by měl zajistit stavový firewall integrovaný ve vstupním prvku sítě, který propustí do sítě pouze pakety ze zařízení, s nimiž v nedávné době proběhla komunikace z vnitřní sítě. Nevýhodou zůstává, že tímto způsobem nejsme schopni znemožnit odhalení struktury vnitřní sítě. Identitu koncových zařízení ovšem skrývá ohromný rozsah koncové sítě, jehož skenování nástroji typu Nmap, Nikto apod. by bylo extrémně časově náročné. Nadto existuje RFC 4941, který dokáže skrýt koncového uživatele náhodně generovanými dočasnými adresami.

3.2.1 RFC 4941

RFC 4941 – *Privacy Extensions for Stateless Address Autoconfiguration in IPv6* bylo zavedeno z důvodu ochrany soukromí uživatele. V IPv6, kdy koncová síť není schována za NATem, by mohlo docházet ke sledování zařízení na základě jeho IP adresy, a to i napříč internetem (jak bylo zmíněno v kapitole 3.2, posledních 64 bitů adresy slouží k identifikaci rozhraní a nemusí se měnit, právě na základě této části

[5]NAT sice není bezpečností prvek a útoku na koncová zařízení nedokáže zamezit, jeho obejití ovšem vyžaduje dalšího úsilí, lokální počítače tedy alespoň částečně chrání.

IPv6 adresy by bylo možné zařízení sledovat). Ochrana funguje tak, že počítač si pro komunikaci s vnějším světem vygeneruje novou náhodnou IPv6 adresu, pod kterou komunikuje. Mnoho zařízení funguje zároveň jako server, tak i jako klient, proto má síťové rozhraní počítače jeden pevný identifikátor (např. pro registraci v DNS serveru) a pro komunikaci jako klient si vytváří anonymizované adresy.[6] Tuto adresu si stroj po definovaném čase vygeneruje novou a starou si ponechá po nastavenou dobu pro případ zpožděné příchozí komunikace.

I v dokumentu RFC 4941 se v sedmé kapitole řeší možné zneužití pro DDoS útoky. K ochraně se doporučuje implementování *ingress filtering* – router ISP kontroluje, zda zdrojová IP adresa spadá do rozsahu jím obsluhovaným, nedokáže však vyřešit změny v části adresy identifikující rozhraní. K řešení tohoto problému se doporučuje použití řízení přístupu na základě jednotlivých adres na výstupním bodu sítě [29].

3.3 Shrnutí generování adres pro DDoS útok

3.3.1 IPv4

Pokud se se znalostmi z kapitoly 3.1 zamyslíme nad variantami podvrhování IP adres, naskytují se následující možnosti:

- s DHCP serverem
  - měnění MAC adresy síťové karty a přiřazení nové IP adresy DHCP serverem,
  - vytvoření virtuálních síťových karet (IP aliasing) [30],
- bez DHCP serveru[7]
  - měnění IP adresy zařízení po odeslání každého požadavku (IP spoofing) [31],
  - přepisování IP adres v odesílaném paketu
    - na síťové kartě,
    - proxy serverem.

Všechny tyto varianty budou samozřejmě omezeny tím, že nemohou přejít přes NAT poskytovatele připojení, všechny tyto útoky by poté pro oběť pocházely z jedné IP adresy a celý provoz by mohl být jednoduše odstřihnutelný.

V případě, kdy musíme využít DHCP serveru, vzniká automaticky nevýhoda, kdy i při optimálním fungování budeme limitováni rozsahem sítě. První zmíněnou

[6] Generování probíhá za pomocí algoritmu MD5, což je relativně rychlý hašovací algoritmus, nicméně ani tak nebudeme schopni dosáhnout takové rychlosti, jako u změn IP adres z předem daného rozsahu.

[7] Veškeré tyto způsoby jsou použitelné i v případě sítě s DHCP serverem.

možnost (měnění MAC adresy) můžeme zavrhnout z důvodů prodlev, než DHCP server adresu přiřadí, a zatížení linky, kde bude zatížena "přebytečným" provozem způsobenou komunikací s DHCP serverem. Tento problém by šlo teoreticky odstranit využitím dedikované linky pro přiřazování IP adres, ovšem zde už potřebujeme stroj s alespoň dvěma síťovými rozhraními a dodatečnou konfigurací, která by se neobešla bez zásahu do DHCP serveru (standardně by server přiřadil každému síťovému rozhraní jinou IP adresu a problému s omezením šířky pásma linky bychom se nezbavili).

IP aliasing oproti tomu žádné speciální konfigurace mimo samotný útočící stroj nevyžaduje. Potencionální omezení je, že linuxová jádra 2.2 a starší měla omezení na 255 rozhraní, novější by měla mít tento počet neomezený [32].

Pokud budeme měnit IP adresy přímo na rozhraní, situace se stane nejjednodušší. Tento způsob nebude vytěžovat linku nadbytečnou komunikací, samotná změna IP adresy je výpočetně nenáročná, tudíž většina výpočetní kapacity může být využita ke generování provozu. Tato metoda je na funkčním příkladu popsána v kapitole 3.4.

3.3.2 IPv6

V případě IPv6 se naskytují následující možnosti:

- Automatická stavová konfigurace – DHCPv6:
  - vytvoření virtuálních síťových karet,
  - opakované generování různých DUID,
  - generování dočasných adres.
- Automatická bezstavová konfigurace:
  - generování 64 bitů pro identifikaci rozhraní,
  - generování dočasných adres.
- Manuální konfigurace:
  - měnění IP adresy po každém odeslání,
  - generování dočasných adres.

Co se týče dočasných adres, problematika je hlubší, než bylo nastíněno v tomto textu. Z tohoto důvodu si autor neodvažuje říct, nakolik by bylo toto hypotetické řešení proveditelné a jaké by mělo případně omezení. Generování dočasných adres přes MD5 hašovací algoritmus nebudeme ovšem schopni provádět tak rychle, jako způsoby zmíněnými výše.

Možnosti zbylých řešení a jejich výhody/nevýhody jsou v zásadě podobné jako u IPv4 s tím rozdílem, že u IPv6 se nepočítá s větším nasazováním DHCPv6. Z tohoto důvodu bude modul naprogramován s použitím manuální konfigurace. Pro generování budou použity znalosti o rozsazích a přidělování adres v kapitole 3.2.

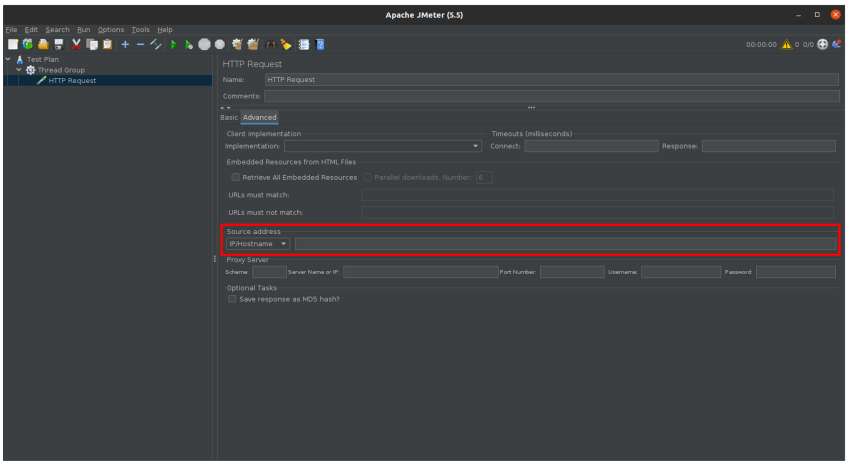
Navíc je pravděpodobné, že tento způsob bude použitelný i v sítích s automatickou bezstavovou i stavovou konfigurací.

3.4 HTTP Flood v JMeteru odesílaný z více IPv4 adres za pomoci IP spoofingu

IP spoofing je využíván k simulování požadavků na webovou aplikaci. Podstatou této metody je zasáhnutí cílového serveru více virtuálními uživateli – toho je dosaženo požadavky z několika IP adres najednou z podstatně menšího množství generujících zařízení.

Tento způsob je efektivní nejen z důvodu lehkého filtrování provozu, ale též pro servery, které jsou za load balancerem[8] – obsluha jedné IP adresy by byla přidělena jednomu serveru. Nemohli bychom tedy například otestovat, jak efektivně je zátěž rozdělena.

JMeter má IP spoofing již využívaný v modulu HTTP Request, kde je na uživatelském vstupu možno vybrat zdrojové adresy [33]. Pro jeho využití je ovšem nutné



Obr. 3.1: HTTP modul a možnosti IP spoofingu

nejprve v operačním systému IP adresy přidat. U operačního systému Windows toto můžeme provést jak přes GUI (Graphic User Interface), tak přes příkazový řádek.

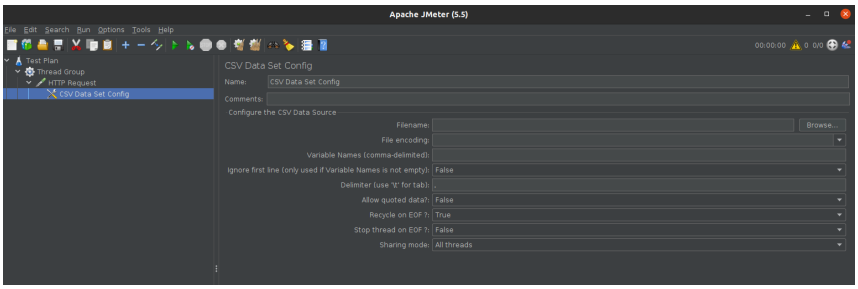
V případě GUI postupujeme následovně: Ovládací panely →Síť a Internet →Centrum síťových připojení a sdílení →Změnit nastavení adaptéru →Vybrání síťového

[8]Zařízení, které slouží k rozložení zátěže mezi více serverů, procesorů, síťových rozhraní apod. za účelem dosažení optimálního využití hardwarových zdrojů, prostupnosti a latence. Zároveň zvyšuje míru robustnosti systému, neboť v případě zhavarování jednoho serveru dokáže zátěž rozdělit na zbývající. K vyvažování zátěže obvykle slouží stroj (počítač, switch) se speciálním softwarem uzpůsobeným pro tuto činnost [34].

rozhraní, vlastnosti →IPv4, vlastnosti →Použít následující IP adresu, upřesnit →Přidání jednotlivých IP adres a masek sítě. Tímto postupem jsme schopni teoreticky nakonfigurovat až 2^{32} adres.

Tento postup je ovšem značně uživatelsky nepřívětivý, zaměříme se proto na možnost přes příkazový řádek. Toho můžeme dosáhnout například napsáním následujícího skriptu: `FOR /L %i IN (2,1,60) DO netsh interface ipv4 add address "enp3s0"192.168.0.%i 255.255.255.0`. Příkaz `FOR` vytvoří cyklus, parametr `/L` cyklu řekne, že v uvedeném příkladu má procházet čísla od 2 do 60 s krokem 1. Jednotlivé iterace jsou uloženy do proměnné `i`, která je dosazována do uvedené IP adresy a v každé iteraci je proveden příkaz pro přidání nové adresy na síťové rozhraní `enp3s0`.

Všechny vytvořené IP adresy nahrajeme i do csv souboru – odtud je JMeter načítá. Toho docílíme přidáním modulu *CSV Data Set Config* za *HTTP Request* (nachází se v nabídce *add →config element*). Do pole *filename* vložíme cestu k vytvo-



Obr. 3.2: CSV Data Set Config modul

řenému CSV souboru a pojmenujeme proměnnou (*Variable Names*), kterou následně vložíme do HTTP Request modulu do pole *Source address* ve tvaru ${*proměnná*}. Do modulu Thead Group nakonec zadáme počet vláken – v případě zadání stejného počtu vláken jako vygenerovaných IP adres bude při každé iteraci přistoupeno z každé IP adresy na server právě jednou.

4 Příprava testovacího prostředí

Pro testování rychlostí a generování komunikace z více IP adres byl postaven server s 10 Gb síťovou kartou. Tato rychlost by měla být dostatečná pro budoucí testy naprogramovaného modulu.

4.1 Využitý hardware a operační systémy

Jako základ serveru slouží procesor Intel Core i7-4770 osazený do základní desky ASUS H87M-Pro s 16 GB operační paměti DDR3. Do základní desky byla přidána 10 Gb síťová karta TP-Link TX 401. Jako operační systém byl zvolen Ubuntu Server 22.04.1 LTS.

Pro spojení serveru s klientským zařízením byl použit switch Mikrotik CRS326-24G-2S+RM, který má porty pro dva SFP+ moduly.[1] Tyto porty byly osazeny ethernetovými S+RJ10 moduly stejného výrobce.[2] Na switchi běží operační systém RouterOS 6.49.6 v režimu bridge.

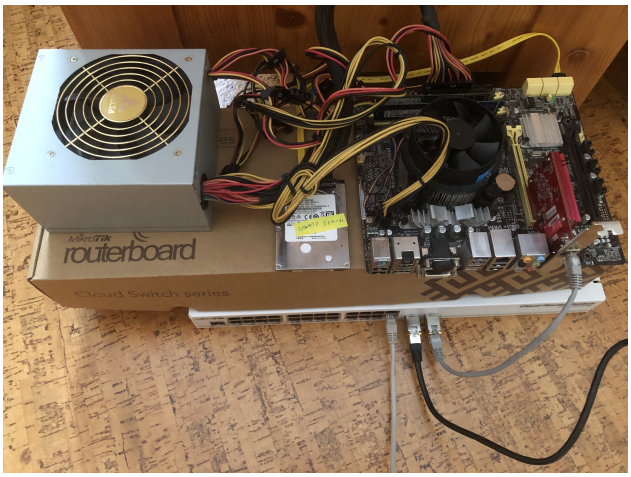
Klientský počítač, který bude používán ke generování provozu, disponuje procesorem AMD Ryzen 9 3900X v základní desce GIGABYTE X570 Aorus Pro, 32 GB operační pamětí DDR4, grafickou kartou RTX 2080 a 10 Gb síťovou kartou Zyxel XGN100C. Na stroji bude pro účely testu nainstalován OS Ubuntu 20.04.6 LTS.

4.2 Konfigurace síťových karet

Po nainstalování operačního systému je třeba upravit konfigurační soubor pod právy super uživatele (root), `/etc/netplan/*.yaml` (do verze Ubuntu 18.04. LTS se soubor nalézá v `/etc/network/interfaces/*.yaml`), kde původní síťovou kartu nastavíme na volitelnou (`optional: true`), čímž výrazně urychlíme spouštění systému [35], jelikož systém nebude kartu tak dlouho vyhledávat a doplníme soubor o rozhraní nové síťové karty, kde parametr `dhcp4` rovněž nastavíme na `true`. Jméno rozhraní zjistíme například příkazem `ip addr`. Tím docílíme, že karta bude při příštím zapnutí systému aktivována.

[1] Autor si je vědom, že zařízení mohla být propojena bez mezilehlého zařízení, ovšem vzhledem k prostorové disponibilitě a občasné potřebě mít obě zařízení připojená do sítě internet, nebylo toto řešení vyhodnoceno za optimální.

[2] V průběhu prvních testů rychlosti bylo zjištěno, že spojení není stabilní, ale pravidelně dochází k 10 minutovým výpadkům. Tyto výpadky byly zapříčiněny přehříváním SPF+ modulů, které dosáhly 95 °C a z bezpečnostních důvodů docházelo k jejich automatickému vypnutí. Tento konstrukční nedostatek byl odstraněn nainstalováním hliníkového profilu a jeho připevněním pomocí teplovodivých podložek na SPF+ porty.



Obr. 4.1: Testovací server se switchem Mikrotik

Výslednou podobu můžeme vidět na obrázku 4.2. Po uložení souboru načteme upravený soubor (`sudo netplan apply`) a následně příkazem `reboot now` restartujeme systém.

```
  GNU nano 6.2
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp3s0:
      dhcp4: true
      optional: true
    enp1s0:
      dhcp4: true
  version: 2
```

Obr. 4.2: Konfigurace síťových karet

4.3 Výběr a konfigurace webového serveru

V současnosti jsou nejpoužívanější webové servery Nginx a Apache. Jejich konkrétní zastoupení se dle různých zdrojů liší, jejich kumulativní zastoupení je ovšem přibližně 50 %. Třetí nejpoužívanější je server Cloudflare s přibližně 10 % [36].

HTTPS flood útok bude tedy otestován na obou zmíněných serverech. Před samotnou instalací obou serverů je vhodné aktualizovat balíčky (`sudo apt update`).

4.3.1 Apache2

Server Apache2 nainstalujeme příkazem (`sudo apt install apache2`) a příkazem `sudo ufw allow "Apache Full"` povolíme ve firewallu port 80 (pro HTTP) a 443

(pro HTTPS)[3]. V prohlížeči zadáme do URL řádku IP adresu serveru a následně se zobrazí výchozí stránka Apache. Pokud chceme stránku zobrazit pomocí IPv6 adresy, **musíme** ji vložit do hranatých závorek. Kompletní URL adresa pak vypadá `http://[<IPv6 adresa>]`.

Dalším krokem je získání certifikátu, kterým se bude server prokazovat pro navázání zabezpečené komunikace. Standardně si o certifikát zažádá autorita, která je pověřena jejím vydáváním, můžeme jmenovat například Let's Encrypt, Comodo, Digicert, GoDaddy, Symantec, CloudFlare. Právě poslední jmenovaná společnost nabízí certifikáty zdarma [38].

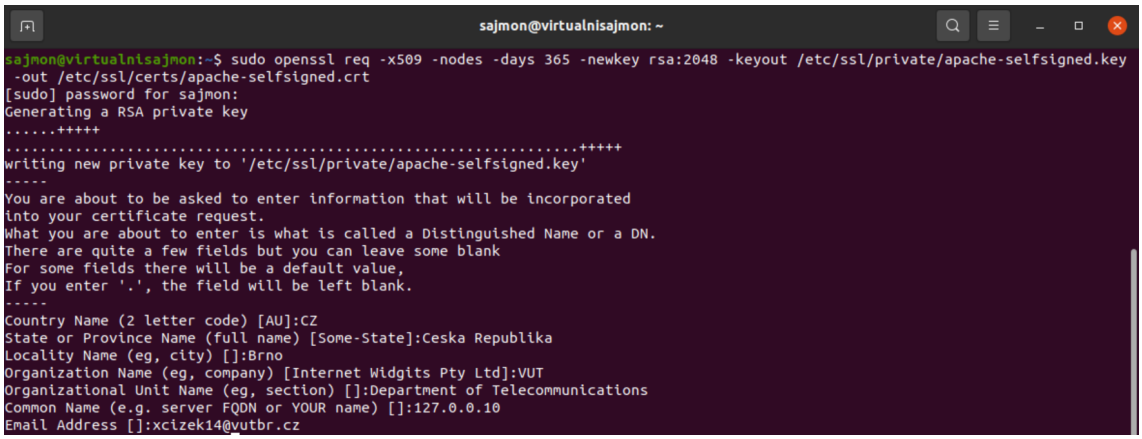
Další variantou je vytvořit námi podepsaný certifikát. Nejdříve povolíme SSL modul (`sudo a2enmod ssl`), pro zavedení změn restartujeme Apache (`sudo systemctl restart apache2`). Samotný certifikát vygenerujeme následujícím příkazem:
`sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048`
`-keyout /etc/ssl/private/apache-selfsigned.key`
`-out /etc/ssl/certs/apache-selfsigned.crt`.
Příkaz `openssl` je nástroj pro tvorbu a správu OpenSSL certifikátů, klíčů a dalších souvisejících souborů, použili jsme přitom následující parametry:

- `req -x509` – určuje, že chceme použít certifikát tohoto standardu, který podporuje TLS,
- `nodes` – certifikát nebude vyžadovat heslo (to by jinak server vyžadoval po každém restartu),
- `days 365` – specifikuje délku platnosti certifikátu, zvolíme maximální délku jednoho roku, neboť prohlížeče delší dobu platnosti odmítají,
- `newkey rsa:2048` – tímto parametrem zadáme vygenerování nového certifikátu a klíče najednou, bude použita asymetrická šifra RSA s 2048 bitů dlouhým klíčem,
- `keyout` – cílové umístění vygenerovaného klíče,
- `out` – cílové umístění certifikátu.

Po potvrzení budeme vyzváni k zadání dalších parametrů, jako je písmenná zkratka země a její celý název, město, jméno organizace, jméno oddělení, e-mailové adresy. Zvláštní pozornost věnujme řádku `Common name (eg, your name or your server's hostname) []`, zde zadáme doménové jméno nebo IP adresu webové služby. Pokud bychom toto neučinili, většina běžných prohlížečů by zobrazila varovnou hlášku, že se `common name` neshoduje s webovou adresou [37].

Nyní musíme vytvořit konfigurační soubor, kam umístíme informace, jaký port se má používat, IP adresu, pro kterou bude certifikát poskytován, a cestu ke klíči a certifikátu – `sudo nano /etc/apache2/sites-available/<doména_nebo_ip>.conf`.

[3]Pokud bychom chtěli povolit port jen pro jeden z protokolů, nahradíme jím `Full`, např. `sudo ufw allow "Apache HTTPS"`.



Obr. 4.3: Tvorba certifikátu

Do tohoto souboru vložíme následující konfiguraci:

```
<VirtualHost *:443>
    ServerName <doména_nebo_ip>
    DocumentRoot /var/www/<doména_nebo_ip>

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/
                                    apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/
                                    apache-selfsigned.key
</VirtualHost>
```

Vytvoříme adresář ve složce www `sudo mkdir /var/www/<doména_nebo_ip>` a v tomto nově vytvořeném adresáři vyrobíme HTML soubor `index.html` (`sudo nano /var/www/<doména_nebo_ip>/index.html`), do kterého vložíme naši webovou stránku. Pro testovací účely stačí krátký kód `<h1>my secured website</h1>`[4].

Po vytvoření a uložení souboru potřebujeme zpřístupnit konfigurační soubor nástrojem `a2ensite` – `sudo a2ensite <doména_nebo_ip>.conf`, pro jistotu provedená nastavení otestujeme příkazem `sudo apache2ctl configtest`. Pokud jsme postupovali správně, výsledek by měl vypadat takto:

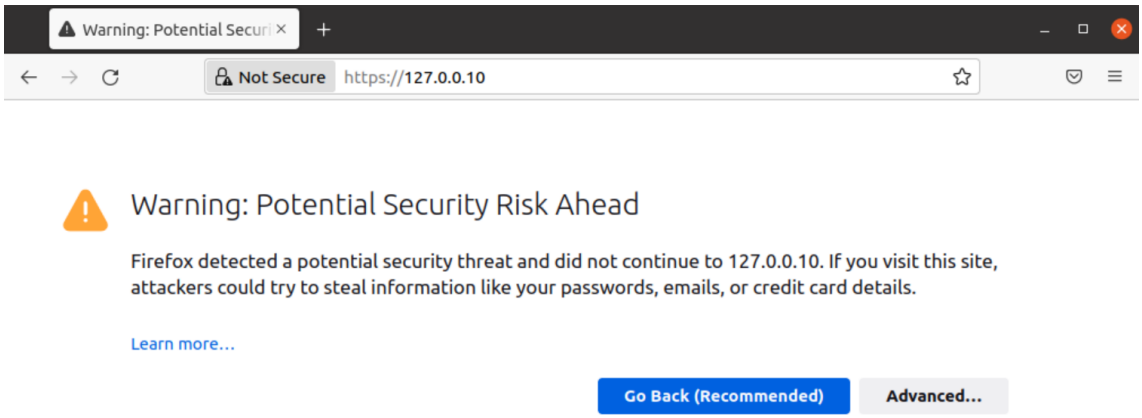
```
Output
AH00558: apache2: Could not reliably determine
the servers fully qualified domain name, using
127.0.1.1. Set the 'ServerName' directive globally
```

[4]Nejedná se kompletní HTML soubor, ale současné prohlížeče s tím nemají problém.

```
to suppress this message
Syntax OK
```

V případě, kdy test nevykazuje chyby, můžeme restartovat Apache (`sudo systemctl reload apache2`) [39]. Nyní se můžeme pokusit navštívit naši stránku přes HTTPS protokol, z tohoto důvodu před adresu serveru napíšeme `https://`, čímž si https protokol vynutíme.

Načtená stránka bude vypadat ve všech prohlížečích podobně, zobrazí se nám upozornění, že stránka může obsahovat bezpečnostní riziko. Prohlížeč nezná auto-



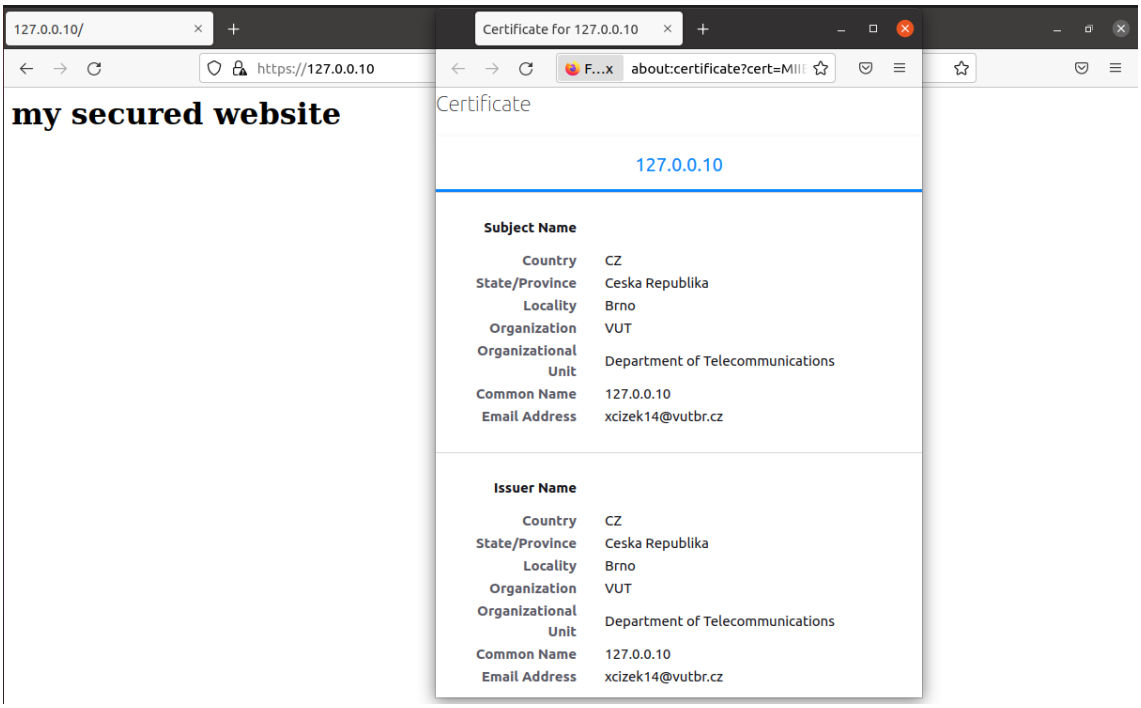
Obr. 4.4: Zobrazení neznámého certifikátu prohlížečem

ritu, která podepsala vystavený certifikát, z toho důvodu označil webovou stránku jako nedůvěryhodnou. Zvolíme možnost pokročilé (advanced), následně potvrdíme, že přijímáme riziko a chceme pokračovat. Načte se nám naše stránka, která komunikuje přes HTTPS protokol.

Po kliknutí na ikonu zámku si můžeme zobrazit námi vytvořený certifikát, viz obrázek 4.5. V případě, že bychom chtěli, aby prohlížeč považoval náš certifikát za důvěryhodný, museli bychom ho přidat v operačním systému mezi důvěryhodné[5], toto zdokonalení ovšem pro naše účely nebude vyžadováno.

Po prvotním testování bylo zjištěno, že Apache server má v základu pouze 150 klientů, po dosažení tohoto počtu začne veškerá další spojení odmítat, pro testy musíme tedy tento počet zvýšit. Nejprve musíme nakonfigurovat MPM (Multi-Processing Modules) modul, který umožní obsluhovat více souběžných spojení. V Ubuntu nalezneme tři druhy MPM modulů, které postupně přicházely s vývojem Apache serveru. Tyto moduly jsou:

[5] *Obvykle* stačí jednou potvrdit certifikát v prohlížeči.



Obr. 4.5: Webová stránka načtená přes HTTPS protokol

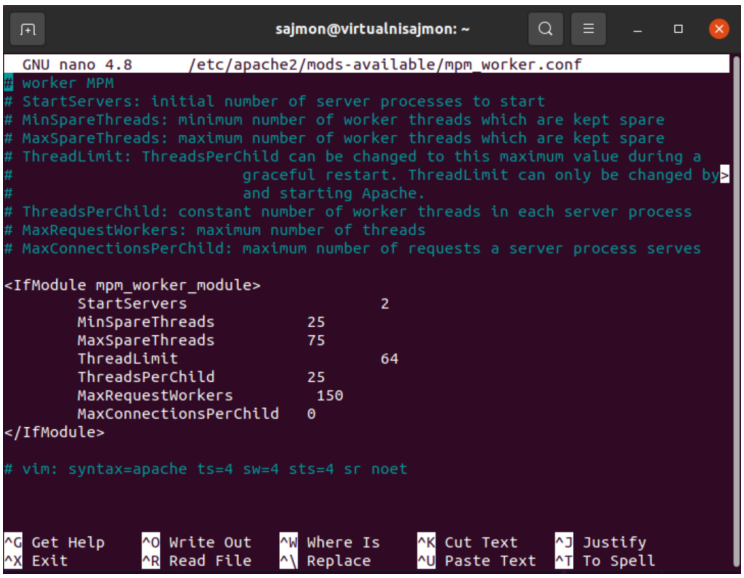
- `mpm_prefork` – nejkompatibilnější, není ideální pro souběžné zpracování požadavků, je paměťově nejnáročnější [40],
- `mpm_worker` – mnohem lépe obsluhuje souběžné požadavky, je méně paměťově náročný,
- `mpm_event` – stejné benefity jako `mpm_worker`, nejnovější, podporuje Apache 2.4 a novější [41].

Pro testy využijeme poslední zmíněný MPM modul. Povolíme ho příkazem `sudo a2enmod mpm_event`, OS Ubuntu by měl mít už tento modul povolen. Pokud tomu tak je, v terminálu se nám zobrazí následující upozornění:

```
Considering conflict mpm_worker for mpm_event:
Considering conflict mpm_prefork for mpm_event:
Module mpm_event already enabled.
```

V případě, že bychom potřebovali libovolný MPM modul zakázat, použijeme příkaz `sudo a2dismod mpm_*`, kde * je jméno konkrétního modulu.

Po spuštění modulu otevřeme konfigurační soubor příkazem `sudo vi /etc/apache2/mods-available/mpm_worker.conf`. Výchozí nastavení vypadá jako na obrázku 4.6. Zde stačí navýšit parametr `MaxRequestWorkers`. Po těchto krocích bychom měli mít server připravený pro testování útoků.



Obr. 4.6: Výchozí MPM konfigurace

4.3.2 Nginx

Jestliže nám na stroji běží už jiný webový server, je vhodné ho nyní zastavit z důvodu kolize portů po nainstalování Nginx – `sudo systemctl stop <jméno serveru>` a následně příkazem `sudo systemctl disable apache2` zamezíme jeho automatickému startu po restartu systému. Nginx server nainstalujeme příkazem `sudo apt install nginx` a obdobně jako u Apache2 povolíme ve firewallu porty 80 a 443 `sudo ufw allow 'Nginx Full'`.

Nginx by se měl po instalaci automaticky spustit, to můžeme ověřit příkazem `systemctl status nginx`. A následně budeme schopni po zadání IP adresy serveru do URL řádku zobrazit výchozí webovou stránku Nginx (pokud jsme ve firewallu povolili i port 80), jejíž zdrojový kód je uložen v `/usr/share/nginx/html`.

Dalším krokem je zprovoznění zabezpečeného přenosu. Po vytvoření certifikátu, jak je popsáno v kapitole 4.3.1, upravíme konfigurační soubor `sudo nano /etc/nginx/sites-available/default`, kde odstraníme část pro HTTP komunikaci a nahradíme ji, jak je popsáno na výpise 4.1.

Výpis 4.1: Konfigurační soubor Nginx pro HTTPS

```
server {
        listen 443 ssl default_server;
        listen [::]:443 ssl default_server;
        ssl_certificate /etc/nginx/certificate/
                                nginx-certificate.crt;
        ssl_certificate_key /etc/nginx/certificate/
```

```
                                nginx.key;
        root /var/www/html;
        index index.html index.htm
                             index.nginx-debian.html;
        server_name _;
        location / {
                try_files $uri $uri/ =404;
            }
    }
```

Po uložení Nginx server restartujeme – `service nginx restart` a můžeme načíst vytvořenou stránku přes HTTPS protokol.

5 Vývoj modulu

Při vývoji modulu bylo vyzkoušeno několik nadějně vypadajících způsobů pro DDoS z jednoho síťového rozhraní. Jednou ze zkoušených možností bylo odeslání ze staticky zadané IP adresy, následně její smazání a nahrazení novou. Toto řešení bylo ovšem problematické z důvodu vícevláknového testu, protože na síťové kartě mohla být v jeden okamžik pouze jedna IP adresa. Další zkoušenou variantou bylo vytváření paketu s již vepsanou podvrhnutou IP adresou.

5.1 Vytváření paketu s podvrhnutou IP adresou

Java samotná neumožňuje nízkoúrovňové operace jako je tvorba/úprava paketů. Z toho důvodu byly pokusy o provedení IP spoofingu pomocí open source knihovny Pcap4j, která je dostupná pod licencí MIT. Je napsána v programovacím jazyce C. Její instalaci na OS Ubuntu provedeme příkazem `sudo apt-get install libpcap-dev` a následně musíme přidat do konfiguračního souboru námi používaného nástroje pro sestavení programu Gradle tyto řádky:

```
compile 'org.pcap4j:pcap4j-core:1.+'
compile 'org.pcap4j:pcap4j-packetfactory-static:1.+'.
```

Nyní je knihovna připravena k použití. Cílem bylo vytvořit TCP paket, do něj vložit námi předem definovanou IP adresu a nad ním vybudovat TLS a HTTP komunikaci. TCP paket s podvrženou IP adresou vytvoří následující kód:

```
1  PcapNetworkInterface nif = null;
2  try {
3      nif = Pcaps.getDevByName("enp7s0"); //name of NIC
4      } catch (PcapNativeException ex) {
5      throw new RuntimeException(ex);
6      }
7
8  // Open the interface with a X-second timeout
9  int timeout = 2;
10 PcapHandle handle = null;
11 try {
12     handle = nif.openLive(65536,
13         PcapNetworkInterface.PromiscuousMode.PROMISCUOUS,
14         timeout);
15     } catch (PcapNativeException ex) {
16     throw new RuntimeException(ex);
17     }
```

```
18
19 InetAddress srcAddr = null;
20 try {
21     srcAddr = Inet4Address.getByName("192.168.1.109");
22     } catch (UnknownHostException ex) {
23     throw new RuntimeException(ex);
24     }
25 InetAddress dstAddr = null;
26 try {
27     dstAddr = Inet4Address.getByName("192.168.1.113");
28     } catch (UnknownHostException ex) {
29     throw new RuntimeException(ex);
30     }
31 TcpPort srcPort = TcpPort.getInstance((short) 12345);
32 TcpPort dstPort = TcpPort.HTTP;
33 TcpPacket.Builder tcpBuilder = new TcpPacket.Builder();
34 tcpBuilder.srcPort(srcPort)
35     .dstPort(dstPort)
36     .sequenceNumber(1)
37     .acknowledgmentNumber(1)
38     .ack(true)
39     .window((short) 4096)
40     .build();
41
42 List<AbstractPacket> packets = new ArrayList<>();
43 packets.add(tcpBuilder.build());
44 IpV4Packet.Builder ipBuilder = new IpV4Packet.Builder();
45 ipBuilder.version(IpVersion.IPV4)
46     .tos(IpV4Rfc791Tos.newInstance((byte) 0))
47     .identification((short)100)
48     .ttl((byte)100)
49     .protocol(IpNumber.TCP)
50     .srcAddr((Inet4Address) srcAddr)
51     .dstAddr((Inet4Address) dstAddr)
52     .payloadBuilder(
53         new AbstractPacket.AbstractBuilder() {
54         @Override
55         public Packet build() {
56             IpPacket.Builder builder =
57                 new IpV4Packet.Builder();
58                 for (AbstractPacket packet : packets) {
```

```
59                          builder.payloadBuilder()
60                              packet.getBuilder());
61                      }
62              return builder.build();
63              }
64          });
65 IpV4Packet ipPacket = ipBuilder.build();
66 // Send the packet
67 try {
68     handle.sendPacket(ipPacket);
69     } catch (PcapNativeException ex) {
70     throw new RuntimeException(ex);
71     } catch (NotOpenException ex) {
72     throw new RuntimeException(ex);
73     }
74 System.out.println("Packet_sent_successfully");
75 handle.close();
76 }
```

Nicméně testy funkčnost tohoto řešení neprokázaly. Problém bude nejspíše v navazování komunikace přes TCP protokol v třícestném handshaku, kdy z podvržené adresy dojde k odeslání TCP s příznakem SYN, server odpoví SYN-ACK, ovšem tato odpověď už k útočícímu zařízení nedorazí, jelikož router nemá tuto adresu ve směrovací tabulce.[1] Tímto způsobem tedy dosáhneme pouze útoku známého pod jménem SYN flood.

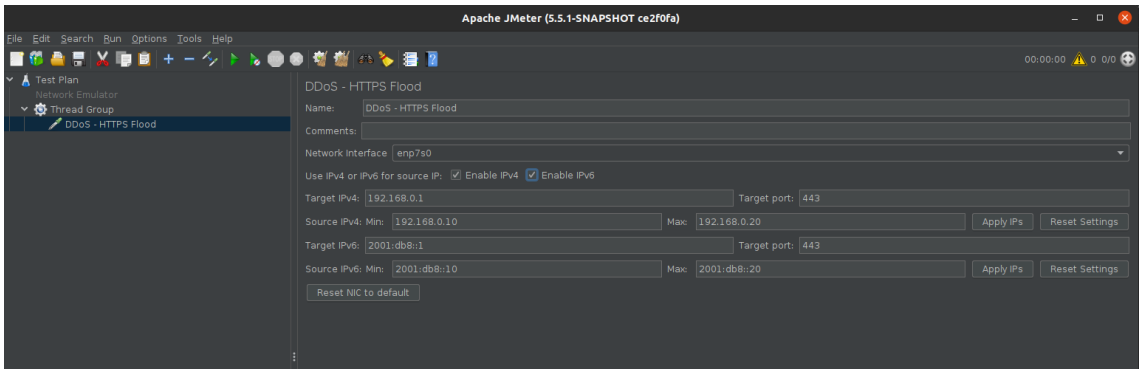
Možnou cestou, jak tímto způsobem dojít až k HTTPS záplavovému útoku, by bylo HTTP paket provozovat na UDP (který nemá inicializační komunikaci), nicméně tím bychom problém pouze odsunuli k TLS komunikaci. Druhou cestou by bylo dostat naše vygenerované IP adresy do směrovací tabulky, nicméně proti tomuto zahlcování jsou i základně nastavené routery odolné. Z těchto důvodů bylo pokračování tohoto řešení ukončeno.

5.2 Přidání všech adres na síťové rozhraní

Konečným vytvořeným řešením se stalo přidání všech IP adres na síťové rozhraní a následně odesílání paketů z jedné náhodně vybrané. Grafické rozhraní bylo upraveno, jak vidíme na obrázku 5.1.

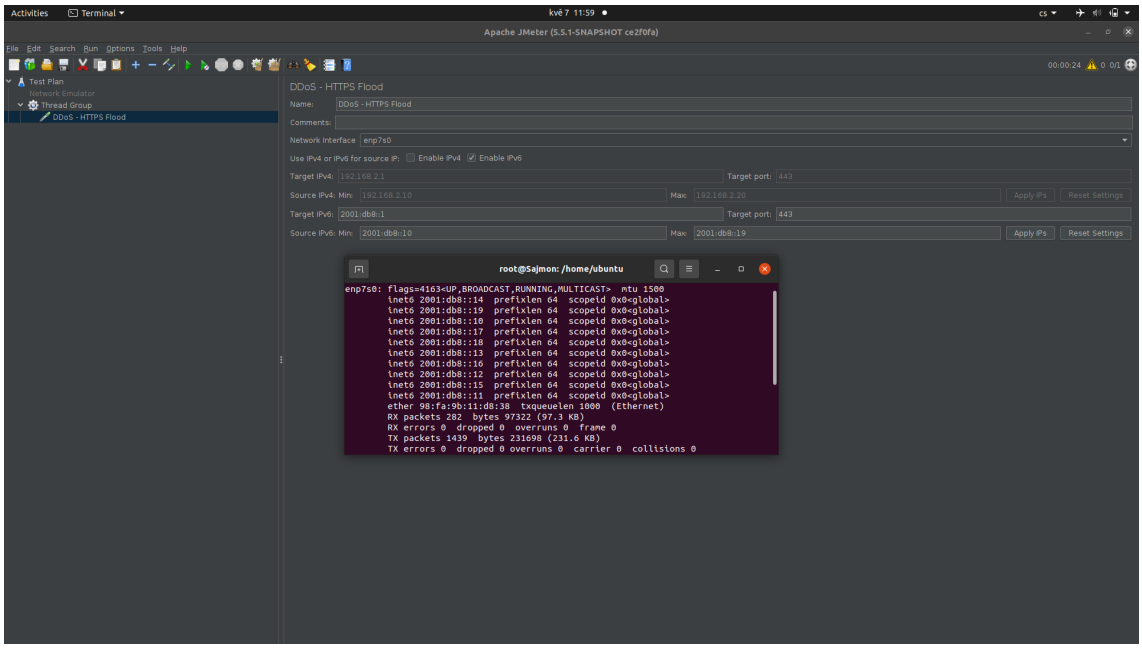
Po kliknutí na tlačítko *Apply* jsou vygenerovány rozsahy IPv4/IPv6 adres ze zadaného intervalu a následně všechny příkazem `ip addr add <IPv4 adresa>/24`

[1] Případně počítač na své síťové kartě.



Obr. 5.1: HTTPs flood modul rozšířený o IPv6

`dev <jméno rozhraní>` nebo `ip addr add <IPv6 adresa> /64 dev <jméno rozhraní>` přidány.[2] Přidané IPv6 adresy jsou znázorněny na otisku obrazovky 5.2.[3] Je též vytvořen soubor pro následné smazání ve složce `/tmp`



Obr. 5.2: Přidané IPv6 adresy na rozhraní

`/deleteIp<4/6>Addresses.sh`.

Tlačítko *Reset* u IP adres na druhou stranu spustí odstraňovací skript vytvořený výše a odstraní programem přidané adresy – původně nastavené zůstávají zachovány, například z důvodu komunikace se serverem přes SSH a podobně. Pro potřebu

[2] Povšimněme si, že útok je vždy vykonáván ze sítě s prefixem 24 u IPv4 a s prefixem 64 u IPv6. Toto nastavení nelze uživatelsky změnit.

[3] Pokud chceme případně zkontrolovat přidané IPv4 adresy, musíme využít příkaz `ip addr` z důvodu jiného chování protokolu – tyto adresy jsou nastaveny jako sekundární.

odstranit a resetovat veškeré adresy na síťovém rozhraní bylo přidáno tlačítko *Reset NIC to default*, který odstraní IPv4 i IPv6 adresy, resetuje rozhraní a vyžádá si novou adresu od DHCP serveru.

Pro HTTPS komunikaci byl vybrán požadavek GET z důvodu jeho nejuniverzálnějšího použití. Rovněž byla provedena výkonnostní optimalizace, v původním modulu například docházelo ke generování rozsahů IP adresy při každé iteraci testu. Nově generování probíhá pouze jednou – po kliknutí na tlačítko *Apply* a následně je z této množiny při každé iteraci testu jedna IP adresa náhodně vybrána.

6 Instalace a provedení simulovaného útoku

K instalaci postačí z elektronické přílohy v systému VUT zkopírovat soubor `Ddos.jar` do nainstalovaného JMeteru, konkrétně do složky `/jmeter-main/jmeter/lib/ext`. Modul byl otestován na operačním systému Ubuntu 20.04.6 LTS, Apache Jmeter verze 5.5.1 a jazykem JAVA 17.0.7 (tu po aktualizaci balíčků – `sudo apt update` – nainstalujeme příkazem `sudo apt install openjdk-17-jdk openjdk-17-jre`).

Zařízení s JMetrem připojíme přímo do síťové karty webového serveru podporujícího HTTPS komunikaci, případně alespoň do stejné sítě. Zařízení **nesmí** být od serveru odděleno routerem. Zjistíme IP adresu serveru a můžeme začít s nastavením testu. JMeter je nutno spustit pod právy super uživatele (*root*) – tato práva jsou potřeba z důvodu měnění adres na síťovém rozhraní. Toho lze dosáhnout například tím, že v terminálu ve složce `/jmeter-main/jmeter/bin` zadáme příkaz `sudo sh jmeter.sh`. Pro provedení HTTPS flood útoku můžeme do testovacího plánu (*Test plan*) přidat následující otestované možnosti:

- *Thread Group* – základní test, plný výkon se spustí ihned, případně s malým zpožděním.
- *setUP Thread Group* – prvně jsou vytvořena vlákna a až pak proběhne test, stejný jako Thread Group.
- *Stairs Thread Group* – test schodovitě zvyšuje počet připojených uživatelů (vláken), vhodný v případě přítomnosti základních ochran proti DDoS – útok nezačne najednou, ale pomalu roste.
- *Complex Thread Group* – podobný *Stairs Thread Group*, dává nám ovšem plnou kontrolu nad časy spuštění, doby trvání a času zastavení jednotlivých vláken.

Ve vzorovém příkladu budeme používat *Thread Group*, ostatní ovšem fungují podobně. V položce *Number of Threads (users)* nastavíme, kolik HTTPS požadavků má být provedeno v jedné iteraci. *Ramp-up period* označuje dobu náběhu, než dojde k plnému testovacímu výkonu, kterou můžeme ponechat na výchozí hodnotě jedna sekunda. Položka *Loop Count* zajistí, kolikrát má být jednotlivá vlna útoků provedena (v případě, že bude zvolen jeden thread a jeden cyklus, bude uskutečněn přístup pouze z jedné IP adresy). Vše je též znázorněno v kapitole 2.2.

Po tomto nastavení přidáme do námi vytvořeného *Thread group* modul *DDoS – HTTPS flood*, jejž nalezneme v položce *Sampler*. V samotném HTTPS flood modulu zvolíme fyzické rozhraní počítače, ze kterého má být test uskutečněn. Vybereme, zda chceme útok vést přes IPv4 či IPv6 adresu, zadáme cílovou adresu, port a rozsah zdrojových adres. Poté zadané údaje aplikujeme tlačítkem *Apply* (pokud jsou pro útok vybrány oba směrovací protokoly, **musíme** potvrdit změny tlačítkem u každého protokolu jednotlivě). Útok následně můžeme spustit.

Před samotným spuštěním testu byl na cílovém serveru zapnut program TShark (obdoba WireSharku, dokáže ovšem běžet pouze v příkazovém řádku) s filtrem na port 443.[1] Po zapnutí útoku jsou ihned vidět přístupy z různých IPv6 adres. Konkrétně na obrázku 6.1 vidíme přístupy na server[2] přes adresy 2001:db8::1c, 2001:db8::17 a 2001:db8::13. Obdobně přes IPv4 a rovněž v případě spuštění testu přes obě verze směrovacích protokolů.

Na výpisech z TSharku si můžeme všimnout toho, že první TLS komunikace je ve verzi 1.2, což je v rozporu s tím, co bylo uvedeno v kapitole 1.1. V této kapitole byl rozebrán případ, kdy je nasazována **pouze** jedna verze protokolu. V reálném světě jsou ovšem stále servery, které běží na verzi 1.2 a v případě odeslání klientského požadavku TLS 1.3 by server tento protokol neznal a nemohl TLS handshake uskutečnit. Z tohoto důvodu je nejdříve komunikace navazována přes verzi 1.2, se kterou je verze 1.3 kompatibilní [42].

Jak samotný útok[3] vytěžuje testovaný server Nginx, je znázorněno na obrázku 6.5. Pro porovnání je na obrázku 6.4 znázorněn stav bez zatížení. Rychlost přenášených dat činí přibližně pouhých 70 mb/s při testu serveru Nginx.

Se stejným nastavením vidíme vytíženost serveru Apache2 na obrázku 6.6. V porovnání s Nginx zátěž rovnoměrněji rozkládá na všechna procesorová vlákna. Zároveň má už ve výchozím nastavení aktivovány některé obrany proti DDoS – jak vidíme na obrázku 6.8.

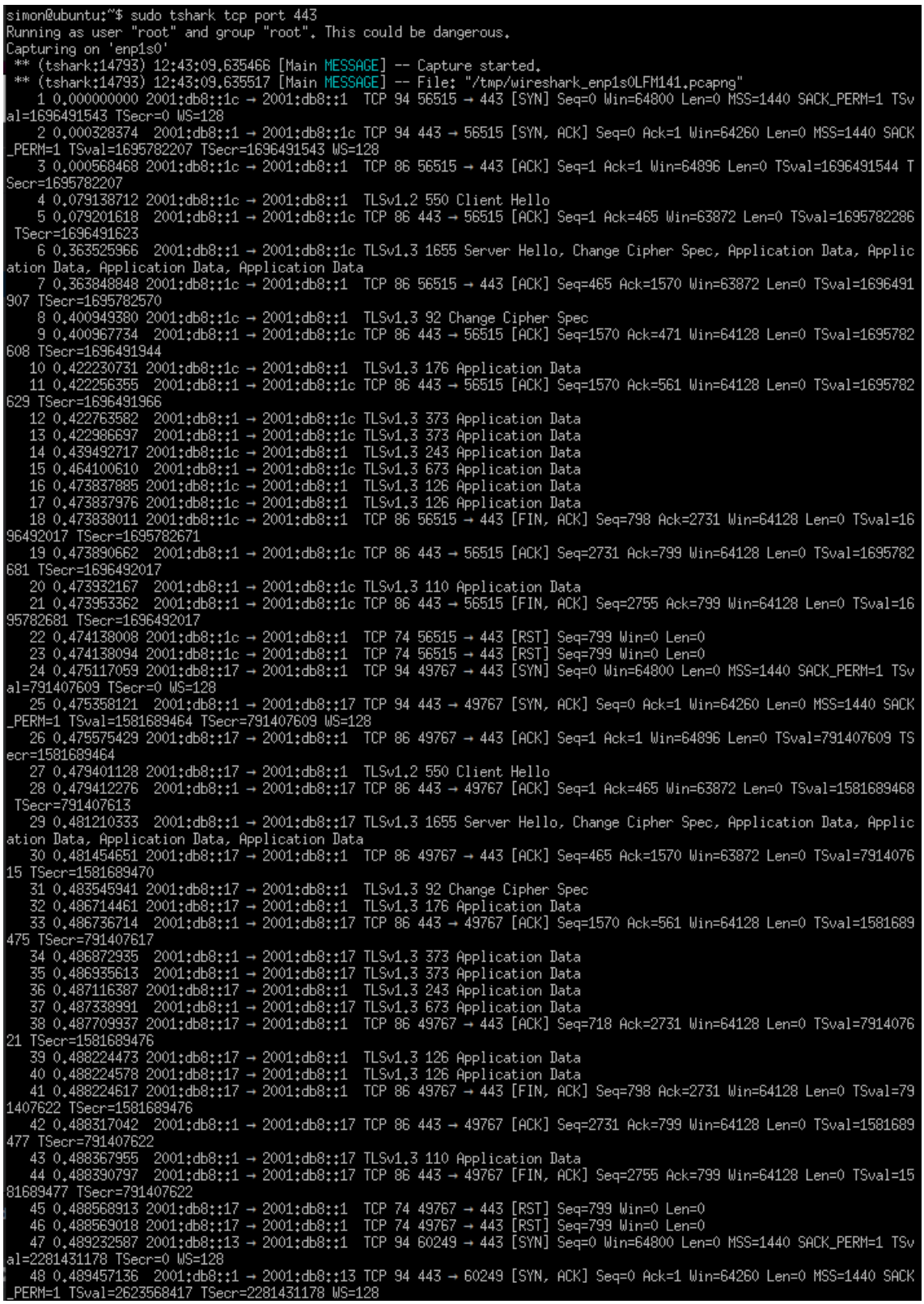
Graf je pro názornost rozdělen do třech částí. V první vidíme klidový provoz na síťovém rozhraní. V druhé části dochází k prvním vlnám útoku, kdy server během několika sekund zareaguje na probíhající útok a začne některou komunikaci odmítat, jak vidíme na poklesu ve třetí části grafu. Nginx žádné takové ochrany ve výchozím stavu nemá, útok proto probíhá se stejným vytížením celou dobu, viz obrázek 6.7.

V případě snahy vytížit procesor webového serveru Nginx ze 100 % stačil přístup přibližně 80 uživatelů (vláken) v jedné iteraci testu (vytížení tohoto testu je znázorněno na obrázku 6.9). Server Apache2 se díky zabudovaným DDoS ochranám ze 100 % vytížit nepodařilo. Přibližně od 65 uživatelů nebyl ve vytížení serveru pozorován žádný rozdíl. Generovaný provoz byl přibližně 150 Mb/s.

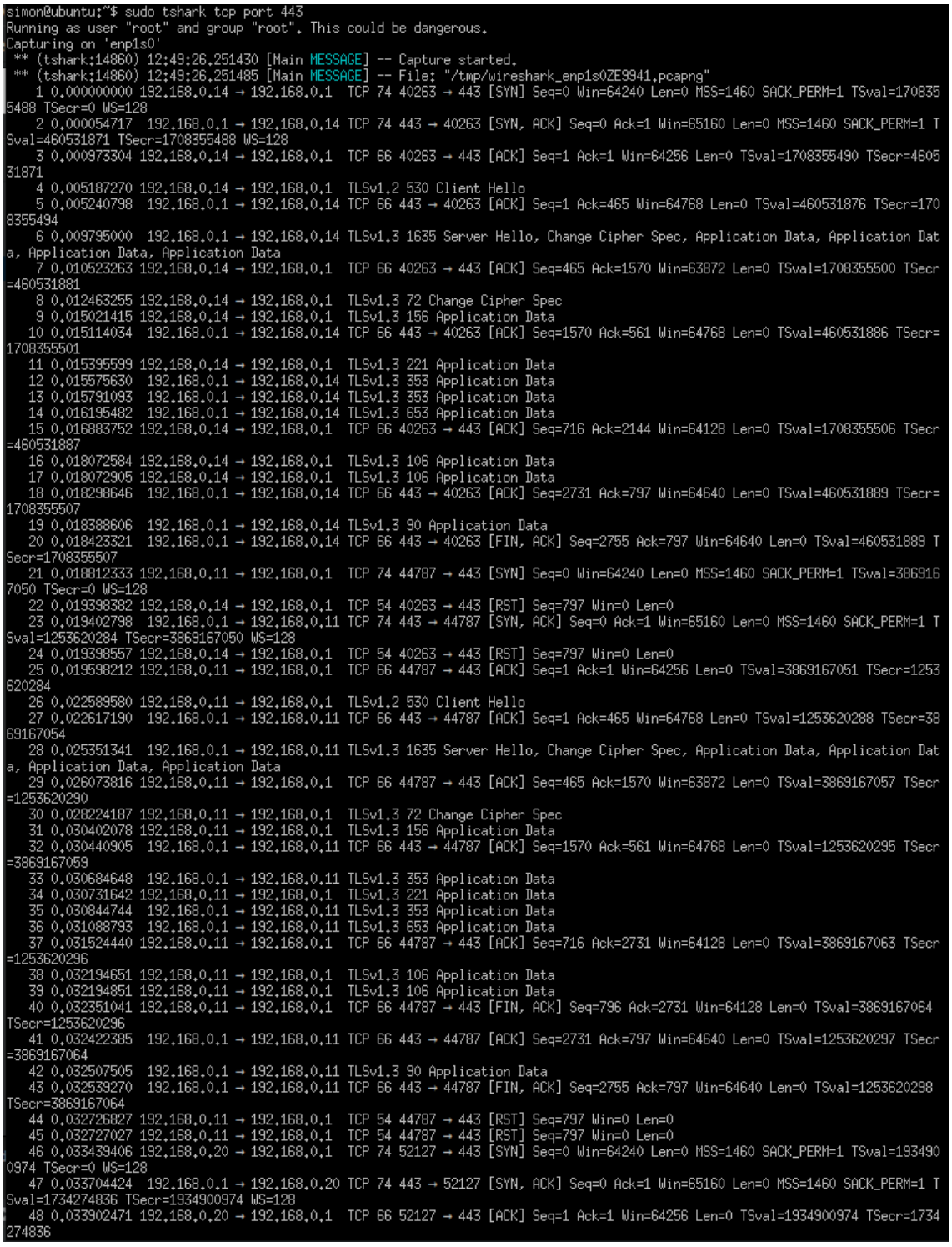
[1] Instalaci je možno provést příkazem `sudo apt install tshark`, pro zapnutí filtru na HTTPS komunikaci slouží příkaz `sudo tshark tcp port 443`.

[2] Server má adresy 2001:db8::1 a 192.168.0.1.

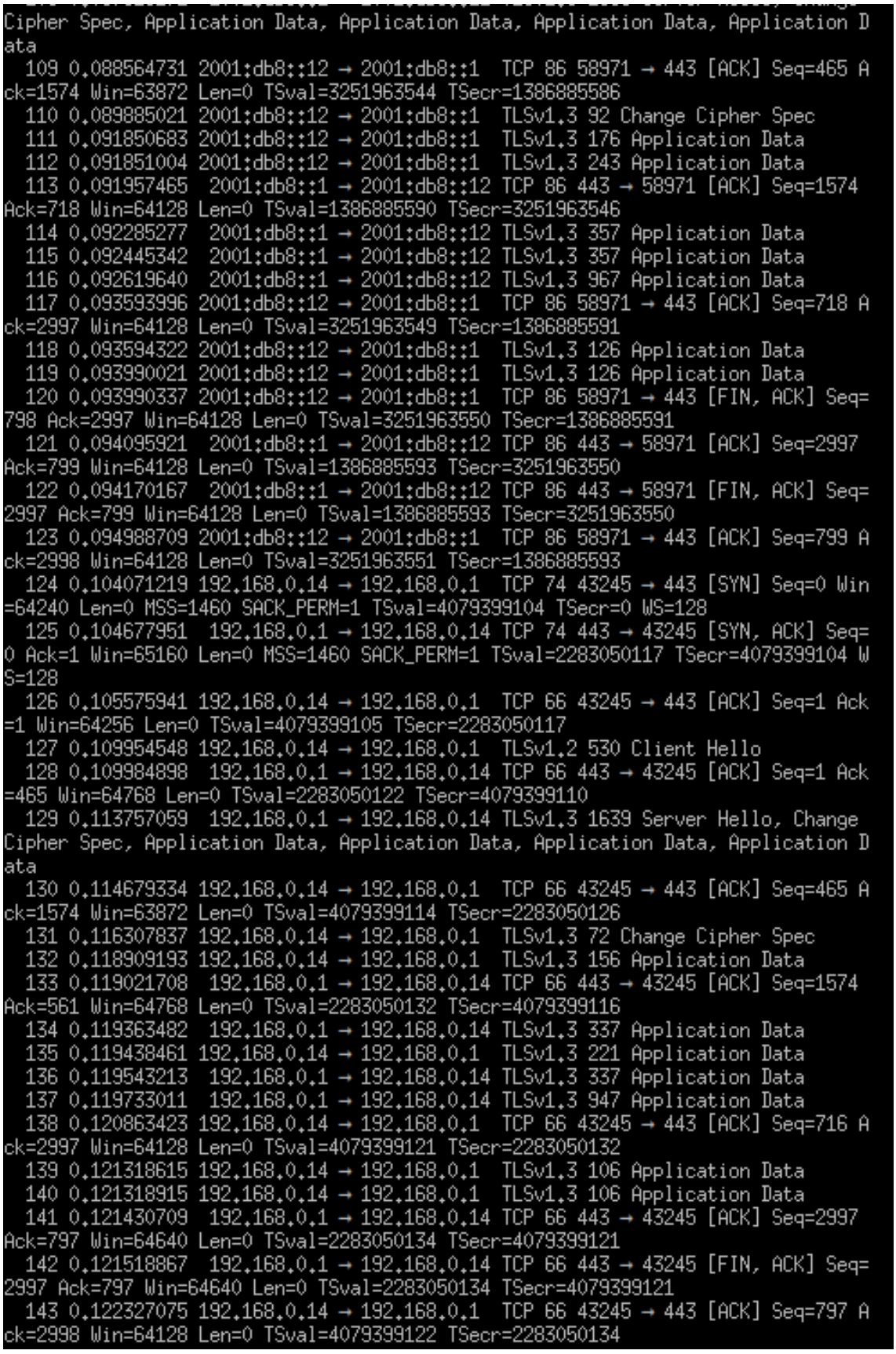
[3] Při konfiguraci 15 vláken a nekonečném cyklu.



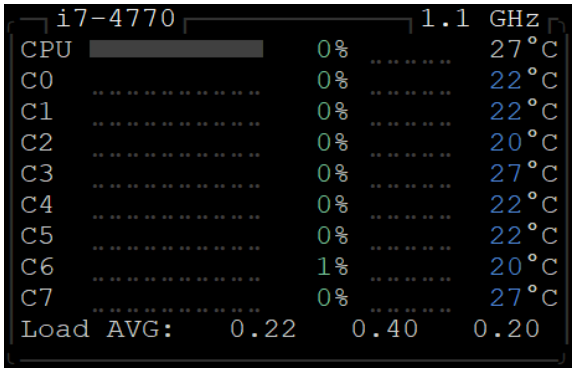
Obr. 6.1: Úspěšný HTTPS flood útok přes IPv6



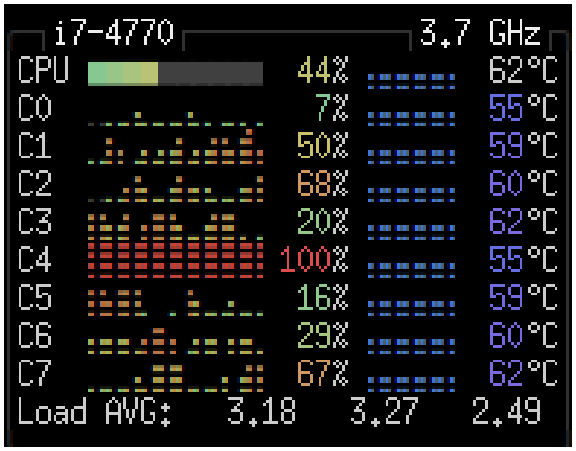
Obr. 6.2: Úspěšný HTTPS flood útok přes IPv4



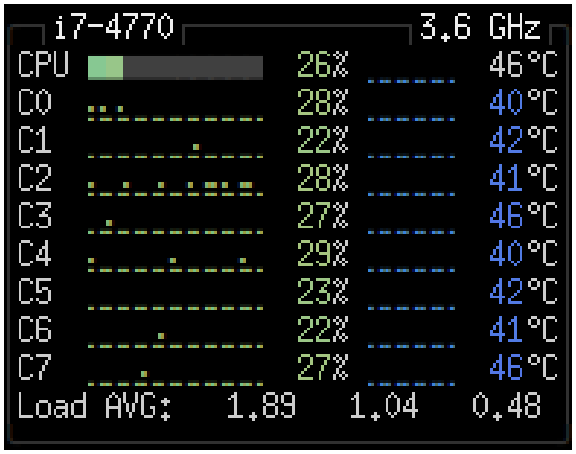
Obr. 6.3: Úspěšný současný HTTPS flood útok přes IPv4 a IPv6



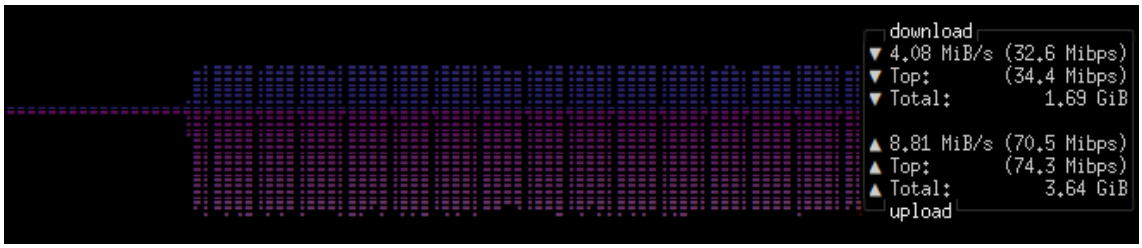
Obr. 6.4: Vytížení procesoru serveru Nginx při nečinnosti



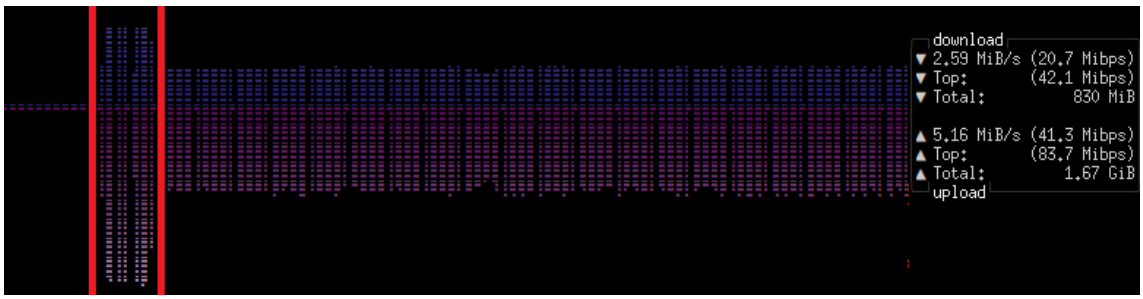
Obr. 6.5: Vytížení procesoru serveru Nginx při HTTPS flood útoku, 15 uživatelů, nekonečný útok



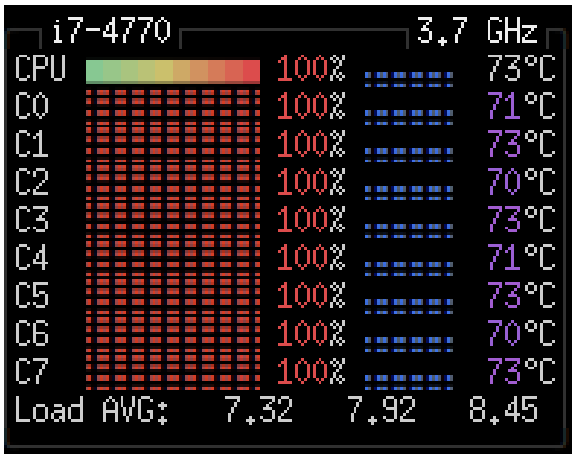
Obr. 6.6: Vytížení procesoru serveru Apache2 při HTTPS flood útoku, 15 uživatelů, nekonečný útok



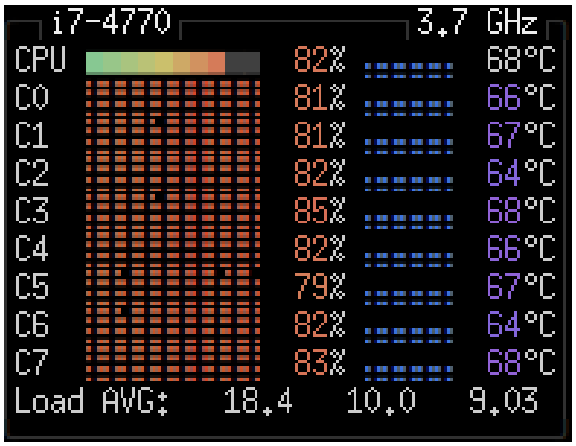
Obr. 6.7: Vytížení síťového rozhraní serveru Nginx při HTTPS flood útoku, 15 uživatelů, nekonečný útok



Obr. 6.8: Vytížení síťového rozhraní serveru Apache2 při HTTPS flood útoku, 15 uživatelů, nekonečný útok



Obr. 6.9: Vytížení procesoru serveru Nginx při HTTPS flood útoku, 80 uživatelů, nekonečný útok



Obr. 6.10: Vytížení procesoru serveru Apache2 při HTTPS flood útoku, 80 uživatelů, nekonečný útok

Závěr

V rámci práce došlo k seznámení s historií DDoS útoků a jejich stále častějším výskytem, účelem, principem a základním dělením. Byl popsán princip komunikace přes HTTP, HTTPS s TLS 1.2 a TLS 1.3. Dále byla vysvětlena stavba a použití IPv4 a IPv6 protokolů. Na všechny tyto protokoly bylo nahlédnuto z pohledu jejich využití pro generování zátěže na webový server z více IP adres.

Byly vysvětleny základní funkcionality softwaru Apache JMeter a jeho využití pro zátěžové testování. Popsány byly původní HTTPS flood a HTTP flood moduly a následně byly probrány možnosti generování komunikace z více IP adres. Pro finální Apache modul byla použita metoda IP spoofingu, kdy jsou všechny adresy nejprve přidány na síťové rozhraní. Z těchto adres je následně náhodně vybrána jedna, z které je generován provoz. Tímto způsobem byl naprogramován funkční HTTPS flood modul v jazyce JAVA s několika Bash skripty, umožnující útok přes IPv4 i IPv6 protokol. Zdrojové kódy tohoto modulu jsou spolu se zkompilovaným souborem `Ddos.jar` v elektronické příloze této bakalářské práce.

Pro testy naprogramovaného HTTPS flood modulu byl vytvořen server, na kterém byly útoky analyzovány. V rámci jeho konfigurace byla provedena základní nastavení Apache2 a Nginx serveru, umožňující komunikaci přes HTTPS protokol.

Pro tento modul byly navrhnuty prototypové testy. Pomocí vybrané metody *Thread Group* byl následně otestován dopad na funkční webové servery Apache2 a Nginx. *Thread Group* byl zvolen z důvodu jeho konstantní generované zátěže, díky které jsme dosáhli přehledného porovnání dopadů na oba webové servery.

Přístupy na server z více IP adres byly analyzovány programem TShark. Data ohledně vytížení procesoru a síťové karty byla následně znázorněna a porovnána pro oba webové servery. Z naměřených dat vyplývá, že Apache2 se už ve výchozím nastavení dokáže lépe bránit DDoS útokům (viz obrázek 6.8), kdy už po pár iteracích testu začne odmítat některou komunikaci. Provoz obsluhovaný serverem Nginx byl oproti tomu konstantní po celou dobu testu a v porovnání s Apache2 se rychleji vyčerpal jednojádrový výkon.

Při pokusu vytížit zcela výpočetní kapacity postaveného serveru s procesorem Intel Core i7-4770 stačilo v případě Nginx 80 uživatelů. Apache2 se ze 100 % vytížit nepodařilo z důvodu zabudovaných DDoS ochran.

Literatura

[1] LAW, Marcus. DDoS protection market to grow amid increase in attacks. *Cyber Magazine* [online]. 2023, **2023** [cit. 2023-05-20]. Dostupné z: https://cybermagazine.com/articles/ddos-protection-market-to-grow-amid-increase-in-attacks

[2] Cloudflare DDoS threat report for 2022 Q4. *CloudFlare* [online]. 2023 [cit. 2023-05-20]. Dostupné z: https://blog.cloudflare.com/ddos-threat-report-2022-q4/

[3] MafiaBoy, the hacker who took down the Internet. *Blackhat ethical hacking* [online]. 2022, 28. 2. 2022 [cit. 2022-12-06]. Dostupné z: https://www.blackhatethicalhacking.com/articles/hacking-stories/mafiaboy-the-hacker-who-took-down-the-internet/

[4] COHEN, Gary. DDoS attacks are born in the Big Ten. *Industrial Cybersecurity Pulse* [online]. 2022, 20. 5. 2022 [cit. 2022-12-07]. Dostupné z: https://www.industrialcybersecuritypulse.com/threats-vulnerabilities/throwback-attack-ddos-attacks-are-born-in-the-big-ten/

[5] DDoS attacks in Q3 2022. *SecureList by Kaspersky* [online]. 2022 [cit. 2022-11-29]. Dostupné z: https://securelist.com/ddos-report-q3-2022/107860/

[6] Co je DDoS útok. *ESET* [online]. c1992-2022 [cit. 2022-11-08]. Dostupné z: https://www.eset.com/cz/ddos-utok/

[7] *How Google block the largest DDoS attack* [online]. 2022 [cit. 2022-11-08]. Dostupné z: https://cloud.google.com/blog/products/identity-security/how-google-cloud-blocked-largest-layer-7-ddos-attack-at-46-\-million-rps

[8] *Advances in Parralel, Distributed computing* [online]. Heidelberg, Německo: Department of ECE, Periyar Maniammai University, Thanjavur, 2011 [cit. 2022-11-28]. ISBN 978-3-540-32296-2. Dostupné z: https://link.springer.com/content/pdf/10.1007/978-3-642-24037-9.pdf

[9] What is SSL, TLS and HTTPS?. *Digicert* [online]. [cit. 2022-11-06]. Dostupné z: https://www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https

[10] What is HTTPS. *SSL* [online]. 2021 [cit. 2022-11-28]. Dostupné z: https://www.ssl.com/faqs/what-is-https/

[11] *Strict-Transport-Security* [online]. c1998-2022 [cit. 2022-11-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

[12] Google on 307/HSTS Redirects. *Search engine journal* [online]. 2020 [cit. 2022-11-28]. Dostupné z: https://www.searchenginejournal.com/google-on-307-hsts-redirects-http-to-https/386232/

[13] Strict-Transport-Security. *MDN web docs* [online]. 2022 [cit. 2022-11-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

[14] Difference between using HSTS and doing a 301 redirect. *UpGuard* [online]. 2022 [cit. 2022-11-28]. Dostupné z: https://help.upguard.com/en/articles/4581202-what-s-the-difference-between-using-hsts-and-doing-a-\-301-redirect

[15] HTTPS Flood. *MazeBolt* [online]. [cit. 2022-11-29]. Dostupné z: https://kb.mazebolt.com/knowledgebase/https-flood/

[16] HTTP Flood. *Imperva* [online]. c2022 [cit. 2022-11-29]. Dostupné z: https://www.imperva.com/learn/ddos/http-flood/

[17] JavaTpoint. *JMeter Tutorial* [online]. JavaTpoint, c2011-2021 [cit. 2022-11-14]. Dostupné z: https://www.javatpoint.com/jmeter-tutorial

[18] The Apache Software Foundation. *Apache JMeter* [online]. Apache Software Foundation, c1996-2022 [cit. 2022-11-06]. Dostupné z: https://www.apache.org/

[19] *Testování softwaru: JMeter* [online]. [cit. 2022-11-14]. Dostupné z: http://testovanisoftwaru.cz/automatizovane-testovani/jmeter/

[20] What is a web crawler? | How web spiders work. *CloudFlare* [online]. [cit. 2023-01-23]. Dostupné z: https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/

[21] *IPv4 Address Report* [online]. 2022 [cit. 2022-12-02]. Dostupné z: https://www.potaroo.net/tools/ipv4/index.html

[22] JEŘÁBEK, Jan. *Pokročilé komunikační techniky*. Brno, 2022. Skripta. Vysoké učení technické v Brně.

[23] Free Pool of IPv4 Address Space Depleted. *NRO* [online]. 2011, 3. 2. 2011 [cit. 2022-12-12]. Dostupné z: https://www.nro.net/ipv4-free-pool-depleted

[24] HARMOUSH, Ed. Why NAT?. *Practical Networking* [online]. 2021, 15. 12. 2021 [cit. 2022-12-02]. Dostupné z: https://www.practicalnetworking.net/series/nat/why-nat/

[25] RFC 1918. *RFC* [online]. Silicon Graphics, 1996 [cit. 2022-12-07]. Dostupné z: https://www.rfc-editor.org/rfc/rfc1918

[26] TYSON, Jeff. How Network Addresses Translation Works. *How stuff works* [online]. [cit. 2022-12-16]. Dostupné z: https://computer.howstuffworks.com/nat.htm

[27] SATRAPA, Pavel. *IPv6* [online]. 4. aktualizované a rozšířené vydání. Praha: CZ.NIC, 2019 [cit. 2022-12-07]. ISBN 978-80-88168-46-1. Dostupné z: https://knihy.nic.cz/files/edice/IPv6-2019.pdf

[28] Individuální adresy. *IPv6* [online]. 2019, 24.9.2019 [cit. 2022-12-11]. Dostupné z: https://www.ipv6.cz/cs/individualni_adresy_unicast

[29] KRISHNAN, Suresh. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. *RFC* [online]. 2007, Září 2007 [cit. 2022-12-12]. Dostupné z: https://www.rfc-editor.org/rfc/rfc4941

[30] How to create Multiple IP Address to Single Network interface. *Linux Help* [online]. [cit. 2023-05-13]. Dostupné z: https://www.linuxhelp.com/how-to-create-multiple-ip-address-to-single-network-interface?utm_content=cmp-true

[31] NIDECKI, Tomasz. IP Spoofing. *Invicti* [online]. [cit. 2022-11-27]. Dostupné z: https://www.invicti.com/learn/mitm-ip-spoofing-ip-address-spoofing/

[32] Maximum number of interface aliases. *Red hat customer portal* [online]. 2015, 27. 5. 2015 [cit. 2022-11-27]. Dostupné z: https://access.redhat.com/solutions/40500

[33] TIKHANSKI, Dmitri. IP Spoofing With JMeter: How to Simulate Requests from Different IP Addresses. *BlazeMeter* [online]. 2021 [cit. 2022-12-08]. Dostupné z: https://www.blazemeter.com/blog/ip-spoofing-jmeter

[34] What Is Load Balancing? *NGINX* [online]. [cit. 2022-12-07]. Dostupné z: https://www.nginx.com/resources/glossary/load-balancing/

[35] Very long startup time on Ubuntu Server (network configuration). In: *Ask ubuntu* [online]. 2021 [cit. 2023-05-13]. Dostupné z: https://askubuntu.com/questions/1321443/very-long-startup-time-on-ubuntu-server-network-configuration

[36] April 2023 Web Server Survey. *NetCraft* [online]. 2023, 24. 4. 2023 [cit. 2023-04-29]. Dostupné z: https://news.netcraft.com/archives/category/web-server-survey/

[37] What is the SSL Certificate Common Name?. *DNS simple* [online]. [cit. 2023-05-10]. Dostupné z: https://support.dnsimple.com/articles/what-is-common-name/

[38] Free SSL / TLS. *CloudFlare* [online]. CloudFlare, 2022 [cit. 2022-11-28]. Dostupné z: https://www.cloudflare.com/ssl/

[39] How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 22.04. *Digital Ocean* [online]. 2022, 26. 4. 2022 [cit. 2022-11-20]. Dostupné z: http://bitly.ws/EKZJ, zkráceno

[40] How To Increase Max Connections in Apache. *Ubiq* [online]. [cit. 2022-12-05]. Dostupné z: https://ubiq.co/tech-blog/increase-max-connections-apache/

[41] REES, Lucas. Comparing Linux Apache Prefork vs Worker MPMs. *Linux Config* [online]. 2023, 9. 2. 2023 [cit. 2023-02-11]. Dostupné z: https://linuxconfig.org/comparing-linux-apache-prefork-vs-worker-mpms

[42] RFC editor. *RFC 5246* [online]. 2008, 2008 [cit. 2023-05-13]. Dostupné z: https://www.rfc-editor.org/rfc/rfc5246.html#appendix-E

Seznam symbolů a zkratek

AFRINIC	African Network Information Centre
API	Application Programming Interface
APNIC	Asia Pacific Network Information Center
ARIN	American Registry for Internet Numbers
CIA	Confidentiality, Integrity, Availability
CNN	Cable News Network
CSV	Comma-separated values
DDoS	Distributed Denial Of Service
DDR3	Double Data Rate 3
DHCP	Dynamic Host Configuration Protocol
DHCPv4	Dynamic Host Configuration Protocol version 4
DHCPv6	Dynamic Host Configuration Protocol version 6
DNS	Domain Name System
DUID	DHCP Unique Identifier
EUI-64	64bitový Extended Unique Identifier
FTP	File Transfer Protocol
GUI	Graphic User Interface
HSTS	HTTP Strict Transport Security
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IMAP	Internet Message Access Protocol
IoT	Internet of Things
IP	Internet Protocol

IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO/OSI	International Standards Organization Open Systems Interconnection
ISP	Internet service provider
IT	Information technology
JDBC	Java Database Connectivity
JMS	Java Messaging Services
LACNIC	Latin America and Caribbean Network Information Centre
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MD5	Message-Digest algorithm
MIT	Massachusetts Institute of Technology
MPM	Multi-Processing Modules
NAT	Network Address Translation
PPS	Packets Per Second
RFC	Request For Comments
RIPE NCC	Réseaux IP Européens Network Coordination Centre
RIR	Regional Internet registry
RPS	Requests Per Second
RSA	iniciály autorů Rivest, Shamir, Adleman
SFP+	Small Form-factor Pluggable Plus
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured query language

SSL	Secure Sockets Layer
SSL/TLS	Secure Sockets Layer/Transport Layer Security
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USA	United States of America
VUT	Vysoké učení technické v Brně

7 Příloha

Elektronická příloha obsahuje následující soubory:

- `HttpsFloodGui.java` – třída vytvářející grafické uživatelské rozhraní, obsahuje metody pro generování rozsahů IP adres a skripty pro přiřazení adres na síťové rozhraní.
- `HttpsFloodSampler.java` – obsahuje samotnou logiku útoku.
- `Ddos.jar` – zkompilovaný kód pro vložení do JMeteru.

Návod k instalaci a spuštění testu je popsán v kapitole 6.